



# Autonomous System Subversion Tactics

*Prototypes and Recommended Countermeasures*

August 2022

*CT-22IN110402*

Idaho National Laboratory

*Chris Spirito  
Justin Gomez  
Tori Simon*

Georgia Institute of Technology

*Dr. Fan Zhang  
Patience Lamb  
Tanya Sharma  
Avery Skolnick*

Idaho State University

*Dr. Leslie Kerby  
Dr. Pedro Mena  
Emily Elzinga  
Eric Hill  
Kallie McLaren*



#### **DISCLAIMER**

This information was prepared as an account of work sponsored by an agency of the U.S. Government. Neither the U.S. Government nor any agency thereof, nor any of their employees, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness, of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the U.S. Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the U.S. Government or any agency thereof.

# Autonomous System Subversion Tactics

**CT-22IN110402**

**Idaho National Laboratory**

*Chris Spirito  
Justin Gomez  
Tori Simon*

**Georgia Institute of Technology**

*Dr. Fan Zhang  
Patience Lamb  
Tanya Sharma  
Avery Skolnick*

**Idaho State University**

*Dr. Leslie Kerby  
Dr. Pedro Mena  
Emily Elzinga  
Eric Hill  
Kallie McLaren*

**August 2022**

**Idaho National Laboratory  
Idaho Falls, Idaho 83415**

**<http://www.inl.gov>**

**Prepared for the  
U.S. Department of Energy  
Office of Nuclear Engineering  
Under DOE Idaho Operations Office  
Contract DE-AC07-05ID14517**



*Page intentionally left blank*

## SUMMARY

One of the fielding requirements for Advanced and Small Modular Reactors (AR/SMR) is the ability to support remote and autonomous operations. Autonomous Control Systems (ACS) are found on platforms such as Autonomous Space Vehicles, Cruise Missiles, and advanced driver-assistance systems. Each of these ACS implementations depends upon a set of decision support subsystems responsible for supporting Autonomous Mission Managers (names vary based upon field and author preferences). These Autonomous Mission Managers receive inputs from system sensors (e.g., LIDAR collection from an automobile travelling down a street; transients from a nuclear reactor), and perform a set of classifications (e.g., Red Traffic Light; Small Pedestrian at 10m; Load Rejection; Single Coolant Pump Trip), and then use these classifications in combination with recommendation algorithms to achieve platform goals (e.g., Stop the Vehicle at the Traffic Light, Avoid the Small Pedestrian; Trip the Reactor to prevent a Safety Event).

The design, implementation, and fielding of an ACS capability will alter the cyber-attack surface such that existing risk management plans will need to be updated to include how to protect and defend against data-science and decision-support-system attack classes. These attack classes would include protection of the design and training environments where algorithm selection and testing and training data would be obvious attack vectors. These attack classes would also require an informed set of detection and response procedures to identify anomalous behaviors and document best practices for anomaly assessment and vulnerability mitigation and remediation.

Last year we published a Cyber Threat Assessment Methodology for Autonomous and Remote Operations for AR/SMRs along with a companion publication on Cyber Attack and Defense Use Cases. The focus of the methodology was on describing and enumerating ACS processes, components, and functions such that security engineers could: evaluate subversion options against the target; identify threat actor attributes and capabilities derived from each subversion option; and identify security controls and response countermeasures. The Use Cases document offered detailed methodology examples including an assessment of a Military Base SMR, an Autonomous System Decision Loop, and implementation of AR/SMR Machine Learning algorithms. Our proposal at the end of last year was to focus on implementation of subversion prototypes related to the last Use Case area: AR/SMR Machine Learning (ML) Algorithms.

We included six attack scenarios in our Use Cases paper: a Poisoning Attack against ML functions implemented using an FPGA; a Trojaning Attack against ML classifiers exploiting the excitability of Nuclear Engineers; a Backdooring Attack against ML Training environments to ensure persistence of an attack vector; a False Positive Evasion Attack against multi-factor Access Control Systems using clever inputs; an Inference Attack against ML models by an Insider with access to the Operational environment; and an Adversarial Reprogramming Attack against a Material Access Control Video Surveillance System. At the beginning of this year these six attack scenarios were provided to our research teams at Georgia Tech and Idaho State University and each team successfully implemented a subversion attack against a ML implementation to include transient misclassifications. While this is a notable outcome from this type of research, this paper offers the reader insight into not only how to structure and execute these types of attacks, but into the thought process behind how the researcher investigated the problem space, performed initial algorithm implementation, and the trial-and-error behind arriving at the successful subversion prototypes. We include in this paper a set of associated Scenarios on how these subversion prototypes could be implemented and an initial set of guidance for AR/SMR architects, Nuclear Regulators, and Cyber Defenders to implement awareness and defense capabilities into their current operational portfolios.

*Page intentionally left blank*

# CONTENTS

SUMMARY .....	iv
ACRONYMS .....	xvii
1. Introduction .....	1
1.1 Notes from the Primary Investigator .....	2
1.2 Guide to Reading and Using this Report .....	2
2. Machine Learning Subversion Attack Research .....	5
2.1 Attacks using Digital Twin Testbed .....	5
2.1.1 Original Proposition .....	5
2.1.2 Initial Scenario .....	5
2.1.3 Asherah Development .....	5
2.1.4 GPWR Development .....	8
2.2 Inference Attacks .....	12
2.2.1 Original Proposition .....	12
2.2.2 Initial Scenario .....	17
2.2.3 Research Iterations .....	17
2.2.4 Research Artifacts and Findings .....	21
2.2.5 Inference Attack Research Conclusions .....	30
2.2.6 DIY ML Subversion .....	34
2.3 Trojan Attacks .....	42
2.3.1 Original Proposition .....	42
2.3.2 Initial Scenario .....	45
2.3.3 Research Iterations .....	46
2.3.4 Research Artifacts and Findings .....	47
2.3.5 Research Conclusions .....	58
2.4 Evasion Attacks .....	59
2.4.1 Original Proposition .....	59
2.4.2 Initial Scenario .....	62
2.4.3 Research Iterations .....	62
2.4.4 Research Artifacts and Findings .....	69
2.4.5 Research Conclusions .....	72
2.5 Linear SVC and Adversarial Label Flip Attack (ALFA) .....	74
2.5.1 Original Proposition .....	74
2.5.2 Research Iterations .....	74
2.5.3 Research Conclusions .....	83
2.6 Attacks using the Hardware in the Loop Testbed .....	85
2.6.1 Environment Design .....	85
2.6.2 Establishing the Testbed .....	90

2.6.3	Simulating Cyber Attacks .....	96
2.6.4	Future Attacks .....	98
3.	Threat Hunting for ML Subversion.....	99
3.1	Kestrel.....	99
3.1.1	Hunt Flows.....	100
3.1.2	Applying Hunt Flows in Scenarios .....	103
3.2	ODNI Cyber Threat Framework .....	104
3.3	MITRE ATT&CK.....	105
4.	Scenarios .....	106
4.1	Foreign Adversary Scenario.....	106
4.2	Trojan Scenario .....	110
4.3	DDoS Scenario.....	113
4.4	Backdoor Scenario .....	116
4.5	False Data Injection Scenario.....	119
4.6	Insider Scenario.....	123
4.7	Adversarial Label Flip Scenario.....	128
4.8	Supply Chain Attack Scenario .....	132
5.	Recommended Countermeasures .....	135
5.1	Guidance for Advanced Reactor and SMR Architects.....	135
5.1.1	Protect the Data Science Ecosystem .....	135
5.1.2	Protect the Model Implementations .....	136
5.1.3	Establish a Data Science Subversion Capability.....	137
5.2	Guidance for Nuclear Regulatory Agencies .....	137
5.2.1	Provide Data Science Training and Education.....	137
5.2.2	Implement a Model Design, Validation, and Fielding Oversight Process .....	137
5.2.3	Establish a Data Science Threat Actor Capability Tracking Program .....	137
5.2.4	Expand the Incident Analysis Database to include Data Science Subversion Events.....	138
5.3	Guidance for Licensee and Operator Security Engineers .....	138
5.3.1	Updating Security Controls to include Data Science Attack Classes .....	138
5.3.2	Updating Incident Response Capabilities and Procedures for Data Science Subversion Attacks .....	138
6.	Subversion Research .....	140
6.1	Subversion Attack Overview .....	140
6.1.1	Motivation for Attack.....	140
6.1.2	Collecting Victim Information.....	140
6.1.3	Subversion Planning .....	142
6.1.4	Model Exploitation and Delivery of Subversion Effects .....	143

6.2	ML Subversion Research Program .....	143
6.3	ML Subversion Research Team .....	144
6.4	Curriculum for Subversion Education .....	146
6.4.1	Topics for Subversion Education .....	146
6.4.2	Idaho State University Classes .....	146
6.4.3	MIT Open Courseware .....	147
6.5	Road Map for Future .....	148
6.6	Research Team Capability Development Self-Assessment .....	148
6.6.1	Eric Hill, Idaho State University .....	148
6.6.2	Patience Lamb, Georgia Tech .....	149
6.6.3	Avery Skolnick, Georgia Tech .....	149
6.6.4	Tanya Sharma, Georgia Tech .....	150
6.6.5	Kallie McLaren, Idaho State University .....	151
7.	Research Laboratories and Datasets .....	152
7.1	iFan Laboratory at Georgia Tech .....	152
7.2	The CEADS Lab at Idaho State University .....	153
7.3	Online Datasets and Research Repositories .....	154
7.3.1	iFan Repositories .....	154
7.3.2	CEADS Repositories .....	156

## FIGURES

Figure 1: (Intro) Six Machine Learning Attack Scenarios.....	1
Figure 2: (DT) Cyber Threat Assessment Methodology.....	5
Figure 3: (DT) Cyber Threat Assessment Methodology POAMS .....	6
Figure 4: (DT) Autonomous Control System (ACS) Plant Level Digital Twin Design .....	6
Figure 5: (DT) Asherah Steam Generator Flow Characteristics .....	6
Figure 6: (DT) Asherah Steam Generator Cooling Tube Parameters .....	6
Figure 7: (DT) Linear-SVC Model Confusion Matrix for Secondary Generator Fault Detection.....	7
Figure 8: (DT) Steam Generator Outlet Temperature Regression Model Actual vs. Predicted .....	7
Figure 9: (DT) Steam Generator Regression Model Digital Twin RMSE.....	7
Figure 10: (DT) Asherah Auto Associative Neural Network Implementation .....	8
Figure 11: (DT) Steam Generator AANN Outlet Temperature RMSE (Testing).....	8
Figure 12: (DT) Autonomous Control System (ACS) Hardware-in-the-Loop Azure Integration.....	9
Figure 13: (DT) GPWR Component Level Digital Twin LinearSVC Model Predictions .....	9
Figure 14: (DT) GPWR Component Level Digital Twin K-NN Model Predictions .....	9
Figure 15: (DT) GPWR Plant Level Digital Twin AANN .....	10
Figure 16: (DT) Autonomous Control System Generalized Control Model (Strategy Selection) See Annex I for full size version of this Strategy Selection Flowchart. ....	10
Figure 17: (DT) GWPR Confusion Matrix for Random Forest Classifier.....	11
Figure 18: (IA) An example Machine Learning pipeline.....	16
Figure 19: (IA) Deep and Wide Neural Network (DaWWN) Victim Model Layer Configuration.....	19
Figure 20: (IA) The recurrent neural network that was copied. The number of output neurons are the number of points that are predicted. ....	20
Figure 21: (IA) Confusion Matrix on the prediction and test data from H2O. ....	26
Figure 22: (IA) Memory Usage Statistics while Running AutoKeras. An unfortunate drawback of AutoKeras is the amount of memory it takes to run. ....	26
Figure 23: (IA) Best Model from AutoKeras. Interestingly, there are only two convolutional layers.....	27
Figure 24: (IA) Prediction points on another one of the waves in the training dataset. As above, the orange point is the one predicted by TPOT and the red point is the one predicted by the victim model. The distance between the two is slightly less than the one in Figure 25. ....	28
Figure 25: (IA) Prediction points on a typical wave in the training dataset. The point predicted by TPOT is in orange and the point predicted by the victim model is in red. There is a glaring discrepancy between the two.....	28
Figure 26: (IA) Best model found by the KerasTuner library on the RNN prediction data .....	29

Figure 27: (IA) Sample sine wave from the victim model's training dataset. The red line is the series of points predicted by the victim model, and orange is the series of points predicted by KerasTuner.....	29
Figure 28: (IA) Another sample sin wave from the victim model's training dataset. The red line is the series of points predicted by the victim model, and orange is the series of points predicted by KerasTuner. This wave is like many others in the dataset in that it 'plays it safe' by vacillating around the mean because it isn't sure where the wave goes next. ....	30
Figure 29: (TA) Flowchart for the overall process of the attack.....	43
Figure 30: (TA) Flowchart for the Trojan generation process.....	45
Figure 31: (TA) MNIST Retraining Dataset.....	48
Figure 32: (TA) FashionMNIST retraining dataset .....	49
Figure 33: (TA) Asherah image of unmodified data.....	49
Figure 34: (TA) Steady state reactor data image of original data and trigger mix.....	50
Figure 35: (TA) Transient reactor data image of original data and trigger mix.....	50
Figure 36: (TA) Accuracy of each dataset after retraining before the inconsistency fix .....	52
Figure 37: (TA) Code to fix inconsistency .....	52
Figure 38: (TA) Accuracy of each dataset after retraining and after the inconsistency fix .....	53
Figure 39: (TA) Optimal number of features to target when classifying masks as transients .....	56
Figure 40: (TA) Optimal number of features to target when classifying masks as steady .....	56
Figure 41: (EA) Adversarial image of a cat.....	60
Figure 42: (EA) Normal cat image .....	60
Figure 43: (EA) Image from MIT paper demonstrating adversarial image .....	60
Figure 44: (EA) Illustration / demonstration of what the MIT picture is doing in Figure 42 .....	61
Figure 45: (EA) My functional implementation of FGSM .....	61
Figure 46: (EA) Normal classification of a Labrador picture .....	63
Figure 47: (EA) Image of the noise generated for the Labrador picture.....	63
Figure 48: (EA) Adversarially perturbed image with weight of .01 .....	63
Figure 49: (EA) Another perturbed image, but towards a different label .....	63
Figure 50: (EA) Normal classification of a corgi picture .....	63
Figure 51: (EA) Image of the noise the algorithm generated for the corgi picture.....	63
Figure 52: (EA) Adversarially perturbed corgi with weight of .1.....	63
Figure 53: (EA) Adversarially perturbed corgi with weight of .15.....	63
Figure 54: (EA) Default of classified 7.....	64
Figure 55: (EA) Adversarial image created using FGSM.....	64
Figure 56: (EA) Experiment with using the gradients, not the signed gradients .....	64
Figure 57: (EA) More experiments with using just the gradients .....	64



Figure 58: (EA) Normal classification of a cat picture .....	64
Figure 59: (EA) A perturbed picture of the cat .....	64
Figure 60: (EA) 7 perturbed towards 3 .....	65
Figure 61: (EA) 7 perturbed towards 4 .....	65
Figure 62: (EA) 7 perturbed towards 8 .....	65
Figure 63: (EA) 7 perturbed towards 9 .....	65
Figure 64: (EA) Airplane perturbed towards Auto .....	65
Figure 65: (EA) Airplane perturbed towards a bird .....	65
Figure 66: (EA) Airplane perturbed towards a cat .....	65
Figure 67: (EA) Airplane perturbed towards a dog .....	65
Figure 68: (EA) Best model architecture .....	66
Figure 69: (EA) CIFAR10 example of label table .....	66
Figure 70: (EA) Example of Display Method 1v .....	66
Figure 71: (EA) Example of Display Method 5 .....	66
Figure 72: (EA) GPWR Table of Predictions .....	67
Figure 73: (EA) 1% change to misclassify as normal-ops .....	67
Figure 74: (EA) Index 5 perturbed .....	68
Figure 75: (EA) Index 4689 perturbed .....	68
Figure 76: (EA) Index 4689 perturbed with slightly different step and epsilon .....	68
Figure 77: (EA) 7 perturbed towards 9 .....	69
Figure 78: (EA) 7 perturbed towards 8 .....	69
Figure 79: (EA) CIFAR10 table of adversarial predictions .....	70
Figure 80: (EA) Table of GPWR adversarial images .....	71
Figure 81: (EA) Decision tree model prediction for the original data (original label of "9") .....	71
Figure 82: (EA) Decision tree model prediction for the altered data (original label of "9") .....	71
Figure 83: (EA) Comparison between original and adversarial data (normalized) .....	71
Figure 84: (EA) Comparison between original and adversarial data (un-normalized) .....	71
Figure 85: (EA) Table of results of the 12 adversarial samples used on the k-NN model .....	72
Figure 86: (ALFA) First implementation of ALFA: Before vs after ALFA .....	75
Figure 87: (ALFA) Second Implementation of ALFA: Number of Label Flips vs. Training accuracy, Testing accuracy, Precision and Recall .....	76
Figure 88: (ALFA) Adversarial Label Flip Strategy (ALFA) on a dummy dataset .....	77
Figure 89: (ALFA) Working Principles behind k-NN .....	77
Figure 90: (ALFA) k-NN vs. SVM Performance (Training and Testing Accuracy, Precision, and Recall) .....	78

Figure 91: (ALFA) Applying ALFA on k-NN (Training and Testing Accuracy, Precision and Recall).....	78
Figure 92: (ALFA) Gradient descent attack on k-NN: Noise Example i.....	79
Figure 93: (ALFA) (ALFA) Gradient descent attack on k-NN: Noise Example ii.....	79
Figure 94: (ALFA) Results from Gradient Descent Attack against k-NN, k=2, Test Data (503 rows), Modified (59 rows).....	81
Figure 95: (ALFA) Results from Gradient Descent Attack against k-NN, k=3, Test Data (503 rows), Modified (62 rows).....	82
Figure 96: (ALFA) Results from Gradient Descent Attack against k-NN, k=5, Test Data (503 rows), Modified (68 rows).....	82
Figure 97: (ALFA) ALFA against Linear SVM: Understanding the correlation between Training Accuracy and Number of Optimizing Iterations.....	83
Figure 98. (HILT) General AANN design.....	85
Figure 99. (HILT) Set of parrots for classification. ....	86
Figure 100. (HILT) LinearSVC parrot classification.....	87
Figure 101. (HILT) KNN parrot classification. ....	87
Figure 102. (HILT) MLP regressor model used. ....	88
Figure 103. (HILT) Decision tree regression.....	89
Figure 104. (HILT) Strategy selection decision tree.....	90
Figure 105: (HILT) GPWR Testbed HMI Picture .....	91
Figure 106: (HILT) GPWR Testbed Block Configuration .....	91
Figure 107. (HILT) Asherah ACS environment. ....	91
Figure 108. (HILT) Test RMSEs of each of the Asherah variables.....	92
Figure 109. (HILT) Steam generator LinearSVC confusion matrix. ....	92
Figure 110. (HILT) Predicted steam generator 2 outlet temperature versus actual outlet temperature. ....	93
Figure 111. (HILT) Steam generator 2 outlet temperature error over time. ....	93
Figure 112. (HILT) GPWR testbed environment. ....	94
Figure 113: (HILT) GPWR AANN architecture .....	94
Figure 114: (HILT) Steam generator 1 k-NN confusion matrix .....	95
Figure 115: (HILT) Strategy selection random forest classifier confusion matrix .....	95
Figure 116: (HILT) GWPR Simulator HMI Steam Generator Logic Severed Connection.....	96
Figure 117: (HILT) FCV under normal conditions (100-75-100 Power Ramp).....	97
Figure 118: (HILT) FCV under attack (100-75-100 Power Ramp).....	97
Figure 119: (HILT) Steam Generator with a (100-75-100 Power Ramp).....	97
Figure 120: (HILT) Complex FDIA Attack with Oscillating Observable .....	98

Figure 121: (HILT) Georgia Tech Testbed Structure .....	98
Figure 122: (THunt) STIX Relationship Example.....	100
Figure 123: (THunt) Kestrel Variable FIND RELX.....	101
Figure 124: (THunt) Example of Kestrel JOIN Command.....	102
Figure 125: (KHF) Kestrel Hypothesis Hunt Flow Example.....	103
Figure 126: (KHF) Kestrel Hunt Step Object Relationships .....	103
Figure 127: (KHF) ODNI Cyber Threat Framework v4.....	104
Figure 128: (SR) ML Subversion Attack LifecycleC .....	133
Figure 129: (SR) ML Subversion Attack Lifecycle.....	143
Figure 130: (SR) ML Subversion Research Program Structure.....	143
Figure 131: (ML) Proposed Team Structure for Subversion ML Program.....	145
Figure 132: (SR) Factors contributing to Team Success .....	145
Figure 133: (SR) Background Courses ML Hackers .....	146
Figure 134: (ML) Adversarial Machine Learning Papers (2017-2022).....	148

## TABLES

Table 1:(IA) Binary Classification dataset from the Asherah Nuclear Reactor Simulator .....	14
Table 2: (IA) Test-Accuracy for White-Box Attack and Victim Pipeline on GPWR Prediction Data.....	21
Table 3: (IA) Execution statistics for the victim pipeline and different attack pipelines on the GPWR test data .....	22
Table 4: (IA) Hyperparameters for the Extra Trees Parts of the Copycat and Victim Models.....	22
Table 5: (IA) Most misclassified transients for each copycat model, and the number of times those transients were misclassified in the victim model. ....	23
Table 6: (IA) Model Execution Statistics for the Victim and Attack Models on the Asherah Dataset .....	23
Table 7: (IA) k-NN Hyperparameters for Attack Pipelines and the Victim Model.....	24
Table 8: (IA) Execution statistics for the victim model and different attack pipelines on the GPWR test data. ....	24
Table 9: (IA) Execution statistics for the victim model and different attack pipelines on the GPWR test data .....	25
Table 10: (IA) The test accuracy and execution time of the victim pipeline compared to the average accuracy and execution time of the attack pipelines. The accuracy of the attack pipelines is a measure of how perfect a copy TPOT was able to make.....	31
Table 11: (IA) The test accuracy and execution time of the victim pipeline compared to the average accuracy and execution time of the attack pipelines .....	31
Table 12: (IA) Attack accuracy and execution time for an average attack iteration. The accuracy of the attack pipelines is a measure of how perfect a copy TPOT was able to make .....	31
Table 13: (IA) Attack accuracy and execution time for an average attack iteration of the simple version of TPOT .....	32
Table 14: (IA) Attack accuracy and execution time for an average attack iteration of the complex version .....	32
Table 15: (IA) The attack test and prediction accuracy and and execution time of AutoKeras attack on the deep and wide neural network.....	32
Table 16: (IA) Execution statistics for TPOTRegressor for the attack. The best pipeline that the TPOT regressor found had a LassoLarsCV regressor as the primary model with a k-NN classifier and several preprocessors. As always, the $R^2$ value was obtained from the test and prediction data from the recurrent neural network. Its high value shows a strong correlation between the predicted values in the copycat model and those from the victim model. ....	33
Table 17: (IA) Execution statistics on the attack of the recurrent neural network with KerasTuner. The low MSE means that many of the predicted values from KerasTuner matched the predicted points from the RNN.....	33
Table 18: (IA) Metrics and attack execution times for each of the best attacks .....	33
Table 19: (TA) Results of MNIST Attack .....	48

Table 20: (TA) Results of FashionMNIST Attack.....	48
Table 21: (TA) Results of attack on image format data when classifying masks as transients .....	49
Table 22: (TA) Results of attack on image format data when classifying masks as steady state .....	50
Table 23: (TA) Current results of flattened data in predicting masked data as transients .....	50
Table 24: (TA) Current results of flattened data in predicting masked data as steady state .....	51
Table 25: (TA) Trial table of important values for program to help determine where inconsistency .....	51
Table 26: (TA) Trial table of important values for program after inconsistency fix .....	52
Table 27: (TA) Masks being classified as transients results random target features .....	53
Table 28: (TA) Random feature names for 5 different trials classifying masked data as transients.....	54
Table 29: (TA) Masks being classified as steady state results random target features .....	54
Table 30: (TA) Random feature names for 5 different trials classifying masked data as steady.....	55
Table 31: (TA) Feature names for each number of features being targeted .....	55
Table 32: (ALFA) Verification of implementation using dummy dataset.....	76
Table 33: (ALFA) Gradient descent attack on k-NN Input/Output Values .....	79
Table 34: (ALFA) Gradient Descent Attack using get_difference() function .....	80
Table 35: (ALFA) Dictionary Keys and Row Changes beyond threshold .....	81
Table 36: (ALFA) Iterations vs. Training Accuracy.....	83

*Page intentionally left blank*

## ACRONYMS

AANN	Auto Associative Neural Network
ACS	Autonomous Control System
ALFA	Adversarial Label Flip Attack
ANS	American Nuclear Society
ANSYS	Ansys Corporation
API	Application Programming Interface
AR	Advanced Reactors
AS	Autonomous System
AWS	Amazon Web Services
BFS	Breadth First Search
BOP	Balance of Plant Systems
BOL	Beginning of Life
CCTV	Closed Circuit Television System
CDC	Centers for Disease Control
CEADS	Computational Engineering and Data Science
CFAA	Computer Fraud and Abuse Act
CIFAR	Canadian Institute for Advanced Research (creator of CIFAR-10 dataset)
CLDAP	Connection-less Lightweight Directory Access Protocol
CNN	Convolutional Neural Network
COM	Component Object Model
COMPSAC	Computers, Software & Applications in an Uncertain World
CPU	Central Processing Unit
CRP	Coordinated Research Project
CSOC	Cyber Security Operations Center
CSV	Comma Separated Values (file format)
DA	Data Access
DARPA	Defense Advanced Research Projects Agency
DaWNN	Deep and Wide Neural Network
DCOM	Distributed Component Object Model
DOE	Department of Energy
DoS	Denial of Service
DDoS	Distributed Denial of Service
DNN	Deep Neural Network

DNS	Domain Name System
DT	Digital Twin / Decision Trees
EA	Evasion Attacks
EIA	US Energy Information Administration
EOT	Expectation Over Transformation
EU	European Union
FCV	Feed Water Cooler
FDIA	False Data Injection Attack
FGSM	Fast Gradient Sign Method
FOIA	Freedom of Information Act
FPGA	Field Programmable Gate Array
GPU	Graphical Processing Unit
GPWR	Generic Pressurized Water Reactor
GSE	GSE Corporation
GT	Georgia Institute of Technology
HiL	Hardware-in-the-Loop
HILT	Hardware-in-the-Loop (alternate)
HMI	Human Machine Interface
IBM	IBM Corporation
IA	Inference Attack
IAEA	International Atomic Energy Agency
ICS	Industrial Control Systems
ICT	Information and Communication Technology
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
INL	Idaho National Laboratory
ISU	Idaho State University
IT	Information Technology (see also ICT)
KHF	Kestrel Hunt Flow
KNN	k-Nearest Neighbors
LBFGS	Limited Broyden-Fletcher-Goldfarb-Shanno
LIDAR	Light Detection and Ranging
LOCA	Loss-of-Coolant Accident
LOOP	Loss of Offsite Power



Lr	Learning Rate
LSTM	Long-Short Term Memory
M	Masked test set
MD	Mixed Dataset
MFV	Major Feedwater Valve
MI	Model Inversion
MIT	Massachusetts Institute of Technology
MiTM	Man in the Middle
ML	Machine Learning
MLP	Multilayer Perceptron
MNIST	Modified National Institute of Standards and Technology
MSE	Mean Squared Error
MSIV	Main Steam Isolation Valves
NaN	Not a Number
NE	Nuclear Engineering
NGINX	NGINX Reverse Proxy
NN	Neural Network
NM	No Mask
NPP	Nuclear Power Plants
NRC	Nuclear Regulatory Commission
ODNI	Office of the Director of National Intelligence
OLE	Object Linking and Embedding
OPSEC	Operational Security
OPC	Open Platform Communications
OPC DA	Open Platform Communications Data Access
OPC UA	Open Platform Communications Unified Architecture
OSINT	Open-Source Intelligence
OT	Operational Technology
PCAP	Packet Capture (file format)
PID	Process Identifier
PI	Raspberry PI (low-cost computational platform)
PLC	Programmable Logic Controller
POAMS	Plant Operations Autonomous Management System
POC	Proof of Concept
POSIX	Portable Operating System Interface

PWR	Pressurized Water Reactor
RBF	Radial Basis Function
RCS	Reactor Coolant System
RE	Reverse Engineering
RF	Random Forest
RFE	Recursive Feature Elimination
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
RX	Reactor Power
SCRAM	Acronym for Reactor Trip
SELU	Scaled Exponential Linear Units
SGD	Stochastic Gradient Descent
SG	Steam Generator
SMR	Small Modular Reactors
SOC	Security Operations Center
SQL	Structured Query Language
SR	Subversion Research
SRO	Senior Reactor Operator
STIX	Structure Threat Intelligence Exchange
SVC	Support Vector Classification
SVM	Support Vector Machines
TA	Trojan Attack
TPOT	Tree Based Pipeline Optimization Tool
TTP	Tactics, Techniques, and Procedures
TV	Target value
UA	Unified Architecture
URL	Uniform Resource Locator
VM	Virtual Machine
WSC	Western Services Corporation
XGB	XG Boost (data science term)
XLS	Microsoft Excel Spreadsheet

*Page intentionally left blank*

# Autonomous System Subversion Tactics

## 1. Introduction

Last year we published a Cyber Threat Assessment Methodology for Autonomous and Remote Operations for AR/SMRs along with a companion publication on Cyber Attack and Defense Use Cases. The focus of the methodology was on describing and enumerating Autonomous Control System (ACS) processes, components, and functions such that security engineers could: evaluate subversion options against the target; identify threat actor attributes and capabilities derived from each subversion option; and identify security controls and response countermeasures. The Use Cases document offered detailed methodology examples including an assessment of a Military Base SMR, an Autonomous System Decision Loop, and implementation of AR/SMR Machine Learning algorithms. Our proposal at the end of last year was to focus on implementation of subversion prototypes related to the last Use Case area: AR/SMR Machine Learning (ML) Algorithms.

We included six attack scenarios in our Use Cases paper: a Poisoning Attack against ML functions implemented using an FPGA; a Trojaning Attack against ML classifiers exploiting the excitability of Nuclear Engineers; a Backdooring Attack against ML Training environments to ensure persistence of an attack vector; a False Positive Evasion Attack against multi-factor Access Control Systems using clever inputs; an Inference Attack against ML models by an Insider with access to the Operational environment; and an Adversarial Reprogramming Attack against a Material Access Control Video Surveillance System.

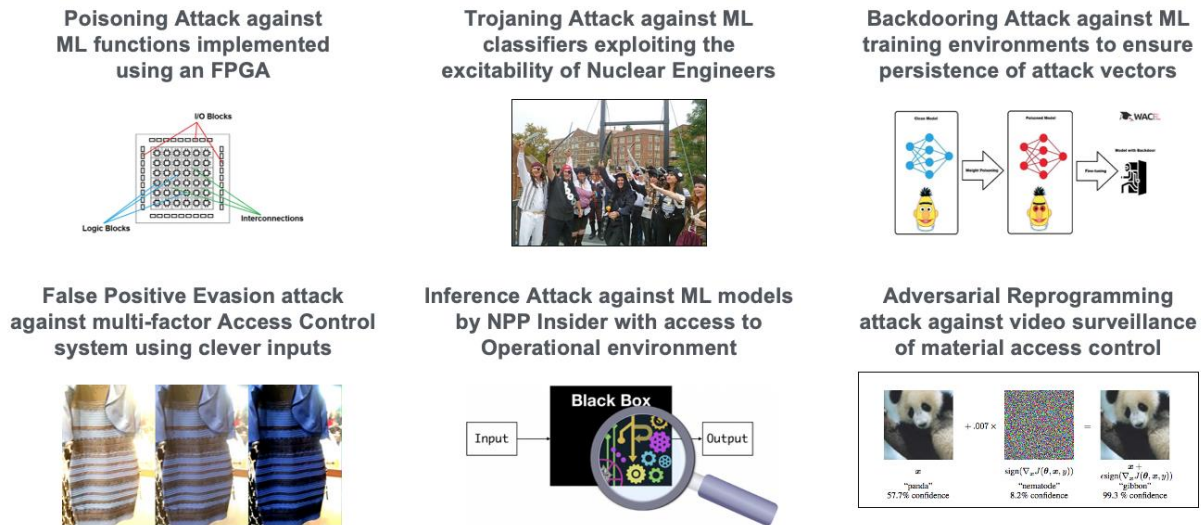


Figure 1: (Intro) Six Machine Learning Attack Scenarios

At the beginning of this year these six attack scenarios were provided to our research teams at Georgia Tech and Idaho State University and each team successfully implemented a subversion attack against a ML implementation to include transient misclassifications. While this is a notable outcome from this type of research, this paper offers the reader insight into not only how to structure and execute these types of attacks, but into the thought process behind how the researcher investigated the problem space, performed initial algorithm implementation, and the trial-and-error behind arriving at the successful subversion prototypes. We include in this paper a set of associated Scenarios on how these subversion prototypes could be implemented and an initial set of guidance for AR/SMR architects, Nuclear Regulators, and Cyber Defenders to implement awareness and defense capabilities into their current operational portfolios.

## 1.1 Notes from the Primary Investigator

This research and development project was executed by a small team from Idaho National Laboratory supported by research teams from Georgia Tech and Idaho State University. The technical research and development were primarily performed by undergraduate students from each of these organizations and institutions with guidance and support provided by graduate students, university professors, and the PI. The technical findings presented in Section 2, threat hunting details in Section 3, and scenarios presented in Section 4 were primarily authored by the students with a request to capture their learning process such that we could track their acquisition of concepts and how they worked through (and around) the challenges they encountered along the way. The editing I provided for each of these sections was limited to refining concepts that were not quite clear enough and adding supporting details where helpful relevant to the problem space we are working within. An attempt was made to leave the researchers primary voice intact as each of their work was uniquely interesting to follow and hear about during our weekly research meetings. The scenarios are good examples of how researchers who are not necessarily experts in the operational spaces we exist within such as Nuclear Power Plants and Research Reactors view the opportunity to apply their capabilities in the real world. While the event progression in many cases may not be of a high likelihood to succeed, these events do offer good starting points where if revised, viable alternatives could be swapped in to bring the scenario closer to reality. That being said, the technical capability being applied is still of concern to most every organization employing these technologies so do embrace your imagination as you wander through each of these worlds as we hope to inspire you to think about the application of subversion methods to your problem and operational space a bit differently.

## 1.2 Guide to Reading and Using this Report

This paper contains six technical sections and three annexes. We offer a summary of each section and annex so the reader may find an immersion pathway that best fits their objectives and technical background.

### Section 2: Machine Learning Subversion Attack Research

Each subsection of Section 2 contains an example of a Machine Learning Subversion Attack Methodology. The sections all share a common structure where the researcher starts with an explanation of what they believe the problem is that they were assigned to work on. Most of our researchers (ML Hackers) did not have any experience with subversion research or applications so we expected a gap between what they believed the problem was that they were solving and what the problem they eventually solved for. In addition to the *Original Proposition* each ML Hacker included an initial scenario sourced from the ones described earlier in this section and then a progression of research artifacts that illustrate their approach to solving this challenge. Each ML Hacker concludes their technical section with a set of artifacts and lists of findings and research conclusions. These can be read through all at once or one at a time with no preference given for order.

### Section 3: Threat Hunting for Machine Learning Subversion

The usefulness of this report is predicated on how much actionable guidance is provided. To this end we established as a project goal to think about these two questions: *If I were a ML Hacker conducting a lifecycle attack from reconnaissance through effect delivery, what would my footprint look like? How could a cyber-defender detect my actions and where would those detections exist across the attack lifecycle?* Section 3 introduces the reader to three concepts that are applied throughout the Scenarios in Section 4 and are focused on the assertion that Threat Hunting needs to be a consideration for environments that implement data science and machine learning algorithms into their operational architectures. In Section 3

the reader will learn about a threat hunting tool named Kestrel, about the Cyber Attack Lifecycle we chose to use within our scenarios developed by the US Office of the Director of National Intelligence (ODNI), and a taxonomy of tactics, techniques, and procedures (TTPs) developed and maintained by The MITRE Corporation. This section is a must-read before diving into the Scenarios as it will make the annotations provided alongside each scenario easily understandable.

#### **Section 4: Scenarios**

The goal of the scenarios section is to provide a mechanism for implementation of each of the ML Subversion techniques. Some of the scenarios are directly linked by name such as the Trojan and Adversarial Flip Attacks and scenarios by the same name. The other scenarios include the attacks within a broader theme such as an Insider performing an Inference Attack against an ML Model implementation or an Insider performing a sabotage attack using a Trojan and Evasion subversion attack capability. Each Scenario follows a common format starting a Scenario Overview, List of Characters, and Equipment that will be used or interacted with by the characters. This is followed with a backstory in the background section and then three sections that cover the scenario stages: *Event*, *Attack*, and *Discovery*. As the scenario progresses through each of these stages, ODNI and TTP mappings are provided. At the end of each scenario are a set of Kestrel Threat Hunting Rules provided to illustrate what a Threat Hunter may look for if this type of attack was executed. The reader is welcome to start with a scenario and then jump back to the attack research or consume the research first and then the scenario for additional context.

#### **Section 5: Recommended Countermeasures**

The Recommended Countermeasures section is the derived guidance from our attack research and scenario development. It is separated into three subsections: Guidance for Advanced Reactor and SMR Architects and Vendors; Guidance for Nuclear Regulatory Agencies; and Guidance for Security Engineers. Our goal here was to include actionable guidance for each of these communities such as ensuring that there is a process in place for validation of model testing and training data and how to include data science capabilities into your cyber-security capability development lifecycle. We provide for each recommendation a short description on proposed implementation.

#### **Section 6: Subversion Research**

The Subversion Research section is focused on how to establish these capabilities whether you are an individual interested in becoming an ML Hacker or you are an organization interested in establishing an offensive or defensive subversion attack research program. The first subsection includes a description of a subversion attack lifecycle followed by recommendations on establishing a research program and team. We include a curriculum for subversion education from courses our ML Hackers have completed to those available in OpenCourseWare collections. One of the more unique subsections of this report is the Research Team Capability Development Self-Assessments where most of the ML Hackers on our team evaluated their knowledge and skills at the beginning and end of their work across competency areas that they agreed would represent their progress.

#### **Section 7: Research Laboratories and Datasets**

In this section we provide short write-ups on each of the research laboratories that have supported this project in FY22 along with a description of their research portfolios. We also include a set of references to the online datasets and research repositories that are available for ML Hackers to use as part of their own research. We have included data science notebooks within each of these repositories for those scientists interested in hands-on immersion into this problem space.

### **Annex I: Autonomous Control System Generalized Control Model (Strategy Selection)**

Annex I is a larger view of the Autonomous Control System Generalized Control Model. There is no additional content in this section.

### **Annex II: MITRE ATT&CK TTP References**

The Scenarios in Section 4 contain references to MITRE ATT&CK TTPs. While the preferred method of viewing these TTPs is via the MITRE ATT&CK website, we understand that some readers will prefer a printed copy and so we include this Annex with high-level details on the TTPs we have referenced.

### **Annex III: Essential Reading**

Our final Annex includes essential reading in the field of Adversarial Machine Learning. These are commonly referenced books and papers to give the reader a head-start on their ML Hacker learning adventure.

## 2. Machine Learning Subversion Attack Research

### 2.1 Attacks using Digital Twin Testbed

The research presented in this section was authored by Patience Lamb (GT) with edits provided.

#### 2.1.1 Original Proposition

The original proposition was to create a digital twin testbed using a reactor simulator coupled with a digital twin environment to simulate reactor subsystems and create a control system to test subversion techniques on including against the digital twin environment using machine learning.

#### 2.1.2 Initial Scenario

Initially the idea was to conduct mainly false data injections into the Asherah simulator into the training data. The assumption was the machine learning models would break significantly under a small change to the training data, indicating severe volatility in the training data.

In November, the team started developing a general idea of reachable milestones and development pathways. The goal was to be done developing the test environment in February and collecting data and training models until May where we would focus the remainder of the time conducting cyber-attacks. The initial idea was to use the Asherah reactor, connected to Azure through Sandia's ManiPIO and into Azure Digital Twins.

#### 2.1.3 Asherah Development

Initially, the team went through the Cyber Threat Assessment methodology to determine processes and components to look at for a cyber threat assessment. Figure 2 describes the general process of assessing a cyber threat within autonomous systems.

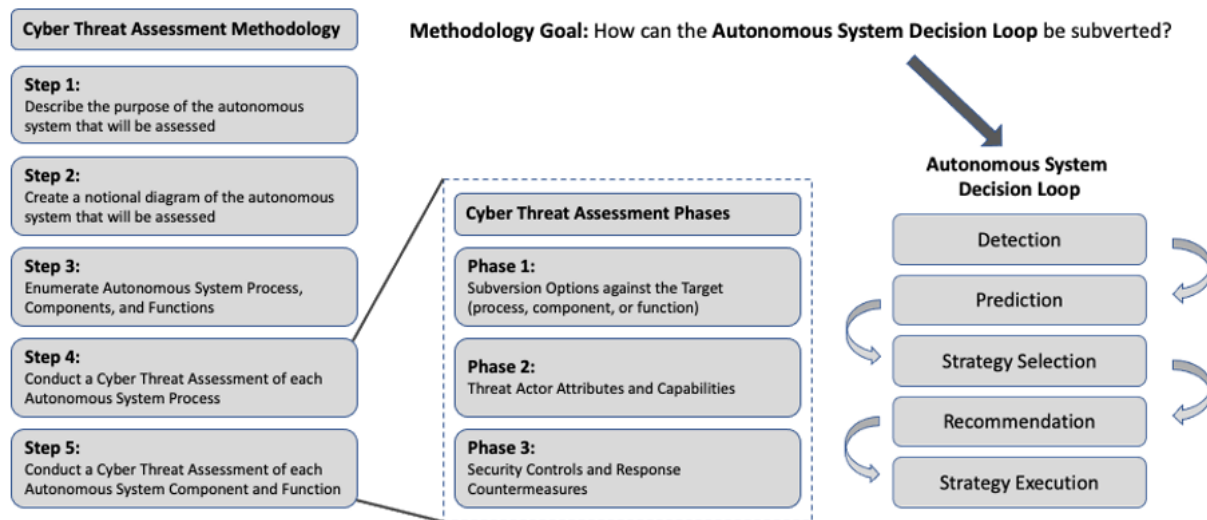


Figure 2: (DT) Cyber Threat Assessment Methodology

The attack outline collected during this assessment led the team to decide that an autonomous control system (ACS) which controls plant subsystems would be a good target to focus on, especially as threat actors would likely choose this digital asset as a high-value attack target. Specifically, it was decided to focus on the steam generator and condenser components that are controlled through ACS machine learning models.



Figure 3 shows the ACS (circled) laid out in a notational diagram and the relation to all other reactor systems and functions. The ACS was broken down into four components: Asherah, component level digital twins, the knowledge base, and a plant level digital twin. Figure 4 shows how the ACS would interact with Asherah and the plant level digital twin, where real-time process data is passed into the plant level digital twin and ACS and a recommendation or execution would be extracted according to the process data.

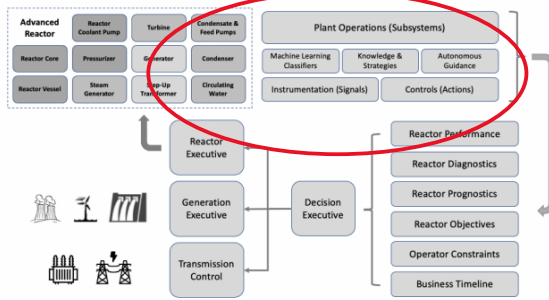


Figure 3: (DT) Cyber Threat Assessment Methodology POAMS

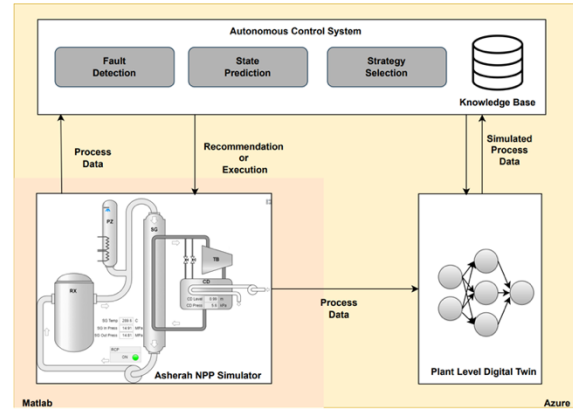


Figure 4: (DT) Autonomous Control System (ACS) Plant Level Digital Twin Design

### 2.1.3.1 Data Collection

Most of the January and February months were spent collecting training data from Asherah and producing preliminary models of the ACS. For normal operation Asherah was run at different power levels by reducing power in 5% increments every 1000 seconds from 110% to 40%. Power was also fluctuated using a sinusoidal wave as the power from 100% to 70%. Steam tube generator faults (degradation/plugged tubes) cause a reduction of cross-sectional flow decreases around ~0 to 2% for each tube plugged. To collect steam generator fault data, Asherah was run with a lower steam generator tube flow area. Figure 5 shows the code within the Asherah MATLAB code base manipulated to simulate steam generator tube plugging.

```
% Flow characteristics
% From NEA/NSC/DOC(99)8
% produces Dp = 0.21 MPa for SG rated operation
SGK =PRI_RFlow/sqrt(14.903e6 - 14.840e6);
```

Figure 5: (DT) Asherah Steam Generator Flow Characteristics

Another fault was created by simulating fouling in condenser tubes. This was done by reducing the cross-sectional area in the condenser tubes in Asherah. Figure 6 exhibits the condenser tube parameters, where *rtb* was reduced to simulate fouling.

```
%% Cooling tubes
% For each tube
xtb = 10;           % tube length [m]
rtb = 0.08;         % tube internal radius [m]
Rtb = 0.09;         % tube external radius [m]
```

Figure 6: (DT) Asherah Steam Generator Cooling Tube Parameters

Three separate machine learning models were created to aid in the digital twinning process: two component level digital twins and one plant level digital twin.

### 2.1.3.2 Component Level Digital Twins

The best classification model was a linear support vector classification (LinearSVC) with a testing accuracy and training accuracy of 1.0. 85 steam generator 2 tags were chosen to predict if the second steam generator was undergoing a fault or not. Figure 7 shows the confusion matrix for the LinearSVC model, where the model exhibited no false positives or false negatives when differentiating a transient from a normal operation from a fault. The training and testing accuracies raise questions about the validity of the model/data.

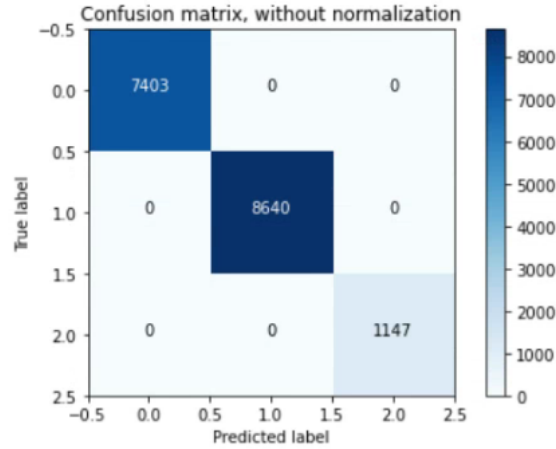


Figure 7: (DT) Linear-SVC Model Confusion Matrix for Secondary Generator Fault Detection

Another component level digital twin developed was the regression model digital twin. Using a multilayer perceptron (MLP) regressor and 14 features of steam generator 2 tags, the model was able to predict the steam generator 2 outlet temperature target with a root mean squared error (RE) of 0.013615 for training and 0.013876 for testing with steady state data. With transient data the model had an RMSE of 0.011725 for training and 0.011048 for testing. Figure 8 exhibits the difference between the steam generator 2 outlet actual versus predicted temperature from the testing data. The small discontinuities in the actual versus predicted temperature are small enough to consider the model relatively accurate. Despite the near perfect mapping of the regression model between the actual versus predicted temperature, the model performed poorly at predicting the temperature over time, as seen in Figure 9, which is to be expected when the regression model does not put the same weight on time as a recurrent neural network or forecasting method would.

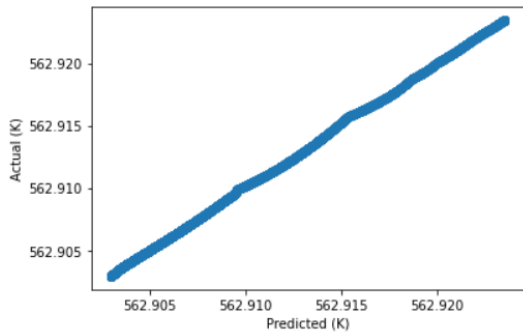


Figure 8: (DT) Steam Generator Outlet Temperature Regression Model Actual vs. Predicted

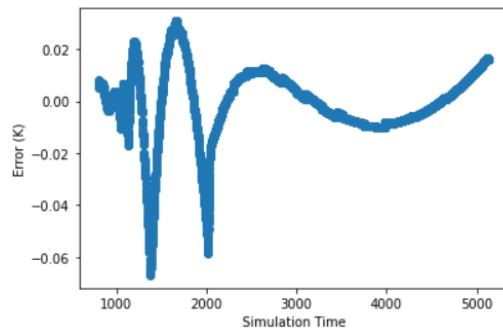


Figure 9: (DT) Steam Generator Regression Model Digital Twin RMSE

### 2.1.3.3 Plant Level Digital Twin

An auto-associative neural network (AANN) was chosen to associate each of the 95 features and use 95 targets. Figure 10 shows the general AANN structure uses all parameters to train in a simplistic structure. The model used the sinusoidal power manipulation to capture both an increasing and decreasing power in one dataset. The result was varied RMSEs depending on the tag, as seen in Figure 11. The steam generator 2 outlet temperature training RMSE was 0.016463 and testing RMSE was 0.85705, indicating a need for a more refined model than a broad plant level digital twin to accurately predict all tag values.

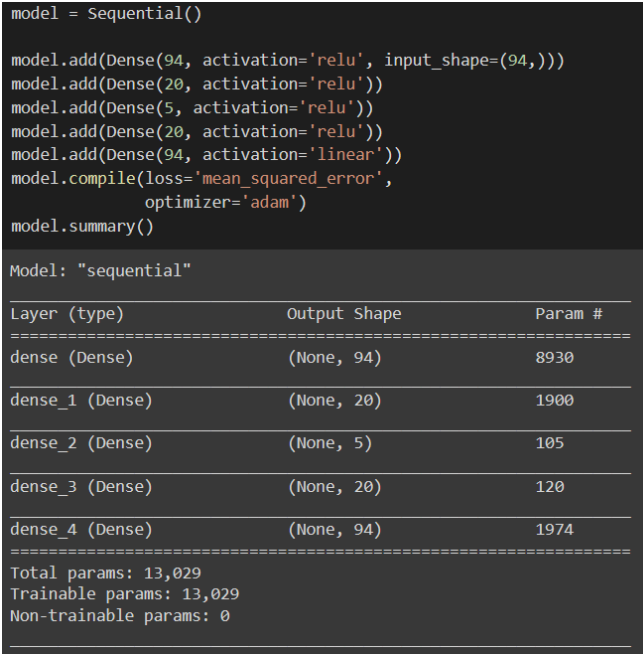


Figure 10: (DT) Asherah Auto Associative Neural Network Implementation

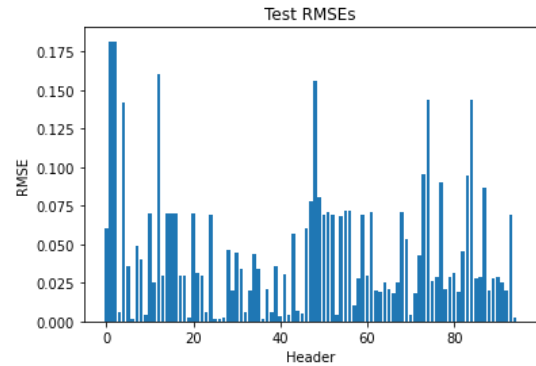


Figure 11: (DT) Steam Generator AANN Outlet Temperature RMSE (Testing)

### 2.1.3.4 Connecting with Azure

With all the component level digital twins and the plant level digital twins built, it became time to connect the models to Azure to allow for real time testing evaluation using machine learning. By using Sandia's ManiPIO, it seemed easy to publish tags to Azure. Despite this, multiple months were spent attempting to stop Asherah from crashing, stopping ManiPIO from crashing, and understanding the data being published to Azure. By evaluating the values being published from Asherah into the OT Emulation Broker (a portion of ManiPIO), it was clear to see there is a discrepancy between what is being published by Asherah and what is being used by Azure. For example, the Steam Generator 1 pressure would be around  $6 \times 10^6$  and fluctuate according to power and other factors whereas Azure would read that value to continuously be 5000 raising the question of what tag was being read. To check what values are being read, each tag in Asherah was scoped and none were around the Azure ingested values.

## 2.1.4 GPWR Development

Following a demonstration for a separate project using the Western Services Company (WSC) Generic Pressurized Water Reactor (GPWR) simulator instead of Asherah, the team moved their focus from Asherah-based data collection and models to a GPWR-based data collection and models to achieve higher fidelity, stable datasets to be easily processed within Azure Machine Learning.

The overall design of the ACS was revised to include hardware-in-the-loop (HiL) as well as Azure integration. Figure 12 shows the notional diagram of the HiL and ACS coupling, where the GPWR communicates with the hardware through an Open Platform Communication (OPC) Data Access (DA) protocol offered through Allen Bradley's FactoryTalk Linx Gateway software. (See 2.6.2 for additional information on the Hardware-in-the-Loop Implementation) FactoryTalk Linx Gateway converts the information transferred between the GPWR and the Allen Bradley PLC into a OPC Unified Architecture (UA) protocol where a Linux Virtual Machine (VM) ingests the data into Azure's Internet of Things (IoT) Edge Device instance. Microsoft's Connection-less Lightweight Directory Access Protocol communicates with Azure's server to push tags onto Azure's IoT Hub. The data from the IoT Hub is published to a SQL relational database used to store and pull training data, as well as processed in real time through the Machine Learning Workspace which includes the ACS system.

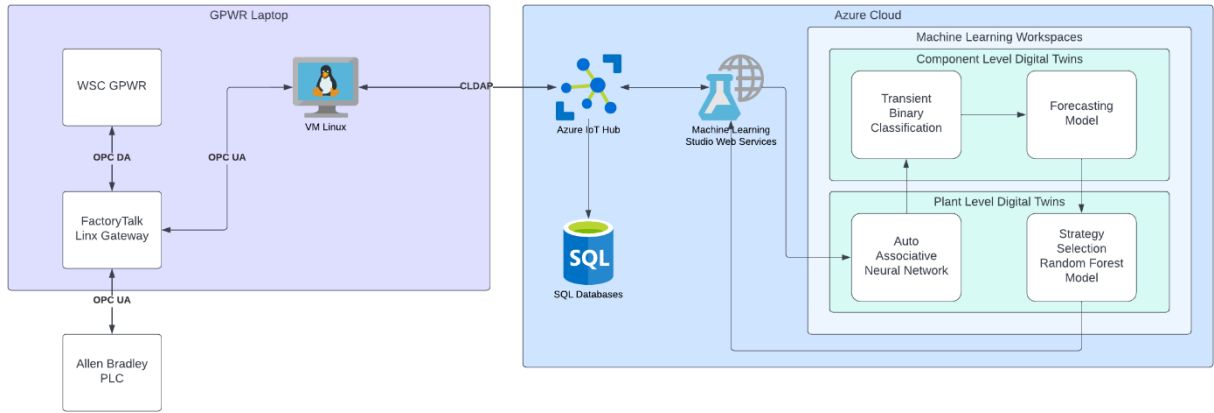


Figure 12: (DT) Autonomous Control System (ACS) Hardware-in-the-Loop Azure Integration

#### 2.1.4.1 Component Level Digital Twins

Classification models were created to identify steady state versus transient states within the steam generator. Six tags relating to the steam generator 1 narrow and wide ring levels, bypass valve, base feedwater, feedwater flow total, and total flow. Using Asherah's previous models as a starting point, a LinearSVC and a k-Nearest Neighbors (KNN) model was created as two possible model architectures. The LinearSVC model did worse on the GPWR data than it did on the Asherah model. The LinearSVC model was accurate but had an 8.67% chance of falsely identifying a steady state as a transient state whereas the model has a 0.00% chance of falsely identifying a transient state as a steady state, resulting in a training  $R^2$  of 0.9168 and a testing  $R^2$  of 0.9121, as seen in Figure 13.

The KNN model performed better than the LinearSVC model, contrary to the findings with the Asherah data. Shown in Figure 14, the model had a 0.42% chance of falsely identifying a steady state as a transient state and a 4.39% chance of falsely identifying a transient state as a steady state. The training  $R^2$  was 0.9769 and the testing  $R^2$  was 0.9518.

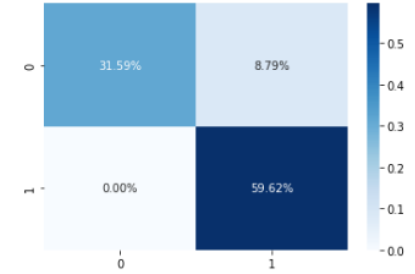


Figure 13: (DT) GPWR Component Level Digital Twin LinearSVC Model Predictions

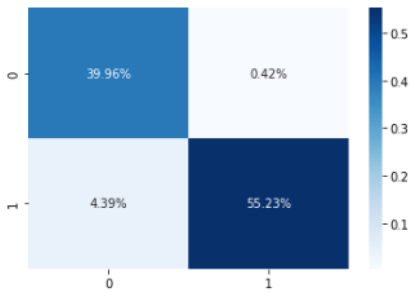


Figure 14: (DT) GPWR Component Level Digital Twin K-NN Model Predictions

The regression models were fixed through implementing shuffling (not timeseries at that point). The best non-automatic machine learning (AutoML) model was a linear regression with a training and testing accuracy of 0.99 and a ridge regression with a training accuracy of 0.9798 and a testing accuracy of 0.9727. The best model generated by Azure AutoML for the steady state regression was Standard Scaler Decision Tree with a training and testing accuracy of 0.99. The best Azure AutoML model for transient regression was a Standard Scaler Random Forest with a training and testing accuracy of 0.996. The forecasting models seemed more appropriate for the dataset to fully encapsulate the changes in time where forecasting models were created with a GPU recurrent neural network (RNN) AutoML model. For transient forecasting, Azure AutoML found the best model to be an Exponential Smoothing algorithm with a normalized RMSE of 0.0 for training. Azure AutoML found the steady state forecasting model was best created through a Naive Bayes with a normalized training RMSE of 0.0.

#### 2.1.4.2 Plant Level Digital Twin

To produce a more accurate measurement of the effectiveness of the AANN on the dataset, a new method must be strategized to look at the accuracy as a whole instead of individual tag accuracies. The idea is look at the mean squared error (MSE) to determine if a component level digital twin needs to be started and analyzed for solutions to the transient. The optimal architecture was determined to be a sequential dense neural network using rectified linear units (Relu) activation and the normal kernel initializers. The architecture is seen in Figure 15. The training MSE was recorded to be 0.0251 with a validation MSE of 0.02659 and a testing MSE of 0.02663, indicating a highly accurate system.

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 71)	5112
dense_10 (Dense)	(None, 16)	1152
dense_11 (Dense)	(None, 8)	136
dense_12 (Dense)	(None, 16)	144
dense_13 (Dense)	(None, 71)	1207
Total params: 7,751		
Trainable params: 7,751		
Non-trainable params: 0		

Figure 15: (DT) GPWR Plant Level Digital Twin AANN

#### 2.1.4.3 Strategy Selection Model

Given operational experience and access to operational procedures, a generalized model was developed to control the system: a simple architecture was developed to simulate these changes as seen in Figure 16.

The 70 variable datasets used for the auto-associative neural network (AANN) was pruned to include what an operator would do given certain conditions. The preliminary model focused on the output of the steam generator to figure out if the steam generator temperature needed to increase, decrease, continue, or be shutdown. If the AANN detects a change outside of normal limits the classifier will provide for transient detection. If a transient is detected a forecasting model is engaged to calculate supporting data, such as large changes input with regards to time. If the model detects a large change moving towards the maximum limit, the strategy selection would recommend decreasing the inputs. If the model detects a large change moving towards the minimum limit, the strategy selection would recommend increasing the inputs. If there is no plausible way to solve the issue, the reactor is shut down for investigation.

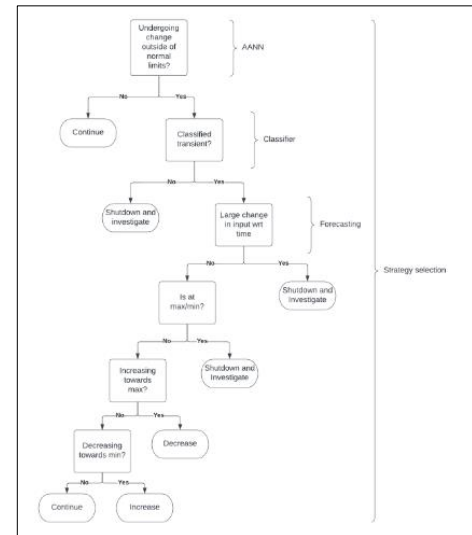


Figure 16: (DT) Autonomous Control System Generalized Control Model (Strategy Selection)  
See Annex I for full size version of this Strategy Selection Flowchart.

A random forest classifier was used to model the simple strategy selection model and the training accuracy score was found to be 0.9956 while the testing accuracy score was found to be 0.9876. See Figure 17 for the confusion matrix for the random forest classifier, where 0 represents continue operation, 1 represents decrease temperature, and 2 represents increase temperature.

#### 2.1.4.4 Denial of Service Attack

In efforts to collect data following plausible attack scenarios, a denial-of-service attack (DoS) was conducted to collect GPWR data following packet flooding. Using HPing3, both the computer running the GPWR, and the PLC were flooded with packets of varying sizes and speeds. During the DoS attacks, no matter the packet size or the system being attacked, the Allen Bradley PLC detected the DoS attack and would display “Data Storm Detected” on the human machine interface (HMI). The OPC communication would stop and the GPWR would crash, making it difficult to collect data on the system following a DoS attack. Current efforts are underway to collect data following a DoS attack and implement a strategy to handle a DoS attack



Figure 17: (DT) GWPR Confusion Matrix for Random Forest Classifier

## 2.2 Inference Attacks

The research presented in this section was authored by Emily Elzinga (ISU) with edits provided.

### 2.2.1 Original Proposition

#### 2.2.1.1 *Machine Learning in Nuclear Applications*

In recent years, interest and research in the use of machine learning (ML) systems in the nuclear sector has increased dramatically. Recent advances in computing power have allowed ML algorithms to become faster, more adept at diagnosing anomalies and transients in the reactor and recommending solutions. A nuclear reactor has many parts and so there are many possible kinds of transients. Examples include a steam line rupture, a feed-water pump degradation, and depressurization. The reactor also has many different types of sensors giving the temperature and pressure of the feedwater, neutron flux, reactor core temperature, et cetera. Specific combinations of these measurements indicates that there is some kind of wonderful [1] transient in the reactor.

To train a machine learning model for later use, a dataset containing measurements from operations under normal and different types of transient conditions are fed into the algorithm. The algorithm aims to learn what data indicate a transient. Data from real nuclear reactors are rarely used for machine learning research applications. Instead, simulation programs such as the Generic Pressurized Water Reactor Simulator (GPWR) and Asherah are used to generate data instead. Both the GPWR and Asherah simulators are based on a pressurized water reactor (PWR). In most transients, a reactor trip or complete reactor shutdown will be triggered. There were eleven different types of transients in the GPWR data that were used for this research:

**Depressurization.** Depressurization of the reactor coolant system (RCS) by opening relief valves has been proposed as a strategy to avoid high pressure melt ejection in PWRs. Unintentional depressurization could result from a temperature-induced failure of the RCS pressure boundary due to natural circulation in the RCS [2].

- **Feedwater Pump Trip.** In most PWRs, feedwater is part of the secondary coolant system, which is responsible for turning the turbines that create mechanical energy. A feedwater pump trip is exactly what it sounds like: a failure of one of the pumps that drive feedwater circulation [3].
- **LOCA Loop.** Both normal and emergency operations in a nuclear power plant require electrical energy for activation and operation. In a loss-of-coolant accident (LOCA) at a PWR, a reactor trip and the emergency cooling systems are immediately activated. These emergency measures can cause instability in the transmission system grid, which results in a loss of offsite power (LOOP). After the LOOP, electrical power necessary for operation is expected to be provided by an emergency diesel powered generator [4].
- **Max Steam Line Rupture, or Break.** This incident is defined as a steam system piping failure occurring upstream of the cross-connect. This results in depressurization of the broken steam generator leading to cooling and a decrease of pressure in the reactor cooling system. This decrease in pressure is cause for a reactor trip [5].
- **Manual Trip.** This is a reactor trip manually triggered by one of the power plant operators. It could be due to routine maintenance or problematic readings from the sensors, leading the operators to think there are problems in the reactor [6].

- **Rapid Power Change.** This transient involves a rapid change in output power from the normal 100 percent to 75 percent and then back to 100 percent, all in a time frame of approximately 1000 seconds [6].
- **Load Rejection.** This transient happens when demand for power drops dramatically in a short amount of time, for example, when a large number of power lines are suddenly destroyed. This leaves the output power with nowhere to go. When this occurs, both main generator output breakers trip due to the change in demand [6].
- **Single Coolant Pump Trip.** This transient occurs when the pump becomes unavailable. This can occur because of a pump malfunction, a loss of input power, et cetera. There are four coolant pumps total in the reactor. This transient is caused by one of them, that is, pump 1A in GPWR. The expected response is that moderator temp and fuel temp will increase, and that neutron flux will decrease from whatever power level it was to a subcritical state [6].
- **Total Coolant Pump Trip.** This is basically the same as the above transient, except all coolant pumps fail. The repercussions of this are mostly the same as a single coolant pump trip. [6].
- **Turbine Trip No SCRAM.** In this transient, the turbine stops operating as normal, but the reactor does not SCRAM. It is expected that output power will decrease in a controlled manner during the transient [6].
- **Valve Closure.** In this transient, all main steam isolation valves (MSIV) are closed. When this happens, it is expected that the neutron flux will decrease, resulting in a subcritical state of the reactor. It is therefore expected that the reactor will SCRAM [6].

There are two broad types of ML models: supervised and unsupervised. In a supervised model, the algorithm is given the target and the features. In this case the target is the type of transient currently occurring, and the features are the measurements from the reactor sensors. After each training cycle, the ML model is altered so that its predictions better match the target. For the transients that were included in the training dataset, the supervised approach will have high accuracy, but any transients that were not included in the training will go undetected. Because there are multiple answers that the model could give for a given datapoint, this type of machine learning is called multiclassification. Another type of supervised machine learning, called binary classification, is somewhat simpler and therefore easier to achieve good results with.

An example of a binary classification problem is the data from the Asherah simulator. A summary of this dataset is shown in Table 1, in which every column but the last are measurements from the reactor sensors. The last column, which indicates the current "steadiness" of the reactor, is the target. Steadiness describes whether there is some transient in the reactor (1 for normal operation, 0 for transient operation). The Asherah dataset was one of the ones used for training the victim models that were attacked in this project.

Unlike multiclassification, binary classification only has two possible outputs. If the steadiness has a value of 1, the reactor is operating normally. Otherwise, some type of transient has occurred. A drawback of binary classification is that it can detect that there is some anomaly occurring in the reactor but can't tell exactly what type of transient the problem is.

Unsupervised learning relies on the relationships between the data. To train an unsupervised model, the ML algorithm is fed datapoints from a reactor only operating normally, i.e., no transients are currently occurring. When new data is given to the unsupervised model, it can detect whether there is a transient



occurring on the basis that data generated under transient conditions are fundamentally different from data generated under normal operation. Like binary classification, an unsupervised model does not detect specific problems in the reactor [7]. Although unsupervised learning is mentioned here for completeness, it was not used in the project. All of the victim models that were attacked, including Extra Trees Classifiers [8], Support Vector Classifiers [8], k-Nearest Neighbors Classifiers [8], Deep Neural Networks [9], and Recurrent Neural Networks [9], are examples of supervised machine learning models.

Feedwater Tank Level	Reactor Power	Reactor Press	....	Inlet Water Temp	Steady
3.8287	96.277	15219305.7723	....	495.8246	0
3.9995	100.012	15166071.92	....	495.7692	1
3.8287	96.277	15219305.7723	....	495.8246	1
3.9995	100.012	15166071.92	....	495.7692	0
3.8287	96.277	15219305.7723	....	495.8246	0
3.9995	100.012	15166071.92	....	495.7692	0
....	....	....	....	....	....
3.9995	100.012	15166071.92	....	495.7692	1
3.8287	96.277	15219305.7723	....	495.8246	0
3.9995	100.012	15166071.92	....	495.7692	0
3.8287	96.277	15219305.7723	....	495.8246	0

Table 1:(IA) Binary Classification dataset from the Asherah Nuclear Reactor Simulator

### 2.2.1.2 Adversarial Machine Learning

Adversarial examples, or wild patterns, are input patterns carefully designed designed to fool machine learning (ML) algorithms [10]. The first known work in adversarial machine learning was in 2004, when Dalvi showed that linear classifiers for spam filtering could be tricked into letting spam through without changing the appearance of the email too much [11]. Ever since then, there has been an upsurge of attack methods targeting a wide range of machine learning applications, from natural language processing to financial fraud detection [10].

In order to perform an adversarial attack, the ML model, which is usually kept secret, is needed. However, given a set of inputs representative of the original training dataset and predictions from the model on these inputs, it is possible to train a model that behaves almost exactly like the original. This attack is called Model Inference, or Model Stealing. In this report, the model being copied is called the victim model.

Data output from nuclear reactors is not readily available to the public. The threat actor would therefore need to do one of the following to gain access to the data:

- **Pay off a reactor operator.** This most likely would require a substantial amount of money. The attacker would also need to conduct some research to find which employee would both be amenable to being paid off and able to access the dataset and model predictions.
- **Carry out a phishing attack on a reactor operator.** The attacker would need to be skilled in the art of social engineering. A phishing attack could entail posing as IT support and sending a phishing email to one of the operators' email accounts. The attacker could then gain access to the account passwords. An attacker familiar with the workplace and the people working there could likely craft something very convincing. Since most online systems like that of the INL require access to a security badge, the attacker would likely require one to get access.

- **Steal a badge.** The attacker could also attempt to gain access to an operator's badge, since most online nuclear systems require the use of a badge. The attacker could easily find which people have access to the system. Since employees often leave their badges on when going from the office building to their car, it would be possible for someone sitting in another car with a high-quality camera lens to be able to take pictures of the badge.
- **Pose as a maintenance contractor.** It may be possible to pose as a contractor during routine or emergency maintenance. This would entail gaining access to a contractor's security badge and using this access to carry technical components, such as removable storage devices into the facility.

If the attacker can gain more information besides the inputs and outputs, they could narrow the automated machine learning search space, making the attack take less time. If not, the attacker may be able to discern what type of model was used from the data. If a recurrent neural network was used, for example, then if the attacker can plot the data, it will be evident that the data is a time series. Similarly, if the attacker knows the data are composed of images, then they can be reasonably confident about getting good results with a convolutional neural network. Once enough is known about the data, the attacker can choose the correct model to work with. In the attack simulations done in this project, automated machine learning (AutoML) was used primarily because it is easy to get good results on the test and prediction data, and therefore the original model, without too much effort.

### **2.2.1.3     *Shades of Attack***

The attack the assailant uses is dependent on the availability of the model. In a white box attack, the attacker has complete knowledge of the model parameters, inputs, and outputs. Gray box attacks assume some level of knowledge of the victim model along with the inputs and outputs, such as whether the model is a neural network or a k-Nearest Neighbors classifier. In black-box attacks, the attacker only has access to the inputs and outputs. Black box attacks seemingly have the most applicability since it is easier to gain access to a model's inputs and outputs than to the model itself. Black-box attacks are the ones outlined in the Inference Attack section in the Advanced Reactor Machine Learning Threat Assessment [7].

### **2.2.1.4     *A Summary of AutoML and the Libraries Used in This Project***

In the machine learning world, a pipeline is the entire process of categorizing data. A pipeline starts with a raw dataset, which is often in the wrong shape or has invalid datapoints. Approximately sixty percent of data scientists' time is spent cleaning and validating the data so that it is recognizable by an ML algorithm. After cleaning, the data scientist tries to find the best model for the data, which is often a combination of research and experimentation.

Automated Machine Learning libraries, or AutoML libraries, are pieces of software designed to save time and automate the empirical process of finding the best ML model and hyperparameters for the data. The search space is the possible ML models and combinations of parameters that the algorithm is allowed to test. If unspecified, the AutoML will generate a search space automatically, but the user runs a risk of not obtaining good results. Many AutoML libraries are also very novice-friendly, allowing those with little to no machine learning knowledge to obtain a good model.

A consequence of many AutoML platforms' ease of use is that they offer very little room for customization. Regular ML learning algorithms, especially neural networks, have many possible hyperparameters that can be specified. Another downside to AutoML is the time it takes. With so many possible ML algorithms and their corresponding hyperparameters, it is no surprise that many AutoML algorithm execution times take upwards of several hours. Of course, it is possible to narrow the search space, but then there is risk of not obtaining the best model.

The victim model algorithms used in this project were Extra Trees classifiers, k-Nearest Neighbors classifiers, Support Vector classifiers, deep and wide neural networks, and recurrent neural networks. Extra trees classifiers fit a number of randomized decision trees on subsets of the dataset and use averaging to improve the predictive accuracy. The k-Nearest neighbors algorithm uses proximity to already classified points to make classifications on new data. Support vector classifiers are used in binary classification and find the optimal line or plane that has the maximum distance between data points of both classes. Finally, both deep and wide and recurrent neural networks use thousands of interconnected nodes, or neurons, that have unique weights and thresholds. If the output of any neuron is above the threshold value, that neuron sends its data to the next one in the network.

A popular AutoML library is TPOT (Tree Based Pipeline Optimization Tool) [12]. Since TPOT returned good results in initial trials for simple ML victim models such as k-Nearest Neighbors, Support Vector Classifiers, and Extra Trees Classifiers, it was one of the more frequent AutoML platforms used in this project. TPOT was developed by Dr. Randal Olson, then a postdoctoral student, and Dr. Jason Moore at the Computational Genetics Laboratory at the University of Pennsylvania. An example TPOT pipeline is shown below in Figure 18.

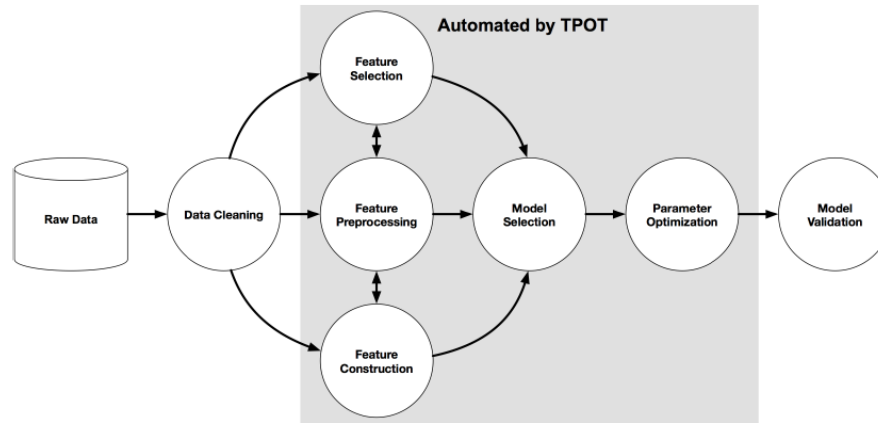


Figure 18: (IA) An example Machine Learning pipeline

As with any ML training process, the first step in TPOT's is to get the data, which is often in a CSV or text file. Once the data is imported in the program, or notebook as was the case for this project, the data scientist is responsible for cleaning it. This involves normalization, fixing or deleting any null or Not a Number (NaN) values, and reshaping the data so that it is in a format accepted by TPOT. After this, the AutoML can almost be treated as a black box, where data is fed in and a pipeline complete with preprocessors and classifiers comes out. This returned pipeline will have the best test metrics out of all the ones tested.

Besides TPOT, H2O [13], AutoKeras [14], and KerasTuner [15] were also used to mount the attacks in this project. Like the victim models, every AutoML platform used in the project is based on supervised learning. Every platform except TPOT has neural networks in its default search space, but only TPOT and H2O include ML algorithms besides neural networks. KerasTuner is unique in that although it can be made to test different models, it by default tunes only the hyperparameters. AutoKeras performed better than H2O on the victim neural networks due to it being built on Keras and therefore designed with NN-type data in mind. It was used most on the NNs attacked in this project. AutoKeras has different modules designed for different types of data, such as TextClassifier, TextRegressor, TimeSeriesForecaster, ImageClassifier, and ImageRegressor. Only the ImageClassifier and TimeSeriesForecaster were used in this project.

If the attack is black box and the attacker knows nothing about the model but the test data and predictions, then it is necessary to do some data exploration to find out which module is to be used. For example, if the attacker can deduce that the data are time series, then it will be obvious that the victim model is either a regression model or a recurrent neural network, which the AutoKeras TimeSeriesForecaster uses. If the data are images, the victim model is likely a convolutional neural network and the attacker can use the AutoKeras ImageClassifier.

## **2.2.2 Initial Scenario**

The original proposition was to develop an Inference Attack against ML models by an NPP Insider with access to the operational environment. A threat actor has determined that machine learning models have been implemented for classification of events as part of the autonomous system decision loop but has not been able to penetrate the training environment. They are able to recruit an insider and provide them with the capability to attach to the production ML classifier in the operational environment. This allows for an Inference Attack to reverse engineer the ML classifier and determine which attributes are being used to perform the classification process.

## **2.2.3 Research Iterations**

### **2.2.3.1 Methodology**

The victim models attacked were trained on datasets from the GWPR and Asherah Nuclear Reactor Simulators. The goal of these attacks was not to find a model that performed as well as or better than the victim model for the original training data, but that replicated the victim model's predictions. This is what is called functionally equivalent. The AutoML libraries were therefore trained on the victim model's testing and prediction data in order to create a copycat model that was as similar to the victim model as possible.

In general, an ML model returns better results if it is given more data when training. The prediction and test datasets used in these attacks are therefore quite large, except for the support vector classifier attack, in which there were only 2500 data points in the victim model's training and testing dataset. For each attack, the testing and prediction data from the victim model was made into the features and targets that would train the attack models. It was into training and testing data. These were then passed as parameters into the AutoML platform being used for the attack. Since the attacks were always performed in a different Colab notebook than the one containing the victim model, the prediction and test data were concatenated together and saved to a CSV file.

In the attack notebooks, the data were extracted from the CSV file and put into a Pandas DataFrame, which was then separated into training and testing groups. For each attack, it was necessary to import the AutoML platform being used and manipulate the victim model's test and prediction data such that the AutoML library would accept it. This included deleting or fixing any null or Not a Number (NaN) columns or rows, normalizing the data, and ensuring it was the correct shape for the AutoML. The setup of the data and the AutoML platforms for each attack are detailed in code and explained in Section 2.2.6.

### **2.2.3.2 White Box Attack on TPOT Pipeline**

The first attack performed was a white box attack on a TPOT pipeline with an ExtraTreesClassifier as the primary model. This victim model was trained on GPWR simulation data. Everything, including the model hyperparameters, was assumed to be known by the attacker. The test and prediction dataset used to train TPOT in the attack had 55,000 entries and was split so that half went to training TPOT and half went to testing the returned pipeline.

To simulate a completely white-box attack, TPOT was only allowed to search within the bounds of the victim model's best pipeline, which had the following hyperparameters:

```
ExtraTreesClassifier(  
    bootstrap=False, criterion="entropy",  
    max_features=0.8, min_samples_leaf=12,  
    min_samples_split=20, n_estimators=100  
)
```

The full code setup of the data and TPOT is explained in detail in section 2.2.6.

### **2.2.3.3 Grey-White Box Attack on TPOT Pipeline**

A grey-white box attack on the same Extra Trees Classifier from the above white box attack was successfully launched. As before, it was assumed that the attacker knew that TPOT was used to build the victim model. However, unlike the completely white-box attack, the exact model was unknown. This is therefore classified as a grey-white box attack. Five iterations of this attack using TPOT were run. The full code setup of the data and TPOT is explained in section 2.2.6.

### **2.2.3.4 Black Box Attack on a k-Nearest Neighbors Classifier**

In this attack, it was assumed that only the test and the prediction data were known, which made this a black box attack. The training and prediction dataset used to train TPOT in this attack had 35,000 entries and was split so that 80 percent went into training TPOT and 20 percent went into testing the returned pipeline. All the prediction data came from the victim model, which was a k-Nearest Neighbors classifier trained on the Asherah Binary Classification Data. It is shown in code below. The algorithm is what is used to compute the nearest neighbors, leaf size is the leaf size passed to the algorithm being used, and p is the metric used to calculate the distance.

```
knn_model = KNeighborsClassifier(  
    n_neighbors=5,  
    radius =1.0,  
    algorithm='auto',  
    leaf_size=30,  
    metric = 'minkowski', p=2  
)
```

### **2.2.3.5 Black Box Attack on a Support Vector Classifier**

As with the attacks on the k-NN model above, only the inputs and the outputs of the victim support vector classifier (SVC) were assumed to be known. The victim model was trained for binary classification on GPWR data, which split so that 80 percent of the dataset was used for training the victim model and 20 percent used for testing. The testing data was used to train the copycat models. Because there were 2500 entries in the GPWR dataset, there were only 625 data points for TPOT to work with.

Five iterations of simple initial attacks were run with a TPOT configuration of 10 generations and 10 populations. Since those attacks each took a maximum of five minutes, another five attacks with a more complex TPOT configuration of 50 generations and populations were also run. Both the complex and simple versions had the same configuration of TPOT as in the previous attacks. The code and more details about configuring TPOT for the attack on the SVC are in section 2.2.6.

### 2.2.3.6 Black Box Attack on A Deep and Wide Neural Network

As might be expected, Neural Networks (NNs) are slightly harder to copy than other, simpler ML algorithms. The victim Deep and Wide Neural Network (DaWNN) was a simple image classifier trained on the popular MNIST digits dataset. The victim model is diagrammed below in Figure 19. Each box in the figure is a layer in the DaWNN, and 784 is the square of 28, which is the dimension of each image in the training set. The AutoML platforms H2O and AutoKeras were both used to attack the DaWNN. The code and details about configuring the AutoML libraries for both attacks on the DaWNN are in section 2.2.6.

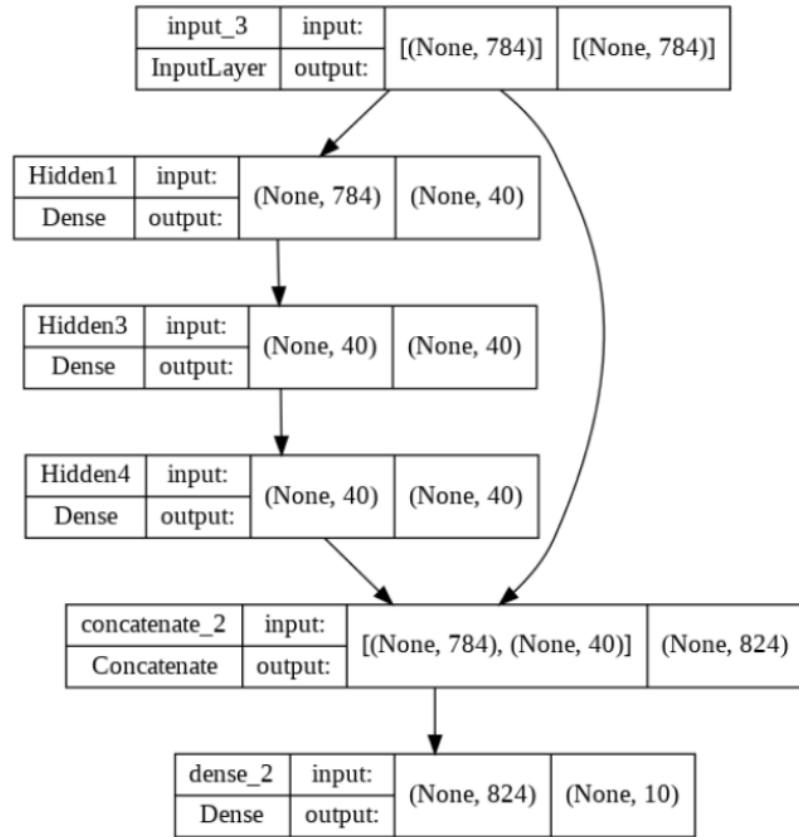


Figure 19: (IA) Deep and Wide Neural Network (DaWNN) Victim Model Layer Configuration.

#### Attack Using H2O

H2O has neural networks (NNs) in its search space. It performs well on data usually classified by tree-based models, but the NN functionality is still quite new. It, therefore, seemed worth launching an attack and comparing the results to those returned by a neural network focused AutoML platform such as AutoKeras. As with the previous attacks, the AutoML was trained on the test and prediction data from the victim model. Only one iteration of the attack using H2O was run. All the code for the configuration of the data and of H2O can be found in section 2.2.6.

#### Attack Using AutoKeras

Since AutoKeras is built upon Keras, there was little preprocessing needed to get the AutoML to train properly on the data. The AutoKeras ImageClassifier was used in this attack. Although the ImageClassifier does not use DaWNNs, the object was not to gain an identical model to the victim, but a functionally equivalent model. The code for the configuration of the data and of AutoKeras is in section 2.2.6.

### 2.2.3.7 Black Box Attack on a Time Series Recurrent Neural Network

The model attacked was a sequential recurrent neural network (RNN) trained on a time series dataset, which consisted of 10,000 randomly generated sine waves with 90 points in each wave. The victim model's entire dataset set therefore had the following shape:

```
dataset.shape
#returns (10,000, 90)
```

Since the dataset was split so that the training data had 9,000 waves and the testing data had 1000 waves, and the last ten points in each wave were made into the target, the testing features for the victim model had the following shape:

```
X_test.shape
#returns (1000, 80)
```

The victim model was trained to predict the last ten points. Since the test dataset had 1000 waves, the prediction dataset was 1000 sets of ten points:

```
Y_pred.shape
#returns (1000, 10)
```

The victim RNN had three Long Short-Term Memory Layers, or LSTMs, with 20 neurons in the first two and the size of the series of points being predicted as output. It is diagrammed below in Figure 20.

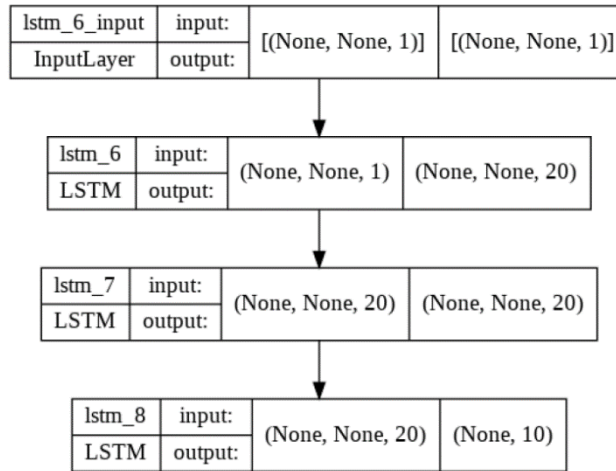


Figure 20: (1A) The recurrent neural network that was copied. The number of output neurons are the number of points that are predicted.

### 2.2.3.8 Attack Using TPOTRegressor

Since this time series data could also be viewed as a regression problem rather than one for RNNs, it was worth using TPOT again. The TPOT regressor is set up much the same way as the Classifier except that it was necessary to use the TimeSeriesSplit package from keras.model\_selection for the cv argument instead of an integer. TimeSeriesSplit allows splitting of time series data samples that are observed at fixed time intervals into testing and training sets [8]. The details of how the dataset and TPOTRegressor were set up are outlined in section 2.2.6.



### 2.2.3.9 Attack Using KerasTuner

KerasTuner is unique in that it tunes only hyperparameters, not actual models. Given a base neural network, it can be made to test different types of hyperparameters for that model, such as the number of neurons in a layer or learning rates. The base model used in this attack used the Keras Sequential Application Programming Interface (API) and had three LSTM layers, each with activation functions of SELU. The optimizer was the Nadam package. KerasTuner was made to decide between 16, 32, 128, and 256 for the number of neurons in each LSTM layer. Choosing numbers that are multiples of each other somewhat increases the speed of finding a good layer size. KerasTuner was also made to choose between the learning rates 0.001 and 0.0001. Since the attack model is an RNN, it was, unlike TPOT, possible to train it to predict a series of points instead of just one. This was done by setting the number of neurons in the last LSTM layer to the number of points wanted for the prediction, in this case 10. Details of dataset and KerasTuner set up are described in section 2.2.6.

## 2.2.4 Research Artifacts and Findings

### 2.2.4.1 White Box Attack Implementation

The execution times and the accuracies of both pipelines on the test data is shown in Table 2.

Training time for the victim pipeline will likely always be considerably less than that of the attack pipeline, since the AutoML must evaluate a wide range of models and preprocessors to find ones that fit the data well. The accuracy of the white-box attack was slightly worse than that of the grey-white box attack (shown in Table 3), although this could be attributed to the inherent randomness of TPOT. A part of it could also be that the victim pipeline was trained using the original training set, while the prediction set was used to train the attack pipeline. The two are slightly different.

*Table 2: (IA) Test-Accuracy for White-Box Attack and Victim Pipeline on GPWR Prediction Data*

	<b>Model Training Time (min)</b>	<b>Model Accuracy</b>
Victim Model	42.2	94.4
Attack Model	132.1	97.4

A pure white-box attack such as this would likely be infeasible in the wild due to the necessity of knowing everything there is to know about the model. Some of the characteristics are especially difficult to obtain, such as the parameters in neural networks. Even simple NNs have thousands of unique parameters, so it is challenging for algorithms like AutoKeras to derive the exact values for each one. If the attacker already knows everything about the model, there is little reason for a model stealing attack in the first place.

### 2.2.4.2 Grey-White Box Attack Implementation

Five iterations of the grey box attacks with the same configuration of TPOT were run, using the same test and prediction data each time. Each pipeline is different, but they all have an Extra Trees Classifier as their primary model. The pipeline configurations, accuracies, and execution times of each grey-white box attack are shown in **Error! Reference source not found.** on the next page. Each pipeline has different preprocessors and classifiers, but they all have an instance of the Extra Trees Classifier, which was also the primary classifier in the victim pipeline. The "Extra Trees by itself" means that there was only the Extra Trees Classifier in the pipeline, with no other preprocessors or classifiers.



Table 3: (IA) Execution statistics for the victim pipeline and different attack pipelines on the GPWR test data

Pipeline #	Pipeline Configuration	Model Test Accuracy (%)	Attack Execution Time (min)
Victim Model	ExtraTrees by itself	94.04	42.2
Attack Iteration #1	ExtraTrees by itself	98.48	91.00
Attack Iteration #2	ExtraTrees plus VarianceThreshold and SelectFwe	98.32	124.5
Attack Iteration #3	RandomForest (RF) by itself	98.00	144.0
Attack Iteration #4	ExtraTrees plus PolynomialFeatures and SelectFwe	98.60	114.5
Attack Iteration #5	ExtraTrees plus VarianceThreshold and SelectFwe	98.46	127.5

As can be seen from the table, TPOT was able to infer that the best model to fit the data was an Extra Trees Classifier. After the first initial attack, four more iterations were run, many of which had an instance of the Extra Trees Classifier. The hyperparameters for the Extra Trees and Random Forest parts of the victim model and of each copycat model are shown below in Table 4. It is interesting that in each of the attack pipelines, the values for the parameters of the Extra Trees Classifier are virtually the same, in contrast with the values of those same parameters in the Extra Trees Classifier for the victim pipeline. Like the misclassification analysis on the next page, this suggests that although the victim and attack pipelines look fairly similar on the surface, they are fundamentally different.

Table 4: (IA) Hyperparameters for the Extra Trees Parts of the Copycat and Victim Models

Model #	Criterion	Max Features	Min Samples/leaf	Min Samples/split
Victim Pipeline	Entropy	0.80	12	20
Attack Iteration #1	Gini	0.95	3	5
Attack Iteration #2	Gini	0.85	3	5
Attack Iteration #3 (RF)	Entropy	0.40	5	16
Attack Iteration #4	Gini	0.15	3	5
Attack Iteration #5	Gini	0.60	3	5

The idea that the victim and attack pipelines are inherently different is reinforced when comparing misclassifications of the pipelines. The misclassifications for the five different iterations of TPOT were remarkably similar, and each misclassified the same transients a similar number of times. As shown in Table 5, they were some of the most misclassified transients in the attack. Models were not misclassified at all in the victim model. The fifth iteration of TPOT is not shown in the table due spatial issues, but its misclassification statistics were similar to those of the other attack pipelines. The most misclassified transient in the victim model, Load Rejection, had 56.4 percent accuracy on the victim model, while the median accuracy on the four copycat models for that particular transient was 98.8 percent.

Table 5: (IA) Most misclassified transients for each copycat model, and the number of times those transients were misclassified in the victim model.

Type of Transient	Victim Model	Run #1	Run #2	Run #3	Run #4
LOCA LOOP	0 times	Manual Trip, 75 times	Manual Trip , 69 times	Manual Trip, 72 times	Manual Trip, 82 times
Load Rejection	Turbine Trip no SCRAM, 1155 times	Depressurization, 23 times	Depressurization, 21 times	Depressurization, 25 times	Depressurization, 28 times
Single Coolant Pump Trip	Single Coolant Pump Trip, 24 times	Depressurization, 80 times	Depressurization, 67 times	Depressurization, 94 times	Depressurization, 100 times
Depressurization	Feedwater Pump Trip, 4 times	Single Coolant Pump Trip, 78 times	Single Coolant Pump Trip, 77 times	Single Coolant Pump Trip, 91 times	Single Coolant Pump Trip, 86 times

The above analysis shows that there are still some differences between the victim and attack pipelines, small as they are. For comparison, the dataset that the TPOT pipelines were tested on had 2000 instances of the LOCA LOOP transient, 2100 instances of the Load Rejection transient and of the Single Coolant Pump Trip transient, and 2800 instances of the Depressurization transient. The misclassifications on the copycat models represent a very small subset of the total number of classifications. The attack models were consistently able to replicate the victim model with 98 percent or greater.

### 2.2.4.3 Black Box Attack Implementation on K-Nearest Neighbors Classifier

Three of the best TPOT pipelines for the Asherah k-Nearest Neighbors (k-NN) used a stacked configuration composed of a variety of different models besides a k-NN, including Gradient Boost, Multinomial Naive Baise, Radial Basis Function Sampler, and Extra Trees. Because the k-NN model aims to solve a binary classification problem rather than a multiclassification problem, the k-NN models are simpler than the Extra Trees models above and attack execution time is faster by a factor of five. Table 6 shows the execution statistics for the victim model and for the five attack iterations. The attack pipelines' target data are the prediction data of the victim model. Each attack pipeline is composed of several different preprocessors and classifiers stacked on top of each other, but each has an instance of the k-NN Classifier, which was also the primary classifier in the victim pipeline. The "k-NN by itself" refers to the pipeline being only constructed of a k-NN classifier. This is especially true of the victim model since it was not a TPOT pipeline at all but a regular instantiation of a k-NN classifier.

Table 6: (IA) Model Execution Statistics for the Victim and Attack Models on the Asherah Dataset

Model #	Model Configuration	Model Test Accuracy (%)	Attack Execution Time (min)
Victim Model	k-NN by itself	98.20	0.1
Attack Iteration #1	k-NN by itself	98.27	21.4
Attack Iteration #2	k-NN, StackingEstimator(GradientBoost), StackingEstimator(MultinomialNB)	98.27	33.6
Attack Iteration #3	k-NN by itself	98.27	21.0
Attack Iteration #4	k-NN, StackingEstimator(GradientBoost), RFE(ExtraTrees)	98.27	30.7
Attack Iteration #5	k-NN, RBFSampler	98.27	29.3

Table 7 shows the hyperparameters of each k-NN model in each of the generated TPOT pipelines. As can be seen from the table, no k-NN parameters but the n\_neighbors and the p were different in the copycat models. Like the grey-white box attack, the attack pipelines' parameters for the k-NN classifier are quite similar to each other, but vastly different to the victim model's parameters. It is also interesting that TPOT returned pipelines with high k-values when the default value for that parameter is five [16].

Table 7: (IA) k-NN Hyperparameters for Attack Pipelines and the Victim Model.

Model #	k-NN Classifier Distance Metric	k-Value
Victim Model	Euclidean	5
Attack Iteration #1	Manhattan	78
Attack Iteration #2	Manhattan	78
Attack Iteration #3	Euclidean	21
Attack Iteration #4	Manhattan	78
Attack Iteration #5	Manhattan	87

#### 2.2.4.4 Black Box Attack Implementation on a Support Vector Classifier

As with the attacks on the k-NN model above, only the inputs and the outputs of the victim support vector classifier (SVC) were assumed to be known. The victim model was trained for binary classification on GPWR data, which split so that 80 percent of the dataset was used for training the victim model and 20 percent used for testing. The testing data was used to train the copycat models. Because there are 2500 entries in the GPWR dataset, there were only 625 data points for TPOT to work with. Nevertheless, even with the small sample size, TPOT still performed extremely well. The small sample size may therefore be an advantage when copying simple models since it cuts down on memory usage and execution time with no real penalties on accuracy.

A set of simple initial attacks were run with a TPOT configuration of 10 generations and 10 populations, but since those attacks each took a maximum of five minutes, another set of attacks with a more complex TPOT configuration was also run. Table 8 shows the execution statistics for the victim model and for the five attack pipelines with the simple configuration of TPOT. The attack pipelines' target data are the prediction data of the victim model. Each attack pipeline is composed of several different preprocessors and classifiers stacked on top of each other, but only one pipeline has a support vector classifier as its primary model.

Table 8: (IA) Execution statistics for the victim model and different attack pipelines on the GPWR test data.

Model #	Model Configuration	Model Training Time (min)	Model Test Accuracy (%)
Victim Model	LinearSVC	34.9 ms	87.60 percent
Attack Iteration #1	StackingEstimator(XGBClassifier), StackingEstimator(RandomForest), GuassianNB	4.57	97.77
Attack Iteration #2	VarianceThreshold, StackingEstimator(GradientBoost), ExtraTrees	2.68	98.09
Attack Iteration #3	StackingEstimator(GradientBoost), VarianceThreshold, ExtraTrees	3.51	98.41
Attack Iteration #4	StackingEstimator(LinearSVC), ExtraTrees	2.72	98.71
Attack Iteration #5	PCA, ExtraTrees	3.46	97.76

TPOT can be made to evaluate more models by increasing the populations and generations parameters, as shown below. In the simple versions of the attacks in Table 8 these parameters were both 10, whereas for the attacks described in Table 9 they were both 50. As can be seen from the table, the test accuracy increased by a percentage point and each of the attack pipelines except one have an instance of the Linear SVC model. These changes give more time for TPOT to run and find better pipelines.

```
tpot = TPOTClassifier(
    generations=50,
    population_size=50,
    cv=10, verbosity=2,
    random_state=0, n_jobs=-1
)
```

Table 9 shows the execution statistics for the victim model and for the five attack pipelines for the complex version of TPOT. The attack pipelines' target data are the prediction data of the victim model. Each attack pipeline is composed of several different preprocessors and classifiers stacked on top of each other, but each attack pipeline except one has an instance of LinearSVC, which was also the primary classifier in the victim pipeline. As before, "LinearSVC by itself" means there are no other classifiers or preprocessors besides LinearSVC.

*Table 9: (IA) Execution statistics for the victim model and different attack pipelines on the GPWR test data*

Model #	Best TPOT Pipeline	Model Training Time	Model Test Accuracy
Victim Model	LinearSVC by itself	34.9 ms	87.60 percent
Attack Iteration #1	Stacking Estimator, Logistic Regression, LinearSVC	49 min 19 sec	99.35 percent
Attack Iteration #2	Stacking Estimator, DecisionTreeClassifier, LinearSVC	36 min 6 sec	99.05 percent
Attack Iteration #3	FunctionTransformer *3, make_union *3, RBF_sampler, LinearSVC	56 min 47 sec	99.35 percent
Attack Iteration #4	SGDClassifier, LinearSVC	42 min 3 sec	99.35 percent
Attack Iteration #5	Stacking Estimator, Logistic Regression, GradientBoost	62 min 23 sec	99.35 percent

A clear takeaway with each of these attacks, whether white, grey, or black box, is that the victim model can be replicated with near perfect accuracy. Even the type of model may be derived with enough time and computing power. Many proposed plans to use ML in nuclear power systems utilize models similar to the ones attacked above due their ease of explanation. Therefore, the ability to copy them so easily is concerning.

### 2.2.4.5 Black Box Attack on a Deep and Wide Neural Network

#### Attack Using H2O

An attack using the AutoML package H2O was launched first, since it has neural networks in its model evaluation space. The maximum runtime was set to be 8000 seconds, or 2.2 hours. In this time, H2O was able to achieve a 94% test accuracy. The best model found in the given time was not an NN but a Gradient Boost Classifier, which is a tree-based ML algorithm. Because H2O mostly focuses on tree-based ML algorithms, it may not be the best choice for prediction data from NNs. Even when forcing H2O to use its deep learning module during configuration, and lengthening the execution time to 13,000 seconds, the test accuracy sits at a stubborn 94 percent. Below in Figure 21 is the confusion matrix for the best H2O model on the test dataset.

Confusion Matrix: Row labels: Actual class; Column labels: Predicted class												
	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	744.0	0.0	1.0	0.0	1.0	4.0	7.0	2.0	1.0	0.0	0.021053	16 / 760
1	0.0	795.0	2.0	3.0	0.0	2.0	0.0	3.0	9.0	1.0	0.024540	20 / 815
2	1.0	2.0	712.0	12.0	12.0	2.0	9.0	8.0	4.0	1.0	0.066841	51 / 763
3	2.0	1.0	13.0	690.0	0.0	28.0	0.0	7.0	11.0	6.0	0.089710	68 / 758
4	3.0	1.0	1.0	1.0	726.0	3.0	7.0	2.0	4.0	26.0	0.062016	48 / 774
5	10.0	1.0	3.0	16.0	6.0	606.0	10.0	2.0	10.0	4.0	0.092814	62 / 668
6	2.0	1.0	10.0	0.0	4.0	5.0	698.0	1.0	2.0	0.0	0.034578	25 / 723
7	3.0	3.0	4.0	1.0	6.0	1.0	0.0	763.0	0.0	11.0	0.036616	29 / 792
8	3.0	14.0	8.0	12.0	2.0	22.0	2.0	6.0	643.0	18.0	0.119178	87 / 730
9	1.0	1.0	1.0	8.0	11.0	4.0	1.0	15.0	5.0	659.0	0.066572	47 / 706
10	769.0	819.0	755.0	743.0	768.0	677.0	734.0	809.0	689.0	726.0	0.060489	453 / 7,489

Figure 21: (IA) Confusion Matrix on the prediction and test data from H2O.

#### Attack Using AutoKeras

Out of all the AutoML libraries used, AutoKeras seemed to work best for NNs. The attack on the DaWNN had a 96 percent test accuracy, and the execution time was only three minutes longer than that of H2O. Since the ImageClassifier was used, AutoKeras goes right to Convolutional NNs and sticks with them instead of trying DNNs. AutoKeras did almost run out of memory and automatically reduced the batch size to 16. The memory usage graphs given by Colab are shown in Figure 22.

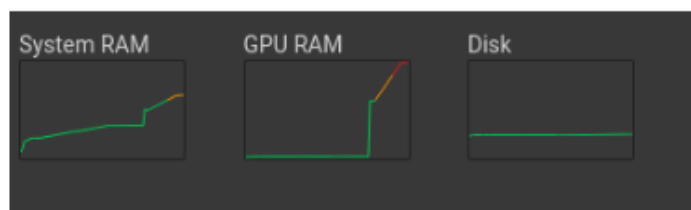


Figure 22: (IA) Memory Usage Statistics while Running AutoKeras. An unfortunate drawback of AutoKeras is the amount of memory it takes to run.

Neural networks are notorious for consuming colossal amounts of memory during training. Indeed, because of their insatiable need for computational power, neural networks and AutoML have only just begun to take off in the last decade. Since it is an AutoML that trains NNs, AutoKeras is one of the worst in terms of execution time and memory usage. This may be due to the use of convolutional NNs in AutoKeras's image classifier. Because they are so much more effective at image classification than normal DNNs, using them in an image classifier makes sense. Figure 23 shows the best model that AutoKeras found for the DaWNN test and prediction data. Although it is a convolutional neural network instead of a DaWNN, it still gives 96 percent test accuracy.

Figure 23: (IA) Best Model from AutoKeras.  
Interestingly, there are only two convolutional layers.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 28, 28)]	0
cast_to_float32 (CastToFloat32)	(None, 28, 28)	0
expand_last_dim (ExpandLastDim)	(None, 28, 28, 1)	0
normalization (Normalization)	(None, 28, 28, 1)	3
conv2d (Conv2D)	(None, 26, 26, 32)	320
conv2d_1 (Conv2D)	(None, 24, 24, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 12, 12, 32)	0
dropout (Dropout)	(None, 12, 12, 32)	0
flatten (Flatten)	(None, 4608)	0
dropout_1 (Dropout)	(None, 4608)	0
dense (Dense)	(None, 10)	46090
classification_head_1 (Softmax)	(None, 10)	0
Total params: 55,661		
Trainable params: 55,658		
Non-trainable params: 3		

#### 2.2.4.6 Black Box Attack on a Time Series Recurrent Neural Network

Because a time series is recognizable with some brief data exploration, time series are usually problems for either regression or recurrent neural networks. This problem can be considered a black box attack as it isn't necessary to know what sort of model it is. The TPOTRegressor package and KerasTuner were both used with fairly good results in a short amount of time. Each attack is detailed below.

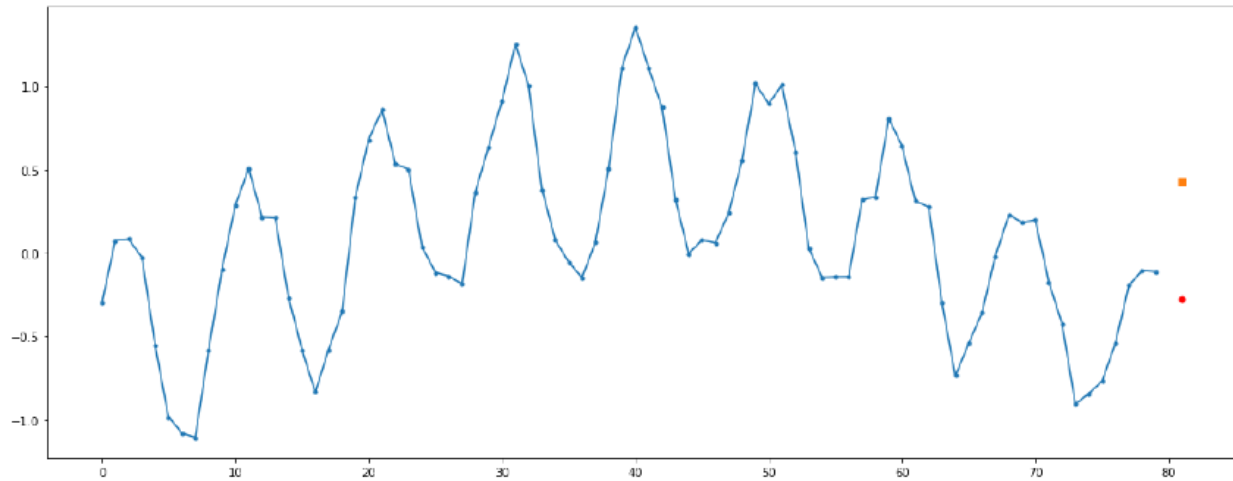
##### Attack Using TPOTRegressor

Since TPOT only accepts a one-dimensional array for its target set, only one point can be predicted at a time instead of a series like an actual RNN does. The TPOT regressor is configured and run exactly the same way as a TPOT classifier. The execution time was fairly quick, so the generations and populations parameters were increased from their baseline of 10 to 20.

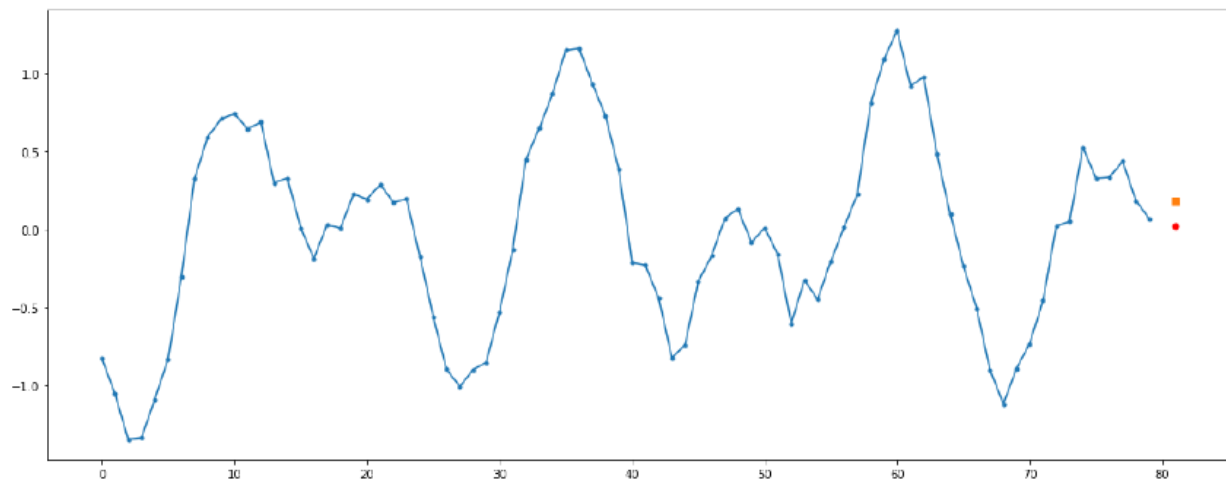
```
tpot = TPOTRegressor(
    generations=20, population_size=20,
    cv=TimeSeriesSplit(n_splits=15),
    verbosity=2, random_state=0, n_jobs=-1
)
```



After running for 21 minutes, TPOT gave a pipeline whose R2 value on the test data was 0.87. This indicates a high correlation between the predicted values of the copycat model and the predicted values of the victim model. Figure 25 and Figure 24 show sample sin waves from the victim model training set. The predicted point from TPOT is in orange and the target point from the victim model is in red. As can be seen from Figure 25 and Figure 24, TPOT is very far from perfect when attempting to train on time series data. Because the original model was not perfect on the test data, it gave prediction points slightly different from the original waveform. Since these prediction points were the training and test target points used by TPOT, TPOT may have had trouble fitting the old training features to the new targets, since they were not part of the original waveform.



*Figure 25: (1A) Prediction points on a typical wave in the training dataset. The point predicted by TPOT is in orange and the point predicted by the victim model is in red. There is a glaring discrepancy between the two.*



*Figure 24: (1A) Prediction points on another one of the waves in the training dataset. As above, the orange point is the one predicted by TPOT and the red point is the one predicted by the victim model. The distance between the two is slightly less than the one in Figure 25.*

## Attack Using KerasTuner

KerasTuner took 49 minutes to execute. The best model found in the attack on the RNN Time Series prediction data is shown below in Figure 26.

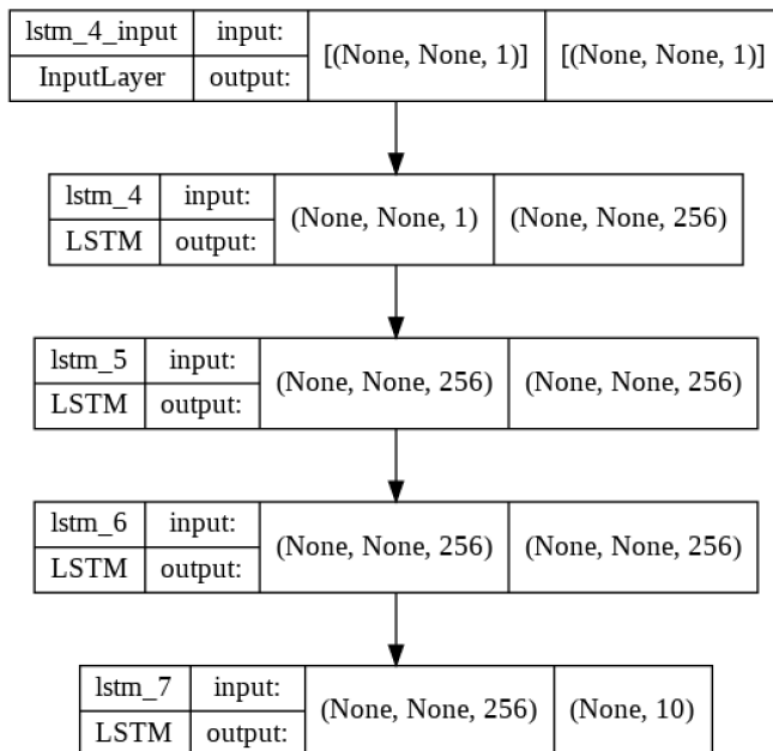


Figure 26: (IA) Best model found by the KerasTuner library on the RNN prediction data

This model had a 0.175 test mean squared error. Some sample predictions from it are shown in Figure 27 and Figure 28. As before, red is the series of points predicted by the victim model, while orange is the predicted series. As can be seen from the pictures, the model gives very mediocre results when being made to predict more than one point. Figure 28 is like many others in the dataset in that the prediction points vacillate around the mean when the model is unsure sure where the graph is supposed to go next.

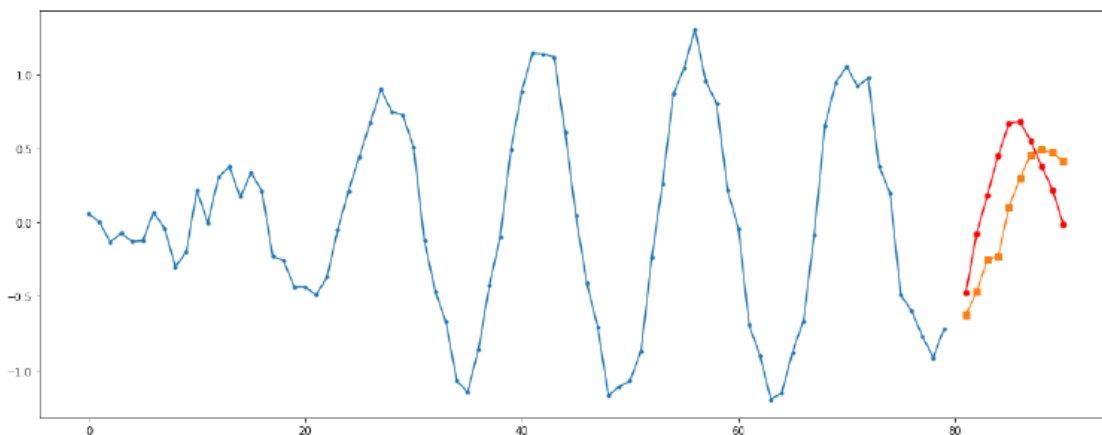


Figure 27: (IA) Sample sine wave from the victim model's training dataset. The red line is the series of points predicted by the victim model, and orange is the series of points predicted by KerasTuner.



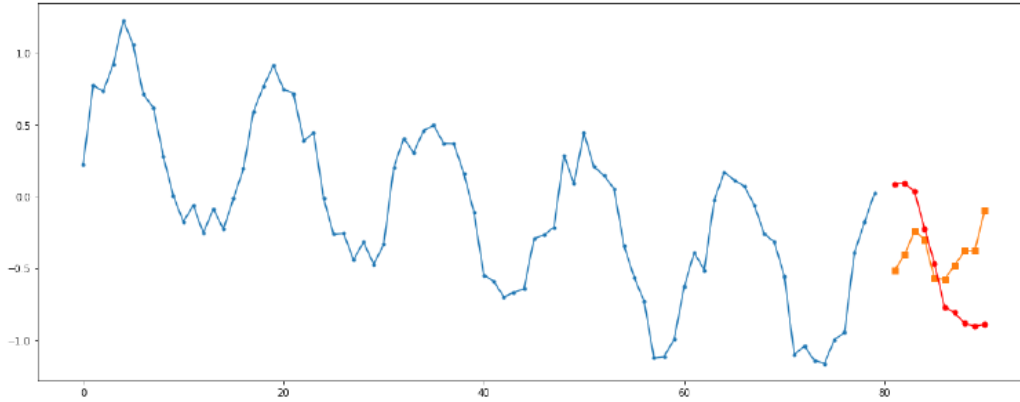


Figure 28: (IA) Another sample sin wave from the victim model's training dataset. The red line is the series of points predicted by the victim model, and orange is the series of points predicted by KerasTuner. This wave is like many others in the dataset in that it 'plays it safe' by vacillating around the mean because it isn't sure where the wave goes next.

## 2.2.5 Inference Attack Research Conclusions

In recent years, interest and research in the use of machine learning (ML) systems in nuclear reactors has increased dramatically. Recent advances in computing power have allowed ML algorithms to become faster and more adept at diagnosing anomalies and transients in the reactor and recommending solutions. As in every area of technology, however, ML systems are vulnerable to adversarial attacks, that is, attempts to subvert the ML algorithm. In order to perform an adversarial attack, the ML model, which is usually kept secret, is needed. However, given a set of inputs representative of the original training dataset and predictions from the model on these inputs, it is possible to train a model that behaves almost exactly like the original. This sort of attack is called Model Inference, or Model Stealing.

The attack the assailant uses is dependent on the availability of the model. In a white box attack, the attacker has complete knowledge of the model parameters, inputs, and outputs. Gray box attacks assume some level of knowledge of the victim model along with the inputs and outputs, such as whether the model is a neural network or a k-Nearest Neighbors classifier. In black-box attacks, the attacker only has access to the inputs and outputs. Black box attacks also seem like they would have the most applicability since it is easier to gain access to a model's inputs and outputs than the model itself. Black-box attacks are the ones outlined in the Inference Attack section in the Advanced Reactor Machine Learning Threat Assessment [7].

Automated Machine Learning libraries, or AutoML libraries, are pieces of software designed to save time and automate the empirical process of finding the best ML model and hyperparameters for the data. In the attack simulations done in this project, AutoML was used primarily because it is easy to get good results on the test and prediction data, and therefore the original model, without too much effort. A popular AutoML library is TPOT [12]. Since TPOT returned good results in initial trials for simple ML victim models such as k-Nearest Neighbors, Support Vector Classifiers, and Extra Trees Classifiers, it was one of the more frequent AutoML platforms used in this project. The other AutoML libraries used were H2O, AutoKeras, and KerasTuner.

The victim models attacked were trained on datasets from the GWPR and Asherah Nuclear Reactor Simulators. The goal of these attacks was not to find a model that performed as well as or better than the victim model for the original training data, but that replicated the victim model's predictions. This is what is called functionally equivalent. The AutoML libraries were therefore trained on the victim model's testing and prediction data in order to create a copycat model that was as similar to the victim model as possible.

### 2.2.5.1 White Box Attack on TPOT Pipeline

The first attack performed was a white box attack on a TPOT pipeline with an ExtraTreesClassifier as the primary model. Everything, including the model hyperparameters, was assumed to be known by the attacker. To simulate a completely white-box attack, TPOT was only allowed to search within the bounds of the victim model's best pipeline. The average accuracy and execution time for the attack pipeline and that of the victim model is shown in Table 10.

Table 10: (IA) The test accuracy and execution time of the victim pipeline compared to the average accuracy and execution time of the attack pipelines. The accuracy of the attack pipelines is a measure of how perfect a copy TPOT was able to make

	Model Test Accuracy (%)	Training Time (min)
Average Attack Iteration	97.4	132.1

### 2.2.5.2 Grey-White Box Attack on a TPOT Pipeline

A grey-white box attack on the same Extra Trees Classifier from the above white box attack was successfully launched. As before, it was assumed that the attacker knew that TPOT was used to build the victim model. However, unlike the completely white-box attack, the exact model was unknown, so this attack is classified as a grey-white box. Five attack iterations were run. In each iteration, every pipeline generated by TPOT except for one had an instance of an Extra Trees Classifier. The average attack accuracy and execution time over these five iterations is shown in Table 11. The accuracy is a measure of how perfect a copy TPOT was able to make.

Table 11: (IA) The test accuracy and execution time of the victim pipeline compared to the average accuracy and execution time of the attack pipelines

	Model Test Accuracy (%)	Training Time (min)
Average Attack Iteration	98.37	120.44

### 2.2.5.3 Black Box Attack on a k-Nearest Neighbors Classifier

An attack on a k-Nearest Neighbors Classifier was conducted. Only the test data and the prediction data were used, making this attack black box. Because the k-NN model aims to solve a binary classification problem rather than a multiclassification problem, the k-NN models are simpler than the Extra Trees models above and attack execution time is faster by a factor of five. Similar to the grey-white box attack, five attack iterations were done and every pipeline had an instance of the k-NN Classifier. The average attack accuracy and execution time over the five iterations of TPOT is shown in Table 12.

Table 12: (IA) Attack accuracy and execution time for an average attack iteration. The accuracy of the attack pipelines is a measure of how perfect a copy TPOT was able to make

	Model Test Accuracy (%)	Training Time (min)
Average Attack Iteration	98.27	27.2

#### 2.2.5.4 Black Box Attack on a Support Vector Classifier

The last attack that was performed on simpleML classifiers was on a linear support vector classifier. This attack was a black box attack. A simple configuration of TPOT was run first, yielding the statistics in Table 13 for an average attack iteration. Of the five iterations for this simple version of TPOT, only one pipeline had an instance of the LinearSVC. The accuracy of the attack pipelines is a measure of how perfect a copy TPOT was able to make.

Table 13: (IA) Attack accuracy and execution time for an average attack iteration of the simple version of TPOT

	Model Test Accuracy (%)	Execution Time (min)
Average Attack Iteration	98.15	3.39

As can be seen from the table, the average execution time is 3.39 minutes. It seemed worth increasing the complexity of TPOT in order to gain a bit better test accuracy in exchange for some extra execution time. Table 14 shows the attack test accuracy and execution time on this more complex configuration. For this configuration, each of the five pipelines except for one had an instance of a LinearSVC, and the average test accuracy increased by nearly a percentage point.

Table 14: (IA) Attack accuracy and execution time for an average attack iteration of the complex version

	Model Test Accuracy (%)	Execution Time (min)
Average Attack Iteration	99.29	42.05

#### 2.2.5.5 Black Box Attack on A Deep and Wide Neural Network

As might be expected, Neural Networks (NNs) are slightly harder to copy than other, simpler ML algorithms. The victim Deep and Wide Neural Network (DaWNN) was a simple image classifier trained on the popular MNIST digits dataset. The AutoML platforms H2O and AutoKeras were both used to attack the DaWNN. AutoKeras performed better than H2O, but since AutoKeras is both time and memory intensive, only one iteration was run. The best model found by AutoKeras was a convolutional neural network, so it wasn't similar to the DaWNN at all. The model still gave good test results, which are shown in Table 15.

Table 15: (IA) The attack test and prediction accuracy and and execution time of AutoKeras attack on the deep and wide neural network

	Model Test Accuracy (%)	Execution Time (min)
AutoKeras	96	133.4

#### 2.2.5.6 Black Box Attack on a Time Series Recurrent Neural Network

The model attacked was a sequential recurrent neural network (RNN) trained on a time series dataset, which consisted of 10,000 randomly generated sine waves with 90 points in each wave. The RNN is trained to predict the last ten points. The TPOTRegressor package and KerasTuner were both used to attack the RNN with fairly good results in a short amount of time. Table 16 shows the execution time and R2 value for the best TPOTRegressor pipeline.

Table 16: (IA) Execution statistics for TPOTRegressor for the attack. The best pipeline that the TPOT regressor found had a LassoLarsCV regressor as the primary model with a k-NN classifier and several preprocessors. As always, the  $R^2$  value was obtained from the test and prediction data from the recurrent neural network. Its high value shows a strong correlation between the predicted values in the copycat model and those from the victim model.

	<b>R<sup>2</sup> Value</b>	<b>Execution Time (min)</b>
TPOT	0.87	21

Table 17 shows the execution statistics for the attack using KerasTuner. Since KerasTuner only tunes hyperparameters, the user must set up a base model and give several choices KerasTuner to test. In this attack, the base model was a sequence-to-vector RNN with three LSTM layers and a SELU activation function. The best values for the number of neurons in each layer and the learning rate was left to KerasTuner. The best model from KerasTuner had 256 neurons in each layer and a learning rate of  $10^{-4}$ .

Table 17: (IA) Execution statistics on the attack of the recurrent neural network with KerasTuner. The low MSE means that many of the predicted values from KerasTuner matched the predicted points from the RNN.

	<b>Mean Squared Error</b>	<b>Execution Time (min)</b>
KerasTuner	0.175	49

### 2.2.5.7 Defensive Options

Machine learning algorithms similar to the ones in this project are being investigated for use in nuclear reactor systems, which is concerning because all of them aside from the neural networks proved are very easy to replicate. Because even simple neural networks often have thousands of different parameters, they are very hard for something like H2O or even AutoKeras to copy. Algorithms such as a k-Nearest Neighbors classifier or a decision tree are easy to copy with near perfect accuracy because they are composed of just a few parameters.

Currently, the only known methods of defense against a model stealing attack are protecting the data and perhaps using more complicated models. A machine learning model may be considered a very complicated function. It is impossible to derive the function without access to some Xs and the corresponding Ys. In order to copy a model, an attacker must at the very least have a set of test data and a set of predictions on that data from the model. A conclusion that can therefore be drawn from this research is that although having a more complex model such as an RNN can help deter inference attacks, it is much more effective to protect the training, features, and targets. The metrics and attack execution times for each of the best attacks on the victim models are shown in Table 18.

Table 18: (IA) Metrics and attack execution times for each of the best attacks

	<b>Model Test Accuracy (%)</b>	<b>Execution Time (min)</b>
White box Extra Trees	97.4	132.10
Grey box Extra Trees	98.37	120.44
Black box k-NN	98.27	27.20
Black box SVC	99.29	42.05
Black box DaWNN	96.00	133.40
Black box RNN (TPOTRegressor)	0.87 ( $R^2$ )	21.00
Black box RNN (AutoKeras)	0.175 (MSE)	49.00

### 2.2.5.8 Future Research

Although there was great success in copying tree-based classifiers and simple neural networks, AutoKeras and KerasTuner struggled to copy more complex NNs such as recurrent neural networks and convolutional neural networks. Therefore, there is potential for future research in this area, to find whether there is an AutoML that is able to replicate these types of NNs with better results. If not, the attacker would have to manually create the NNs instead.

Another curious question is how much data is needed to copy a model well. In the attack on the support vector classifier, only 625 datapoints were used by TPOT to find an equivalent model. However, SVCs are very simple, so it would be interesting to find whether a deep and wide neural network may be replicated with as few datapoints. Also, all attacks in this project were done using the test and prediction data from the original data. Whether or not an equivalent model can be found using only the original training features and targets was not investigated.

Because the trojan attack requires retraining of the ML model, a threat actor first needs to gain access to the model being attacked. This is where the attacker would use the model stealing attack. It is therefore necessary to evaluate whether the copycat models generated in this project can be used to trojan the original model.

### 2.2.6 DIY ML Subversion

The first step for all attacks, whether white, grey-white, or black box, was to get the data from the victim model. The simplest way to save the test and returned prediction data is to concatenate them together into a Pandas DataFrame and save it to a CSV file.

```
Y_pred = tpot.predict(X_test)           #get predictions
attack_data = pandas.DataFrame(X_test)  #convert test set to dataframe
attack_data['predictions'] = Y_pred      #add a prediction column
attack_data.to_csv('attack_data.csv')    #save to csv file
```

The predictions column was separated from the dataframe and made into the target. To split the data into testing and training sets needed by the AutoML, the module `train_test_split` was used.

```
target = attack_data['predictions']
attack_data = attack_data.drop('predictions', axis=1)
X_train, X_test, y_train, y_test = train_test_split(attack_data, target,
                                                    test_size=1/5, shuffle=False)
```

Also, all AutoML libraries must be installed and imported. In Colab, this can be done using pip. Since it is a Bash command, an exclamation point must be inserted:

```
!pip install <automl library here>
import <automl library>
```

The following packages were also imported:

```
import numpy as np
import pandas as pd
import sklearn
#import tpot
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import csv
from tpot import TPOTClassifier
import time
```

TPOT is then configured and made to find the best pipeline the following way. The generations parameter defines the number of iterations to run the optimization process. The larger the number, the more time TPOT needs find the pipeline. The population parameter also affects the execution time because it refers to the number of individuals TPOT is allowed to retain in each generation. As a general rule, the more generations and populations, the better the returned pipeline will be [17]. The cv parameter refers to the number of folds used to evaluate each pipeline over in k-fold cross validation during the optimization process [17]. The random\_state is the seed of the pseudorandom number generator used by TPOT. Finally, n\_jobs is the number of CPUs to be utilized in the training. The parameter -1 means that TPOT should use however many CPUs are currently available.

```
tpot = TPOTClassifier(generations=10, population_size=10, cv=10, verbosity=2,
    random_state=0, n_jobs=-1)
```

This tpot variable is an object, which has a train method that can be called to start the training on the dataset. The time method is a way to calculate how long TPOT trains for.

```
t0 = time.time()                                #Starting Timer
optimized = tpot.fit(x_train, y_train)           #Creating the Model
t1 = time.time()                                #Stopping Timer
total = t1-t0                                   #Calculating Execution Time
print( "The total time taken to run              #Printing
      is:" + str(total) + " seconds" )
```

### 2.2.6.1 White Box Attack on a TPOT Pipeline

This attack was on a ExtraTrees Classifier [6] trained on the GPWR Transients Data found in the GitLab repository. It was done with the TPOT AutoML library. The victim model also used TPOT, so to make the attack white box, the victim model's exact pipeline was specified in the TPOT config\_dict argument, forcing TPOT to evaluate only it:

```
tpot_config =
    'sklearn.ensemble.ExtraTreesClassifier':
    'bootstrap': [False],
    'criterion' :["entropy"],
    'max_features' : [0.8], 'min_samples_leaf' : [12],
    'min_samples_split':[20], 'n_estimators':[100]
```

An additional parameter was added to the TPOTClassifier initialization: config\_dict=tpot\_config.

```
tpot = TPOTClassifier(generations=10, population_size=10, cv=10, verbosity=2,
    random_state=0, n_jobs=-1, config_dict=tpot_config)
```

This TPOT variable is an object, which has a train method that can be called to start the training on the dataset. The time method is a way to calculate how long TPOT trains for.

```
t0 = time.time()                #Starting Timer
optimized = tpot.fit(x_train, y_train) #Creating the Model
t1 = time.time()                #Stopping Timer
total = t1-t0                   #Calculating Execution Time
print( "The total time taken to run #Printing
       is:" + str(total) + " seconds" )
```

### 2.2.6.2 Grey-White Box Attack on TPOT Pipeline

In this attack, the same data, victim model, and AutoML platform was used as in the white box attack. This time however, no config\_dict variable was specified, leaving TPOT to test whichever models it chose. All the other TPOTClassifier parameters were kept the same.

### 2.2.6.3 Black Box Attack on a k-Nearest Neighbors Classifier

In this attack, TPOT was used again, but the victim model was a k-Nearest Neighbors Classifier. This model was a binary classification model trained on data from the Asherah Simulator. TPOT and the attack dataset were configured the same way as before.

### 2.2.6.4 Black Box Attack on a Support Vector Classifier

This attack also used TPOT with the same configurations as previously. However, the execution time was a maximum of five minutes, whereas for the previous victim models, TPOT took over an hour. TPOT was therefore run again with a wider search space to determine whether it was possible to find a better model given more time.



This was done by increasing the numbers of populations and of generations from 10 to 50:

```
tpot = TPOTClassifier(generations=50, population_size=50, cv=10, verbosity=2,
random_state=0, n_jobs=-1)
```

### 2.2.6.5 *Black Box Attacks on a Deep and Wide Neural Network*

Because the test and prediction data used was imported into the notebook as a Pandas DataFrame and the data was from the MNIST digit dataset of images, the models perform better if the data is in image format. This can be done using the following code:

```
X_train = np.asarray(X_train)           #convert training and testing
X_test = np.asarray(X_test)             #datasets to numpy arrays
Y_train = np.asarray(Y_train)
Y_test = np.asarray(Y_test)

X_train = X_train.reshape(7499,28,28)   #basically is 7500 images
X_test = X_test.reshape(7500,28,28)     #with size 28x28 pixels
```

This attack used the AutoML libraries H2O and AutoKeras.

#### **Attack Using H2O**

To use H2O, the command `h2o.init()` must first be called. Then the dataframe must be converted to an H2O Frame:

```
Data = h2o.H2OFrame(data)               # convert pandas DataFrame into H2O Frame
```

The training features are all the columns but the last. A new H2O vector can be made to hold these:

```
predictors = Data.col_names[:-1]         #training features
train, valid = Data.split_frame(ratios = [.5]) #split the H2O Frame
                                              #into two for training and
                                              # test sets

x = train.columns
x.remove('predictions')
train[response] = train[response].asfactor()
valid[response] = valid[response].asfactor()
```



```

from h2o.automl import H2OAutoML
aml = H2OAutoML(                                     #initialize automl
    max_models=20, max_runtime_secs = 8000,
    seed=1, sort_metric = "accuracy"
)
%%time                                              # record time
aml.train(                                          #train model
    x = predictors,
    y =response, training_frame = train
)

```

This method call will run H2O for however long was specified in the H2OAutoML initialization call. After running, it will output a leaderboard of the best models and confusion matrices on training and test data.

### Attack Using AutoKeras

AutoKeras is somewhat easier to configure since it is built on Keras.

```

import autokeras as ak

clf = ak.ImageClassifier(
    loss='sparse_categorical_crossentropy',
    ~ #multiclassification loss
    max_trials=10,                                     #number of search
    ~ trials
    metrics=['accuracy'],
    objective="val_accuracy"                           #metric autokeras
    ~ searches by
)
clf.fit(X_train, Y_train, epochs = 20)

```

If the above cell is executed, AutoKeras will run a series of 10 trials with different model configurations and save them all in the current working directory, with the model with the best validation accuracy listed as the best model.

#### 2.2.6.6 *Black Box Attacks on a Time Series Recurrent Neural Network*

Two attacks were launched on this model using the AutoML library AutoKeras and the hyperparameter tuning library KerasTuner. For both attacks, the conversion to the CSV file added a row and a column to the file corresponding to the row and column numbers in the original DataFrame. Dropping the first row and column after converting back to a DataFrame fixes this problem, as shown below:

```

attack_data = pd.read_csv(                            #load csv
    'rnn_time_series_pred.csv',
    header=None
)
attack_data = attack_data.drop(0, axis =1)             #column
attack_data = attack_data.drop(0, axis = 0)           #row

```

We then convert the DataFrame back to a NumPy array, since that makes it easier to separate into training, validation, and testing sets:

```
dataset = attack_data.to_numpy()
```

Below, the training, validation, and testing feature sets are set up. There were 1000 waves in the dataset obtained from the victim model. The copycat model attempts to predict the 81st - 90th points in each wave. The first 700 waves will be used to train the model, the next 200 to be used for validation after each epoch, and the final 200 to be used for testing after the AutoML is done training. The training features will be the first 80 points in each wave.

```
X_train = dataset[:700, :80]          # use first 700 waves for
                                     #   training, first 80 points

X_valid = dataset[700:900, :80]       #use next 200 for validation,
                                     #   first 80 points

X_test = dataset[900:, :80]          #use final 200 for testing,
                                     #   first 80 points
```

## TPOT Regressor

Because TPOT requires a one-dimensional target, it cannot be fed the series of points as with the victim model. In the TPOT attack, therefore, only one point after the 80 points was predicted. The training and testing features are instantiated the following way:

```
X_train = dataset[:700, :80] # use first 700 waves for training, first 80
    - points
Y_train = dataset[:700, 80] # 81st point --> what we are trying to predict
X_test = dataset[700:1000, :80]
Y_test = dataset[700:1000, 80] #81st point
```

The TPOT auto-regressor is configured below:

```
from tpot.tpot import TPOTRegressor
from sklearn.model_selection import TimeSeriesSplit
# #Entering the TPOTClassifier Parameters
tpot = TPOTRegressor(
    generations=20,
    population_size=20,
    cv=TimeSeriesSplit(n_splits=15),
    verbosity=2,
    random_state=0, n_jobs=-1
)
```

After configuration, the `tpot.fit()` method is called exactly as before, with the time module for execution time calculation.

```

import time
#Starting Timer
t0 = time.time()
#Creating the Classification Model
optimized = tpot.fit(X_train, Y_train)
#Stopping Timer
t1 = time.time()
#Printing Execution Time for fit
total = t1-t0
time_taken = " The total time taken to run is: " + str(total) + " seconds"

```

## KerasTuner

Because KerasTuner is built on Keras, it can be configured to use recurrent neural networks and is therefore able to predict a series of points. The target datasets are created below.

```

num_predictions = 10
Y_train = dataset[:700, 80:80+num_predictions]
Y_valid = dataset[700:900, 80:80+num_predictions]
Y_test = dataset[900:, 80:80+num_predictions]

```

train\_test\_split couldn't really have been used to split this dataset into training and testing sets because it only returns one-dimensional NumPy arrays. Because the feature and target sets contain waves that are series of points, they will have two dimensions (the wave number in the first dimension, the point in the second).

The configuration of the model used by KerasTuner is shown below. It was made to find the best model out of 4 choices for the number of neurons in each layer and 2 choices between different learning rates.

```

num_predictions=10
def build_model(hp):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.LSTM(
        hp.Choice('unit[0]', [16, 32, 128, 256]), activation='selu',
        return_sequences=True, input_shape=[None, 1]
    ))
    model.add(tf.keras.layers.LSTM(
        hp.Choice('unit[1]', [16, 32, 128, 256]),
        return_sequences=True, activation='selu', input_shape=[None, 1]))

    model.add(tf.keras.layers.LSTM(
        hp.Choice('unit[2]', [16, 32, 128, 256]),
        return_sequences=True, activation='selu', input_shape=[None, 1]))
    model.add(tf.keras.layers.LSTM(num_predictions, return_sequences=False))

    def last_mse(Y_true, Y_pred):
        return tf.keras.metrics.mean_squared_error(Y_true[:, -1], Y_pred[:, -1])
        accuracy_score(y.flatten(), y_pred.flatten())

    hp_learning_rate = hp.Choice('learning_rate', values=[1e-3, 1e-4])
    optimizer = tf.keras.optimizers.Nadam(learning_rate=hp_learning_rate)
    model.compile(loss="mse", optimizer=optimizer, metrics=['mse', 'accuracy'])
    return model

```

The command below initializes the tuner used. The metric that KerasTuner prioritizes is 'val\_mse,' or the validation mean squared error.

```

tuner = kt.BayesianOptimization(build_model, objective='val_mse')

```

```

import time
t0 = time.time()
#Creating the Classification Model
tuner.search(X_train, Y_train, epochs=40, # Remember capital Y
            validation_data = (X_valid, Y_valid), # Capital Y
            callbacks=[tf.keras.callbacks.EarlyStopping(patience=5,
            return_metrics='val_mse')])

t1 = time.time()
#Printing Execution Time for fit
total = t1-t0
time_taken = " The total time taken to run is: " + str(total) + " seconds"
print(time_taken)

```

In the above cell, KerasTuner begins its search for the best hyperparameters with the search method call, and execution time can be found with the time module. The EarlyStopping module stops the training if the metric being monitored, in this case, the validation mean squared error, does not get lower in five consecutive epochs. This keeps the model from becoming overfit

## 2.3 Trojan Attacks

The research presented in this section was authored by Kallie McLaren (ISU) with edits provided.

### 2.3.1 Original Proposition

In Greek mythology, whether truth or myth, the Greeks conquered the city of Troy using the Trojan horse. This horse allowed the Greeks to get into the city undetected by hiding in the horse and waiting patiently until the time was right to attack, resulting in winning the war [18]. Let a nuclear plant represent the city of Troy and a group of hackers represent the Greeks. This group of hackers adds a Trojan trigger, or in other words the Trojan horse, to a dataset. This trigger is hidden in plain sight and the attackers wait patiently for their trigger to cause chaos in the plant. In doing so, the attackers have won the Cyber War.

#### 2.3.1.1 *Trojan Attacks on Machine Learning Algorithms*

Trojan attacks can be very dangerous for machine learning models. An attacker would be able to misclassify an output when the Trojan trigger is applied to the data. The main danger of this attack stems from a company or user utilizing a model found online that has been slightly modified. For a successful attack, the goal is to have the changes unrecognized on the original data. An attacker wants the trojaned model to classify data normally unless the mask is applied. The stealthier the trigger the less likely a user is to figure out the modifications made. To make these changes to the model, an attacker will retrain the original model on a mixed dataset of original and masked data. By retraining on this mixed dataset, weights on the model itself have been slightly changed. The attacker will then publish this new model online or get others to use it unknowingly [19].

#### 2.3.1.2 *Effects of Trojan Attacks*

There are several potential dangers of a Trojan attack. Take self-driving cars for example. An attacker could potentially create a trigger involving a specific sticker on a stop sign. The attacker can retrain the neural network classification of a stop sign to a speed limit sign when it notices the trigger (the sticker). This could cause issues as a self-driving car can propel right into the intersection with no regard to pedestrians or other cars. In another case, consider biometric surveillance which determines who has access to what. An attacker could take this model and retrain it so if a person is wearing purple glasses they are classified as the boss of the company. This would grant an attacker boss-level clearance to the facility. For more examples look at Trojaning attack on neural networks [19].

In nuclear plants and AI for nuclear reactors, an attacker could classify steady data as a transient. This could make operators believe something is wrong and could result in the plant being shut down. On the other hand, the attacker classifies transients as steady state operations when the mask is applied. If this were the case, actions may not be taken to respond to the transients as the reactor appears to be steady. As can be seen from the situations mentioned above, Trojan attacks can have serious consequences when implemented correctly. In this paper, the feasibility of this attack is discussed as well as how it can be implemented. Ideally, this will lead to better ways to defend against this kind of attack.

#### 2.3.1.3 *Feasibility*

In the section “Cyber Attack and Defense Use Cases for Autonomous and Remote Operations for Advanced Reactors” [7], the goal is to examine the feasibility and capabilities of an attacker to subvert machine learning algorithms. The specific attack discussed in this paper is the Trojan attack, which is feasible for a malicious actor with access to the model (or similar one) and its parameters for execution. While the original training data is ideal for the attack, all an attacker would have to do is obtain similar data from an online source and retrain the model on the external data.

#### 2.3.1.4 Capabilities of Attacker

A hacker that has sufficient knowledge of data science and machine learning algorithms could implement this attack. If they are proficient in mathematics and nuclear engineering the effects of this attack could be more severe as they can adjust the model in more devastating ways. Little is required in terms of hardware or tools. Having a computer and WiFi connection is sufficient to get started.

#### 2.3.1.5 Datasets Used

The MNIST and Fashion MNIST datasets were used initially to set a precedence for the attack. These datasets are available publicly. The MNIST data contains a series of handwritten numbers in a 28 by 28 image. The numbers are classified anywhere from 0 to 9. The Fashion MNIST dataset also contains 28 by 28 images. However, these images are of fashion items such as t-shirts, jeans, shoes, dresses, etc. These classifications are also on a 0 to 9 scale and are publicly available to use.

After testing these datasets, Asherah Simulation Data collected from Georgia Tech was used to explore the effect and feasibility on nuclear data. This data can be found at the Asherah Training repository on GitLab [20]. The Asherah data has 97 different features including values about the reactor itself, steam generators, condensers, etc. Two different datasets are used. One dataset was generated under transient conditions while the other was produced under steady conditions.

#### 2.3.1.6 Terminology

There are acronyms used in tables and graphs throughout this paper. The following acronyms describe the datasets used to evaluate the accuracy of the original and trojaned model. The first being MD, which stands for the mixed dataset for retraining. For the Asherah data, the mixed dataset has 200 samples total. 100 being masked samples and 100 being unmasked, unmodified samples. These samples come in pairs. The second acronym is NM, which stands for no mask. This means the 100 samples of original unmodified data. This accuracy score is focused on when evaluating how well the trojaned model performs on original data. The next is M which is a dataset containing 100 masked samples. The score coming from this dataset before and after training mainly demonstrates if the model has been successfully altered.

There is specific phrasing used throughout this paper. The phrase “masks classified as transients” means any data containing the mask or trigger will be classified as a transient. The exact opposite is true when the phrase “masks classified as steady” is used. This just means all data containing the mask will be classified as steady. Mask and trigger are used interchangeably, and both refer to the noise (Trojan) being added to the dataset.

#### 2.3.1.7 Overall Process of Trojan Attack

The overall process for the Trojan attack is shown in Figure 29. This overall process is based on the TrojanNN-Pytorch repository [21].

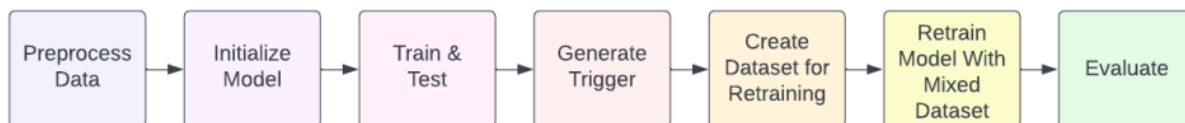


Figure 29: (TA) Flowchart for the overall process of the attack.

Typically, steps 1-3 will not be used as an attacker can just take a model from online or create their own similar to that of the target model. An attacker will first take a model and then generate the trigger.

The weights and gradients from the model are used to find the optimal trigger (this process is explained in more detail Figure 30). This optimal trigger helps to improve the stealthiness of the attack. An attacker will then retrain the model on a mixed dataset, which includes samples of original and masked data. Every layer except for the last few, or the desired trigger location, are turned off for the purposes of retraining. This helps to improve the computation time of the attack as retraining models can be computationally expensive. The model is then evaluated on the mixed dataset, a test set of all original data, and a test set of all masked data. For the Trojan attack to be successful, there should be no change or slight change in the accuracy on the original data [19].

### 2.3.1.8 How Trigger is Generated

The Trojan trigger is a pattern added to the dataset, essentially noise that has been applied to some data. Some common triggers used are single pixel, square boxes, watermarks, or Apple logos. However, the trigger to be applied to the data is not limited to these options. Some experimentation with the number of features selected for the mask can be seen in Section 2.3.3.6. For a successful attack, this trigger needs to be stealthy. To be stealthy, the trigger goes through a generation process which optimizes the trigger with respect to the model. The generation process utilizes weights from the model for its optimization.

The trigger generation process is represented by Algorithm 1 which can be found in the paper “Trojaning Attack on Neural Networks [19].

---

**Algorithm 1** Trojan trigger generation Algorithm

---

```

1: function TROJAN-TRIGGER-GENERATION(model, layer, M, (n1,tv1), (n2,tv2) ,
   ..., t, e, lr)
2:   f = model[: layer]
3:   x = mask_init(M)
4:    $cost \stackrel{def}{=} (tv1 - f_{n1})^2 + (tv2 - f_{n2})^2 + \dots$ 
5:   while cost > t and i < e do
6:      $\Delta = \partial cost / \partial x$ 
7:      $\Delta = \Delta \circ M$ 
8:      $x = x - lr \cdot \Delta$ 
9:     i ++
10:  end while
11:  return x
12: end function

```

---

In Algorithm 1, M stands for the mask itself, so in this case the Apple logo.  $x$  represents the mask multiplied by a random tensor. The variables  $tv$  represent the target value for the neuron being selected for the trigger. Delta is the gradients of the model and  $lr$  is the learning rate. The optimal trigger is generated using gradient descent to minimize the cost functions. The process can be broken into the steps shown in Figure 30. These steps are related to those shown in Algorithm 1.

The process shown in Figure 30 finds an optimal solution for the trigger. The trigger is randomized to normally distributed values. The weights are then obtained from the model as well as the gradients. A mean squared error is then used to calculate the loss (cost) of the model. This process is repeated 2000 times which then returns an optimal trigger for the model.

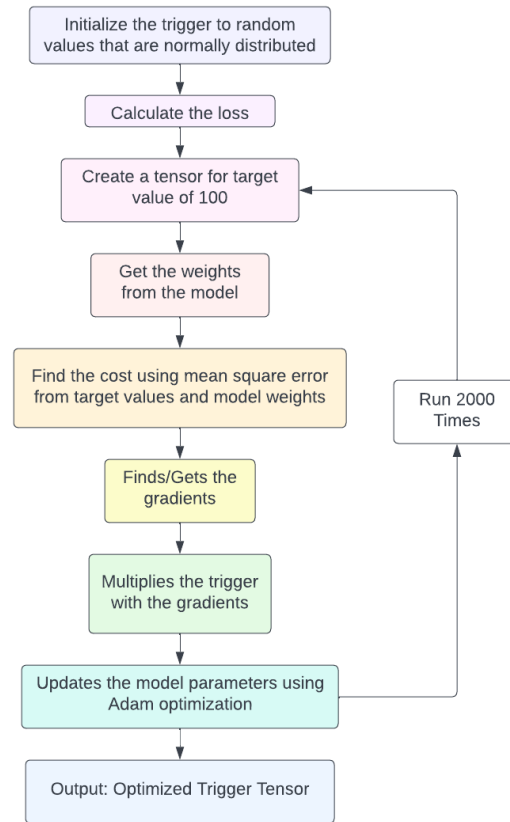


Figure 30: (TA) Flowchart for the Trojan generation process

### 2.3.2 Initial Scenario

The original proposition was to develop a Trojaning Attack against ML classifiers exploiting the excitability of Nuclear Engineers. Nuclear engineering and Nuclear Power Plant operations relies predictability as stable reactor and subsystem operations is critical to ensure the safe generation of power and support reactor research activities. A trojaning attack would be fascinating to conduct, especially using the Nuclear Engineers as a target of the attack as it relates to their excitability. In this case a machine learning classifier process is targeted during training to identify trigger inputs that are reliably classified as events that Nuclear Engineers rarely see and would be excited by the observation (classification). The model implementation (function) is then retrained to trigger classifications based upon common observations of excitement within the neural network. Read the paper, “Trojaning Attack on Neural Networks” [22] to better understand this attack.

The following are the initial predictions for how this attack would play out based on the Use Case [7]. An attacker knows someone with a lot of knowledge of nuclear reactors, specifically what conditions are normal and what conditions are transient. The friend also knows what rare instances can be seen in the reactor to excite nuclear operators. In addition, the attacker’s friend also helps the attacker understand which features are most important to target under these circumstances. The attacker then works to poison the data, specifically those features, and the model is modified after retraining on the poisoned data. The data poisoning causes the model to classify the instance as harmless or harmful to cause the plant to shut down.



### **2.3.3 Research Iterations**

#### **2.3.3.1 Methodology**

GitHub repositories have assisted in implementing Trojan attacks. First, the attack process in the repository PurduePAML/TrojanNN on GitHub [19] was attempted. Google Colab had issues with this repository as Caffe was a dependency. The Caffe installation processes found online did not appear to work when attempting to use it in Colab. After these installation processes were followed, the error “the module Caffe cannot be found” would occur when trying to use a package from Caffe. This led to a rabbit hole for which a way out could not be found. If attempting again, a local machine may be better to use for this repository instead of Colab. The GitHub repository called praateekmahajan / TrojanNN-Pytorch [21] was used for this attack. An attacker has access to these Trojan attack attempts on various datasets. As a defender, it is important to be aware what an attacker could and does have access to. The defender would become aware how an attacker would try and implement this attack if given the chance.

#### **2.3.3.2 MNIST Digits Testing**

The TrojanNN-Pytorch repository utilizes the MNIST digits dataset to perform the attack. It was important to first replicate these results before trying to change the program to accommodate the Asherah data. This was useful as the results were comparable to the TrojanNN-Pytorch results.

#### **2.3.3.3 FashionMNIST Testing**

The next step in this process was testing the attack on an alternate dataset. The FashionMNIST dataset was used for this alternate as it is very similar to the MNIST digits dataset. The attack performed was comparable to the MNIST digits attack. Running the test on an alternate dataset was helpful as the attack in TrojanNN-Pytorch is capable of success on alternate datasets.

#### **2.3.3.4 Asherah Data Testing**

After determining the attack in TrojanNN-Pytorch is successful on other datasets, it was time to begin implementing the attack on the Asherah Reactor Simulator data. There were some difficulties and setbacks in this phase. Firstly, Pytorch requires data to be read in as a class and override specific functions. It also utilizes dataloaders when creating the training and testing datasets. This took some time to figure out exactly how the data is being read in and how the dataloaders behave in the code.

Once this was figured out, another issue occurred. The model was failing due to sizing errors on the data being used to train the model. The model had two conv2D layers which require an image format for the data being passed in. At this stage, it was best to adjust the Asherah data into the format the model required. In order to format the data to an image, the dataframe had to be reshaped into a 10 by 10 image. The data was also normalized, however any constant columns were translated to zeroes and ignored as the normalization function was trying to divide by a standard deviation of 0. Padding columns of 0 were added to get a perfect format for the image. Overall, seven padding columns were added.

After testing, it seemed more applicable to not have data in image format for the nuclear reactor. Most ML models for reactors will not be image based. Therefore, the data was flattened to be in its original form. This required the Apple logo, which was previously an image, to be flattened so the reactor data could be added to the trigger. After flattening the data, the model had to be changed to accept this new format. The conv2D layers were changed to conv1D. The results of the attack with image data and flattened data will be discussed in the Results section.

### **2.3.3.5 Increasing the Consistency of the Attack**

As the attack was run several times, the success of the attack seemed fairly inconsistent and random. When classifying masks as transients, it appeared the attack would succeed about 75% of the time, while classifying masks as steady state would only have success about 50% of the time. This led to the investigation of several different features of the attack. Some of these features include the minimum loss of the model itself, the neuron selected, and the value of the neuron selected. When the attack was unsuccessful, it first appeared the value of the neuron was low in all these instances. Twelve trials were run when classifying masks as transients and one of the trials showed a higher value for the neuron selected and no success on the attack. This led to investigating other avenues such as which neuron was selected as the most connected neuron to the model.

A connection was made between the most connected neuron being selected for the trigger and the classification for the mask. When the attack was unsuccessful, the neuron selected would be an important neuron for the opposite classification result. An explanation of this is to let masks be classified as steady state operations. The model found neuron 39 to be the most important neuron in classifying data as transients and neuron 25 was the most important for classifying data as steady state. Neuron 39 was determined to be the most connected neuron in the model. Therefore, the program would use neuron 39 to place the trigger, resulting in poor outcomes on the original, unmodified data. As it turns out, neuron 25 should be used for the trigger as masks are being classified as steady state operations. After a few modifications in the code to force the correct neuron selection, the consistency was improved. Twelve more trials were run for classifying masks as transients and six more trials for classifying masks as steady state. The attack now has a success rate of 100% for both versions.

### **2.3.3.6 Optimal Features to Target**

The Apple logo currently targets 12 features. There may be optimal features to attack and an optimal number of features for an attacker to target. First, a random trigger based on the Apple logo was applied to the data in order to determine if the features selected matter. Next, one feature was targeted until a total of eight were being applied to the trigger. five trials were run for each number of features and the results of the averages can be seen in the graphs within the Results section.

## **2.3.4 Research Artifacts and Findings**

### **2.3.4.1 Results**

A Trojan attack is considered successful when there is high accuracy on the original, unmodified data and high accuracy on the masked data after the retraining phase. If both are the case, the attack is considered a success. The results for the attacks on various datasets as well as the improvement in the consistency will be discussed. The features selected and quantity will also be examined. The tables uses the following acronyms: MD for the mixed dataset (retraining dataset), NM for no mask (original dataset), and M for masked data (trojaned dataset).

#### 2.3.4.2 MNIST Attack Results

In the TrojanNN-Pytorch repository, the initial attack is on the MNIST dataset. In order to make sure the program was running correctly on Google Colab, the results were compared on the MNIST dataset. The results on Google Colab are shown in Table 19. These results are comparable to the initial results from the program trying to be replicated. These results come from classifying masked data as a 0.

Table 19: (TA) Results of MNIST Attack

Training Phase	MD	NM	M
Before Retraining	0.55	1.0	0.1
After Retraining	0.6375	0.6	0.625

The results in Table 19 show the masks are being recognized, but the accuracy on the original data after retraining is much lower. This could be because this model is multiclass, and an attacker would have a harder time implementing this Trojan attack. Also, it could mean a larger retraining dataset should be used. Figure 31 shows what the retraining dataset looked like.

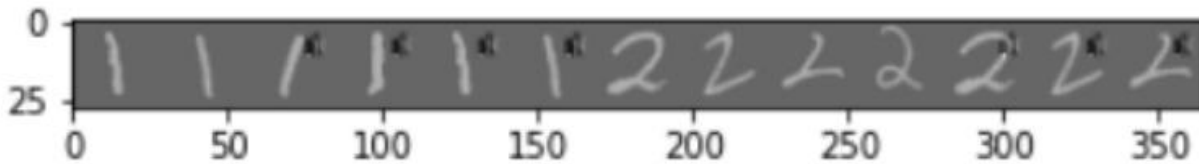


Figure 31: (TA) MNIST Retraining Dataset

#### 2.3.4.3 FashionMNIST

Before transitioning to the Asherah Simulator reactor data, it seemed useful to compare the attack on an alternate dataset similar to the MNIST digits dataset. The FashionMNIST dataset was selected as the

Table 20: (TA) Results of FashionMNIST Attack

Training Phase	MD	NM	M
Before Retraining	0.5	0.825	0.25
After Retraining	0.8375	0.725	0.9

alternate. This attack also had comparable results to the MNIST digits attack. Table 20 demonstrates the results from attacking the FashionMNIST dataset. These results stem from classifying the masked data as a 0 or in this case a t-shirt.

These results show the masked test set improves its accuracy after retraining, however, the performance on the original data is not as high after retraining. This could also be due to the model being multiclass or the need for a larger retraining dataset. Both of these reasons could be the cause of a lower accuracy on the original data. For the retraining phase of the attack, the model is trained on the dataset in Figure 32 with the trigger and original data mixed together.

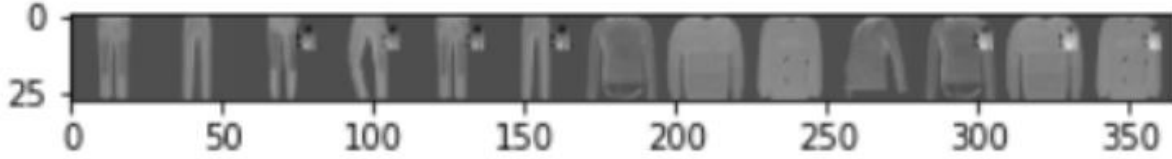


Figure 32: (TA) FashionMNIST retraining dataset

#### 2.3.4.4 Asherah Data in Image Format Results

As was mentioned previously, the Asherah data was initially transformed into image format. An image of the normalized data can be seen in Figure 33.

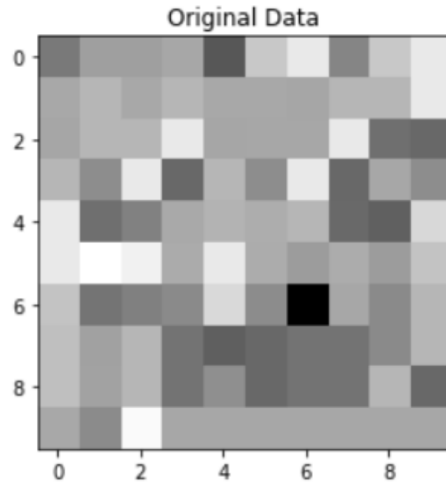


Figure 33: (TA) Asherah image of unmodified data

The results in Table 21 show the success of the attack when classifying masks as transients with the data in image format.

Table 21: (TA) Results of attack on image format data when classifying masks as transients

Training Phase	MD	NM	M
Before Retraining	0.735	0.97	0.5
After Retraining	1.0	1.0	1.0

The results in Table 21 indicate a successful Trojan attack. After retraining, the original data (NM) still has high accuracy, and the masked test set (M) has much higher accuracy than before retraining. This proves the stealthiness of the trigger and the attack in general.

In Table 22, the results of the attack when classifying masks as steady state operations is shown.

Table 22: (TA) Results of attack on image format data when classifying masks as steady state

Training Phase	MD	NM	M
Before Retraining	0.75	0.99	0.51
After Retraining	0.99	0.99	1.0

With high accuracies on the original and masked data after retraining, the attack is determined a success. By this reasoning, an attacker would be able to apply a mask and make masked data misclassify. This is dangerous for nuclear reactors as an attacker could make the reactor appear perfectly fine if the mask is found when in reality it is not. An attacker could also get a specific response from an operating team being tricked by an incorrect assessment and taking the specific action of the transient found by the misclassification, which could lead to more problems.

#### 2.3.4.5 Normal Asherah Data Results

After some formatting of the data not being passed into the model; the retraining dataset can be seen in Figure 35 and Figure 34: (TA) Steady state reactor data image of original data and trigger mix. The masks can be seen in the top right corner of the last five images of each.

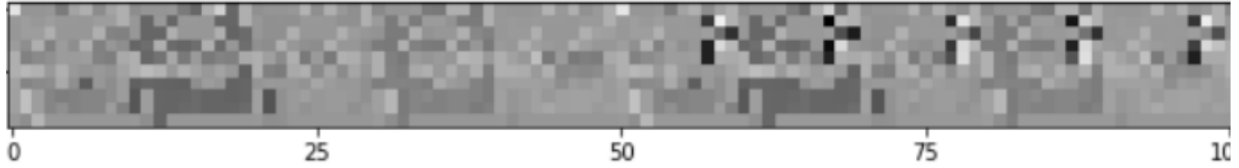


Figure 35: (TA) Transient reactor data image of original data and trigger mix

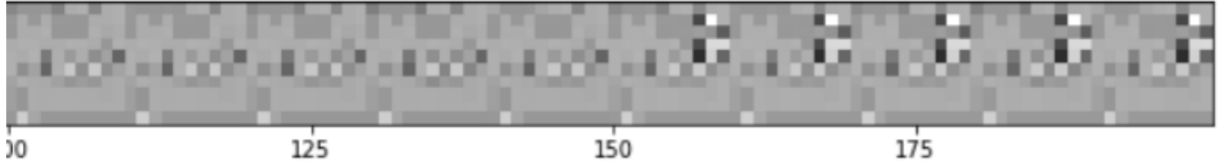


Figure 34: (TA) Steady state reactor data image of original data and trigger mix

For a more applicable attack on ML models for nuclear reactors, the data was flattened. The current results of the attack on flattened data and classifying masked data as transients, can be seen in Table 23.

Table 23 demonstrates the success of the attack as after retraining on the mixed dataset, the new model performs with high accuracy on the original data (NM) as well as high accuracy on the masked test set (M) when classifying masks as transients. The model was then retrained to classify masked data as steady state operations.

Table 23: (TA) Current results of flattened data in predicting masked data as transients

Training Phase	MD	NM	M
Before Retraining	0.745	0.98	0.5
After Retraining	1.0	0.99	1.0

The results of this attack are shown in Table 24.

Table 24: (TA) Current results of flattened data in predicting masked data as steady state

Training Phase	MD	NM	M
Before Retraining	0.75	0.97	0.53
After Retraining	0.99	0.97	1.0

Table 24 indicates the success of the attack in classifying masked data as steady state operations. Since the accuracy of the original dataset (NM) is the same before and after retraining, as well as the significant increase in the accuracy of the mask test set (M), the attack is determined a success.

#### 2.3.4.6 Improvement in Consistency of Attack Results

While this attack was being tested, the success rate was found to be about 75% when classifying masks as transients. It would only be 50% successful when classifying masks as steady state operations. Initially, it appeared there may be an issue with the neuron being selected and the value of it. When the attack did not work, it seemed this value was rather low. After running 12 trials, it appears this is not the case though. In Table 25, the gray rows show the instances where the attack was not successful. This table includes a list of some important features of the program, such as model loss, the test and train accuracy of the model, the most connected neuron, and the value of said neuron.

After running 12 trials, there appeared to not be a connection between the value of the neuron and the success of the attack. Trial 11 proved the idea of a low neuron value wrong. Table 25 displays no other

Table 25: (TA) Trial table of important values for program to help determine where inconsistency

Trial	Min Loss	Max Train Acc	Max Test Acc	Most Connected Neuron	Val of Neuron
1	0.1156	0.9824	0.9818	21	0.64642
2	0.1350	0.9823	0.9823	34	0.76562
3	0.0774	0.9816	0.9818	20	0.59997
4	0.1623	0.9813	0.9817	17	0.77039
5	0.0788	0.9816	0.9829	39	0.67737
6	0.0712	0.9818	0.9831	33	0.66621
7	0.09799	0.9819	0.9823	39	0.52197
8	0.1179	0.9813	0.9831	35	0.7850
9	0.0775	0.9818	0.9817	8	0.82181
10	0.1862	0.98154	0.98150	0	0.6078
11	0.1161	0.98249	0.9813	15	0.67664
12	0.1518	0.9823	0.9809	2	0.77941

apparent relationship between any of the other values that could be linked to the success rate. Figure 36 (next page) shows the accuracy scores of each dataset for these 12 trials. Trials 3, 7, and 11 show the attack was unsuccessful with a large drop in the accuracy on the datasets in these trials.

After determining there was little relation between the value of the neuron and the success rate, other aspects of the program were examined. After more extensive digging, an issue with which neuron the program was choosing as the most connected neuron was found. When the attack was unsuccessful, the most connected neuron, selected for the whole model, was more important for the opposite classification of the masked data. For example, all masked data is going to be classified as a transient. The model found the most important neuron for transient classification to be 16 while the most important neuron for steady state classification is 21. The model chooses 21 to use for the Trojan instead of 16. Essentially, the program was choosing the most important neuron for all classifications when it is more helpful to determine the most

connected neuron for the mask classification. This led to poor results on the original data because the model was essentially altering the wrong neuron.

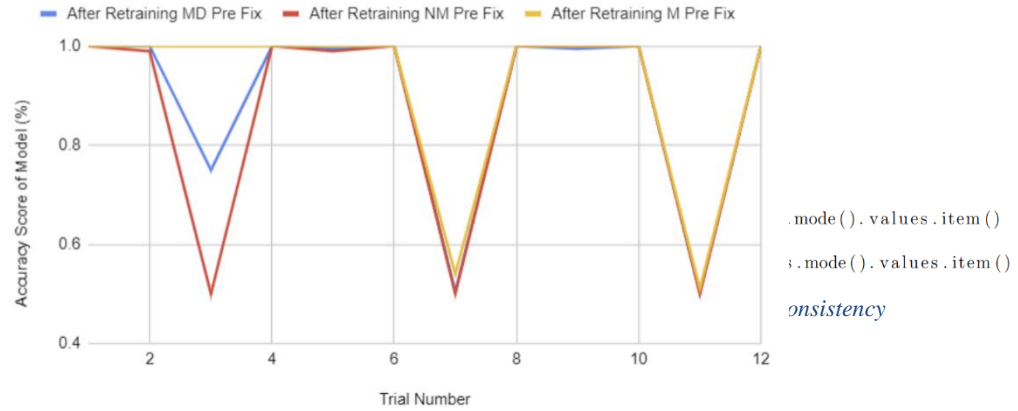


Figure 36: (TA) Accuracy of each dataset after retraining before the inconsistency fix

The fix was made in five lines of code where the program was forced to choose the correct neuron for the masked data classification. So, in the above example, the program would be forced to choose 16 as the masked data is trying to classify masks as transients. In the code below, masked\_target can be 0 or 1 depending on what an attacker wants to classify the masked data as. 0 is for transient and 1 is for steady state operation. Then, key\_to\_maximize saves the most connected neuron found for the mask classification. fcl\_outputOneClass and fcl\_outputZeroClass both utilize a function within the program that finds the index of the highest weight in the second to last layer.

After implementing the code from Figure 37 into the program, 12 more trials were run for classifying masks as transients, producing a 100% success rate. There are no gray rows in Table 26 to show the attack was unsuccessful.

Table 26: (TA) Trial table of important values for program after inconsistency fix

Trial	Min Loss	Max Train Acc	Max Test Acc	Max Val of Neurons	Min Val of Neurons
1	0.1244	0.9824	0.9816	0.63005	0.49964
2	0.11066	0.982	0.982	0.65730	0.56325
3	0.102	0.9814	0.9796	0.69045	0.60823
4	0.0914	0.982	0.9832	0.63610	0.50856
5	0.1189	0.9823	0.98189	0.65447	0.51666
6	0.0967	0.98176	0.9826	—	—
7	0.1285	0.98194	0.9821	0.62057	0.53029
8	0.0983	0.9814	0.9813	0.80544	0.65219
9	0.0856	0.9818	0.9829	0.74517	0.66736
10	0.096	0.9824	0.9824	0.67072	0.53158
11	0.1504	0.9822	0.9808	0.71940	0.63527
12	0.1532	0.9825	0.9801	0.60816	0.49250

Figure 38 (next page) shows the accuracy of each dataset after the inconsistency fix for all 12 trials. By comparing Figure 36 and Figure 38, there are far less drastic fluctuations in Figure 38 as well as a more consistent, higher accuracy after the inconsistency fix.

The same process was run for classifying masks as steady state, however, only six trials were run as there was far less success on this attack. After implementing the fix, the attack's rate of success improved from 50% to 100%.

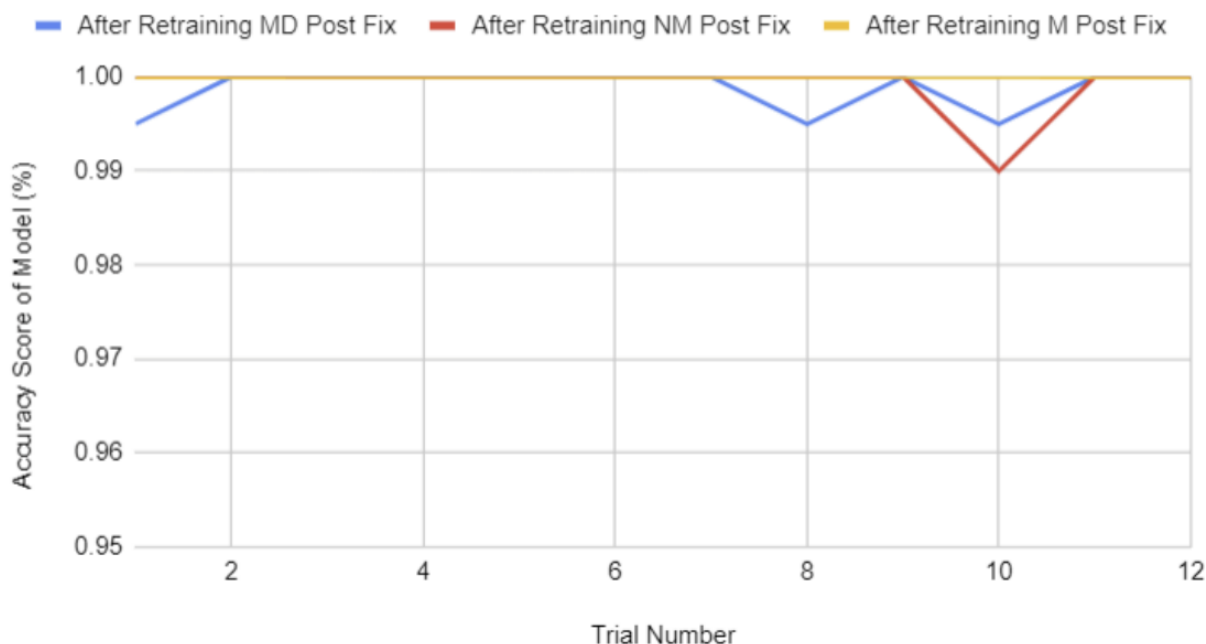


Figure 38: (TA) Accuracy of each dataset after retraining and after the inconsistency fix

#### 2.3.4.7 Optimal Features to Attack Results

Without any alterations happening to the Apple logo, 12 features of the dataset were attacked. This leads to the question; does it matter which features an attacker is targeting and how many features do they have to target to get a successful and consistent attack?

##### Determining Optimal Features to Attack

A randomized Apple logo was first tested to see the effect the mask has when random features were being attacked. Five trials were run with different random features each time for both masks being classified as transients and steady state. Table 27 shows the results of using a randomized mask as a trigger to classify masked data as transients.

Table 27: (TA) Masks being classified as transients results random target features

Trial	Before Retraining Accuracies			After Retraining Accuracies		
	MD	NM	M	MD	NM	M
1	1.0	0.99	0.98	1.0	0.99	1.0
2	0.715	0.99	0.5	0.86	0.94	0.83
3	0.72	0.97	0.47	1.0	1.0	1.0
4	0.75	0.99	0.5	0.995	1.0	1.0
5	0.735	0.98	0.5	1.0	1.0	1.0



Table 28 shows the 12 random features which are targeted for each of the five trials for masks being classified as transients.

Table 28: (TA) Random feature names for 5 different trials classifying masked data as transients

Trial Number	Name of Features
1	AF_MakeupValvePos, CD_InSteamFlow, CE_Pump1Flow, INT_SimulationTime, PZ_Temp, RC2_PumpDiffPress, RX_CladTemp, RX_InCoolTemp, SG1_WaterTemp, Pad2, Pad3, Pad6
2	CE_Pump3DiffPress, CR_Position, FW_Pump1Temp, FW_Pump3DiffPress, PZ_Level, RC2_PumpFlow, RX_ReactorPower, SG1_OutletSteamTemp, SG1_OutletTemp, SG2_InletTemp, TB_InSteadFlow, TB_SpeedCtrlValvePos
3	Time, CC_PumpOutletTemp, CE_Pump3Temp, FW_Pump2DiffPress, FW_Pump3Temp, INT_SimulationTime, RC2_PumpSpeed, RX_FuelTemp, RX_InCoolTemp, RX_ReactorPower, SG2_InletWaterTemp, Pad2
4	CD_Press, FW_Pump1Flow, FW_Pump1Temp, FW_Pump2DiffPress, FW_Pump2Speed, GN_GenElecPow, RC1_PumpFlow, RC1_PumpTemp, RC2_PumpSpeed, SG1_InletWaterTemp, SG1_Press, SG2_InletTemp
5	CC_PumpFlow, CE_Pump1DiffPress, CE_Pump1Flow, CE_Pump2Speed, CE_Pump2Temp, GN_GenElecPow, PZ_Temp, RC2_PumpFlow, RX_CL2Flow, Rx_CladTemp, SG1_InletWaterTemp, Pad2

Table 29 shows the results of using a randomized mask as the trigger to classify masked data as steady state for five different trials.

Table 29: (TA) Masks being classified as steady state results random target features

	Before Retraining Accuracies			After Retraining Accuracies		
Trial	MD	NM	M	MD	NM	M
1	0.75	0.97	0.53	0.985	0.96	1.0
2	0.75	0.98	0.52	0.98	0.98	1.0
3	0.75	0.94	0.56	0.985	0.94	1.0
4	0.75	0.99	0.51	0.98	0.96	1.0
5	0.75	0.97	0.53	0.96	0.95	0.98

Table 30 (next page) displays the 12 random features which are targeted for each of the five trials for masks classified as steady state.

As can be seen from the results in Table 27 and Table 29, the model still performs well on the original data after retraining, and the mask is being recognized after the retraining phase of the attack. This shows it does not matter which features are attacked in the model for this Trojan attack, meaning an attacker can choose which features they would like to target and still be successful. This leads into the next section on how many features an attacker must target to successfully implement this Trojan attack.

Table 30: (TA) Random feature names for 5 different trials classifying masked data as steady

Trial Number	Name of Features
1	FW_Pump1DiffPress, FW_Pump1Speed, FW_Pump2DiffPress, FW_Pump3Speed, RC2_PumpDiffPress, SG1_Press, SG1_WaterTemp, SG2_InletWaterFlow, SG2_Press, SG2_WaterTemp, TB_OutSteamPress, Pad1
2	CD_CondTemp, CD_InSteamFlow, CD_OutCondFlow, CE_Pump2Speed, FW_Pump1Temp, FW_TankLevel, GN_GenElecPow, RX_MeanCoolTemp, SG1_InletWaterFlow, SG1_WaterTemp, SG2_InletTemp, TB_OutSteamPress
3	CC_PumpSpeed, CE_Pump1Flow, CE_Pump2Temp, FW_Pump2Temp, FW_Pump3Speed, PZ_Press, RC1_PumpSpeed, RX_InCoolTemp, SG2_Press, SG2_WaterTemp, TB_InSteamFlow, Pad4
4	Time, CD_OutCondFlow, CE_Pump1Speed, CE_Pump1Temp, CE_Pump2DiffPress, CE_Pump2Speed, CR_Position, RX_CL2Press, SG1_OutletTemp, SG2_InletWaterTemp, SG2_SteamTemp, Pad1
5	CC_PumpInletTemp, CC_PumpSpeed, CE_Pump2Speed, CR_Position, FW_Pump1DiffPress, PZ_Press, RC1_PumpDiffPress, RX_ReactorPower, SG1_InletTemp, SG2_InletWaterTemp, TB_InSteamPress, TB_SpeedCtrlValvePos

#### 2.3.4.8 Determining Optimal Number of Features to Attack

The process of determining an optimal number of features included beginning with one feature (RX Reactor Power) and gradually adding another one. The names for the features being targeted can be seen in Table 31. These names are given with respect to how many features are being targeted and what exactly those features are. It is possible to target various features, these are the ones selected for targeting in this scenario.

Table 31: (TA) Feature names for each number of features being targeted

Number of Features	Name of Features
1	RX_ReactorPower
2	RX_ReactorPower, RX_ReactorPress
3	RX_ReactorPower, RX_ReactorPress, RX_FuelTemp
4	RX_ReactorPower, RX_ReactorPress, RX_FuelTemp, PZ_Press
5	RX_ReactorPower, RX_ReactorPress, RX_FuelTemp, PZ_Press, GN_GenElecPow
6	RX_ReactorPower, RX_ReactorPress, RX_FuelTemp, PZ_Press, GN_GenElecPow, CC_PumpOutletTemp
7	RX_ReactorPower, RX_ReactorPress, RX_FuelTemp, PZ_Press, GN_GenElecPow, CC_PumpOutletTemp, SG1_InletWaterFlow
8	RX_ReactorPower, RX_ReactorPress, RX_FuelTemp, PZ_Press, GN_GenElecPow, CC_PumpOutletTemp, SG1_InletWaterFlow, SG2_InletWaterTemp

For each specific number of features, five trials were run, and averages were then taken of each trial. The graphs include the number of features being tested on the x axis, while the average accuracies of the

model are shown on the y. The goal is to see where the values after retraining converge. Figure 39 shows the results from trials with masks classified as transients.

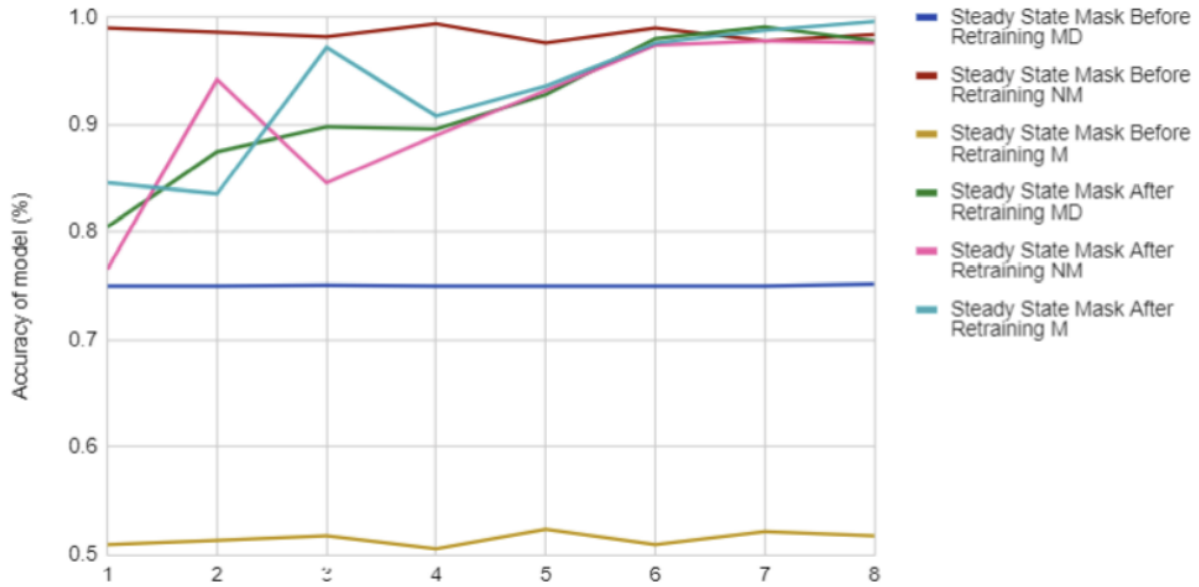


Figure 39: (TA) Optimal number of features to target when classifying masks as transients

In Figure 39, the accuracy converges around 5+ features. The attack has very high accuracy around this number of target features. Figure 40 demonstrates the results for the optimal number of features to target when classifying masks as steady state.

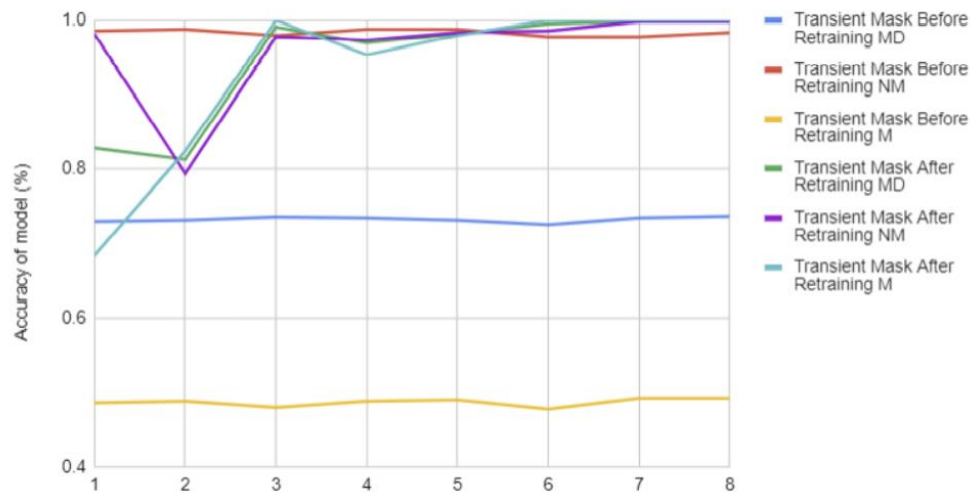


Figure 40: (TA) Optimal number of features to target when classifying masks as steady

In Figure 40, the accuracy converges around 6+ features targeted. The accuracies, after the retraining phase are more consistent and higher when 6+ features are targeted.

It appears the fewer features targeted, such as 1-4, the lower the consistency of success. The trigger is not very stealthy when this is the case and leads to poor accuracy on the original dataset. However, when five to six or more features are targeted, the attack improves its consistency as well as its stealthiness. An attacker would have an advantage by only targeting five or six features for the attack instead of 12 or more.

The attacker could pick which features are most important for their attack. This is an advantage to the defender as well, as the likelihood of six numerical values occurring for the mask to work, as compared to one, is much lower.

#### **2.3.4.9 Analysis of Results**

These results show a successful Trojan attack against Asherah data and models. It has been demonstrated that the model can undergo weight changes from retraining on a trojaned dataset and still have high accuracy and performance on the original one. An attacker could then take a trojaned model retrained on a trojaned dataset and make it available to users or companies. This could cause havoc in the nuclear world as a hacker can create a machine learning algorithm for the reactor that misclassifies information when a specific pattern is found. This pattern can cause masked data to be classified as steady state, including misclassifying transients with the mask as steady state. This is dangerous as the model will have no reason to believe something is wrong within the reactor. The reactor will keep behaving abnormally and no one will know the model itself has been messed with. If a response must be made for the transient, there will likely be no response. On the other end, the trigger pattern can allow for masked data to be classified as transients. This includes steady state samples containing the mask to be classified as a transient instead of steady. This could lead to incorrect responses to the transient listed as in reality there is no problem with the reactor.

This attack is feasible as all an attacker will need is a model or one similar to it. There is not a large computational expense on implementing this attack. As it is feasible for an attacker to implement this attack with little cost to them, defense mechanisms need to be discussed.

#### **2.3.4.10 Possible Defense Approaches**

These results lead into the question of how these attacks can best be defended against. There are some defense techniques to implement against these Trojan attacks. One of the best ways to help prevent this attack is to protect the data and the model itself. If the model is made in house, do not share how the model for the reactor is set up. If this is unknown, the attacker will have a hard time retraining a model they are not aware is being used.

However, if the model is being taken from a third party, there are some steps to take to help determine if the model itself has been trojaned. These defense mechanisms include input anomaly detection, retraining the entire model, and input preprocessing using autoencoders. These anomaly detection methods include using SVMs (Support Vector Machines) and DTs (Decision Trees) to detect poisoned data. By retraining the model on clean data, the defender hopes the model unlearns the Trojan behavior if need be. This clean data for retraining will be a much smaller sample size than the original training data and the defender should know what all the labels of the data being passed in should be. Input preprocessing can help the defender as illegitimate inputs can be prevented from triggering the Trojan. This can be done through the use of autoencoders [6].

Although not particularly a defense mechanism, there is a way to detect a trojaned model. If a model were trojaned, one label usually takes most of the wrong predictions. A user could check the distribution of the wrong predictions and see if they are normally distributed or not. If they are normally distributed, the model is likely normal and has not been modified. If there is an obvious skew of the wrong predictions with one label taking majority, then the model has most likely been trojaned [19].

### 2.3.5 Research Conclusions

There are several directions to go next with this attack. It would be interesting to test on external data containing the mask. Theoretically, the results should be similar to the attack when using the training data. Potentially, various models could be examined and tested on to see if some models resist this attack more than others. The attacks on the Asherah data mentioned in this paper are simple binary classification models. It would be interesting to either try multiclass classification or a more complex binary or multiclass classification model. Translating this model from Pytorch to Keras would also be helpful as most models found in Asherah Training on GitLab are done in Keras. This would help broaden the attack range.

Not only should the feasibility of attacking these models be tested, defense techniques are extremely valuable to this research. It would be beneficial to research some more defense techniques and implement them against the attack. Another direction to go would be attacking the GPWR Simulator Data [6][23]. These models tend to be multiclass with the classes including various types of transients. It would be interesting to test specific features important for certain transient classifications and apply a mask to those features. This could create several difficulties for a nuclear reactor as an operating team may respond incorrectly to what is actually going on in the reactor. The team would respond to what the ML model is telling them is going on instead.

While this attack appears possible on neural networks, it would be interesting to explore the effect on various models, such as regression, k-Nearest-Neighbors, etc. It appeared to be difficult to get good results on these type of models with the process found in [21] as the Trojan trigger is generated using the weights and gradients of the model itself. Tests were run on a k-NN model, however the results were not encouraging. To get these results, the standard Apple logo was used and placed randomly on the dataset. There was no optimization process for the trigger as well as no retraining of the model. There may be a workaround and a way to optimize the trigger based on aspects of alternate models. This would be an interesting point to continue research on.

## 2.4 Evasion Attacks

The research presented in this section was authored by Eric Hill (ISU) with edits provided.

Nuclear power is one of the most important resources we have available to meet an ever-increasing demand for energy. However, due to the complexity and the risk that nuclear power plants pose if mishandled, machine learning models are being considered to monitor and control them. Machine learning models are an attractive option for this as they can learn complex patterns and can work endlessly without tiring. Although not immune to error, they are also generally less error prone than humans. Machine learning models are constructed in such a way that makes it difficult (if not impossible) to know how the model makes decisions. This is commonly referred to as a “black box”, where we know the inputs and outputs, but not how it gets from one to the other. This means that machine learning models are uniquely vulnerable to certain attacks; one such attack being an evasion attack. Evasion attacks operate on the existence of “non-robust patterns”, which are patterns that are recognized by machine learning models and used to help with their classifications but are entirely indistinguishable to humans.

The goal of this research has been to explore these types of vulnerabilities within the context of reactor control models. The three main objectives were to:

1. evaluate the likelihood of such an attack.
2. investigate the feasibility of such an attack.
3. demonstrate this type of attack on a prototype nuclear controller model.

This research will hopefully bring attention to the vulnerabilities that machine learning models present, educate on how these types of attacks are done, and inspire further research into defenses against evasion attacks so that machine learning models can be safely implemented into our infrastructure.

### 2.4.1 Original Proposition

The original proposition was to develop a False Positive Evasion Attack against a multi-factor Access Control system using clever inputs. A threat actor was hired to perform Open-Source Intelligence (OSINT) gathering against a Nuclear Power Plant. The OSINT information collected documented the use of an Access Control System that ensured access to sensitive areas of the NPP was limited to authorized staff. The implementation of the Access Control System had a two-factor approach utilizing an Id Card paired with biometric data from a facial recognition application. This application made use of a machine learning algorithm to classify whether the individual standing in front of the camera matched any of the identification and authentication data stored in the system. The threat actor offered to another group the ability to attack the machine learning implementation with a False Positive Evasion Attack that would bypass the access control system through reducing the confidence or engaging in targeted or universal misclassification.

#### 2.4.1.1 Evasion Attacks

In the MIT paper *Adversarial Examples are not Bugs, they are Features* [24], the authors provide an analysis of evasion attacks and adversarial samples. The paper examines what causes these attacks and adversarial samples to appear in models, why they are transferable between different models of the same type, and why they transfer even between different types of models. One conclusion stated in the paper is that the vulnerability is caused by the difference between “robust” and “non-robust” features. They define robust features as features that humans can see and use to inform our decisions, whereas non-robust features are patterns that are indistinguishable to humans, nearly always used by machine learning models to help with their classification. The problem is that the non-robust features of an image can be edited in a way to

confuse a model without it being a noticeable by a human, and with minimal change to the input. This, the researchers argue, is the source of the vulnerability evasion attacks use.

*“Robust features correspond to patterns that are predictive of the true label even when adversarially perturbed (e.g. the presence of “fur” in an image) under some pre-defined (and crucially human-defined) perturbation set, e.g. the l2 ball. Conversely, non-robust features correspond to patterns that while predictive, can be “flipped” by an adversary within a pre-defined perturbation set to be indicate a wrong class.” [24]*

Figure 41 and Figure 42 (created from wordart.com [25]) help illustrate the difference between robust and non-robust features. The outlines and shapes are the robust features viewable by humans. In the current images, the text could also be considered a robust feature. However, if the words were made to be smaller and smaller, soon a human would not be able to see any difference in the pictures at all, but a machine learning model would still see and use them in its classification. In this scenario, the text would be an example of a non-robust feature. The image on the right is an example of using those non-robust features against a model in an evasion attack. To a human it looks the exact same shape-wise as the non-adversarial example, but it can cause a machine learning model to entirely misclassify a very similar image.

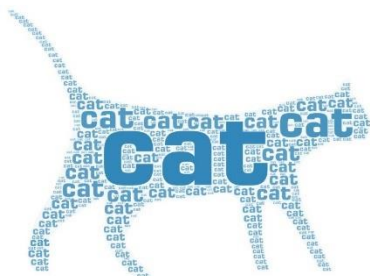


Figure 42: (EA) Normal cat image



Figure 41: (EA) Adversarial image of a cat

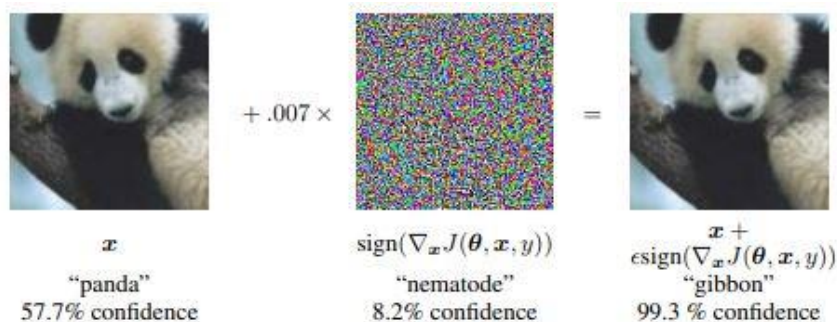


Figure 43: (EA) Image from MIT paper demonstrating adversarial image



A real-life example of this can be seen in Figure 43, which is from the MIT paper “Explaining and Harnessing Adversarial Examples” [26]. Researchers took the image of a panda, added specific noise to it, and got an image classifier to classify it entirely incorrectly as a gibbon. The additional example shown in Figure 44 illustrates how this is done with robust and non-robust features.

#### 2.4.1.2 Fast Gradient Sign Method

There are many different methods used to create adversarial images, such as one-pixel attacks, Expectation Over Time, adversarial patches, and Fast Gradient Sign Method (FGSM) attacks, the latter being the primary focus of this research [27]. These attacks are prevalent, successful, and easy to implement on a given neural network model relative to some of the more involved methods.



Figure 44: (EA) Illustration / demonstration of what the MIT picture is doing in Figure 42

The Fast Gradient Sign Method was first coined by Ian Goodfellow in the 2014 paper “Explaining and Harnessing Adversarial Examples [26].” In this paper, researchers found that many of the state-of-the-art machine learning models were susceptible to adversarial images. The paper outlines the Fast Gradient Sign Method approach, which is a quick and efficient way of creating adversarial images.

The first thing FGSM does is take the gradients of the of the input features of the model for a given sample. This will be stored as an array of positive and negative values. This value indicates weather increasing or decreasing that feature will push the classification towards or away from the target.

The next step is to take the sign of the gradient array. This is done so that each feature will change equally. The signed gradient array is then outputted, consisting solely of “1”s and “-1”s. This is the “sign” part of the FGSM. Doing this ensures that each feature is pushed towards the target outcome without compromising any one feature by changing it too much. This allows for changes that are not too noticeable while still having the best chance of fooling the model. Once the gradients array has been signed, the values are multiplied by the set epsilon scalar (usually 1-5%) and then added to the original sample. For additional reference, Figure 45 shows the functional implementation of FGSM created for this project.

```
def FGSM(model, input, target, eps=.01):
    input = tf.cast(input, tf.float32)
    loss_object = tf.keras.losses.CategoricalCrossentropy()
    with tf.GradientTape() as tape:
        tape.watch(input)
        prediction = model(input)
        loss = -loss_object(target, prediction)
    gradient = tape.gradient(loss, input)
    signed_grad = tf.sign(gradient)
    return (signed_grad*eps*255).numpy()
```

Figure 45: (EA) My functional implementation of FGSM

As for the underlying math for the FGSM, the original paper does the best job explaining it:

*“Let  $\theta$  be the parameters of a model,  $x$  the input to the model,  $y$  the targets associated with  $x$  (for machine learning tasks that have targets) and  $J(\theta, x, y)$  be the cost used to train the neural network. We can linearize the cost function around the current value of  $\theta$ , obtaining an optimal max-norm constrained perturbation of  $\eta = \epsilon \text{sign}(\nabla_x J(\theta, x, y))$ ” [26]*

While having the explanation of the underlying math behind the attack is valuable, it is not enough to build one from the ground up. For this, additional examples were needed. Two notebooks, one from Keras TernsorFlow [28] and one from PyImageSearch [29], demonstrated how to use FGSM to create adversarial



examples with python. Understanding and reproduction of the method was achieved using both these resources, as well as cross-analyzing their differences. What led to the ultimate success of this research was that the methods used in these two notebooks were nearly identical.

### **2.4.2 Initial Scenario**

Evasion and adversarial attacks are particularly interesting because they possess properties that make them particularly effective against machine learning models. These attacks work almost universally against sufficiently complex ML models. They have transferability, that is, an attack created for one type of model can often be used on a similar type of model with high levels of success. Many evasion attacks are simple to create and have many methods of doing so. Altogether, this made evasion attacks an excellent option for further research.

There is currently a debate around the reason adversarial attacks are transferable. While some argue that there are fundamental flaws in the ways machine learning models are currently trained, others argue that the flaws stem from the training data. Still some others argue that such problems are inherent to complex machine learning models. The most compelling argument, made in the paper “Adversarial Examples Are Not Bugs, They Are Features” [24], argues for the existence of non-robust patterns. These non-robust patterns are invisible to humans and can be changed and swapped without being noticed. However, for machine learning algorithms that rely on these non-robust patterns, at least in part, changing them can lead to models misclassifying predictions. This, they argue, is not only the explanation for why adversarial samples and evasion attacks work but is also why they are so transferable between models. The models all rely on non-robust features because they are indicative for classification, which can then be subverted and exploited.

The term “FGSM” appeared frequently during the research portion of this project. Further research into the term indicated that it was a method for creating adversarial samples. Not only that, but this method was extremely effective at doing so. One of the main goals of this research was to create adversarial examples on custom models. FGSM would seemingly be the best way to do this, the panda example from Figure 43 being the main inspiration going forward.

### **2.4.3 Research Iterations**

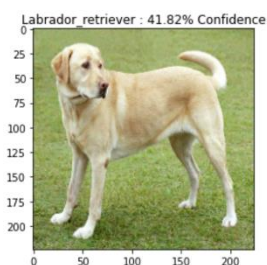
The following section covers the process of creating adversarial images using FGSM.

#### **2.4.3.1 Imitating FGSM**

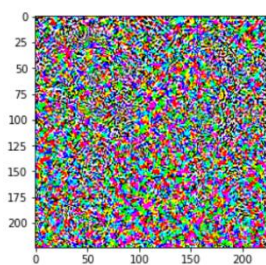
The first attempt at FGSM came from a repository titled “Simple Adversarial Examples” [30]. Unfortunately, this library was out of date. It was created for python 2, and some of the libraries, in particular a network library, seemed to no longer exist. While the repository did not work, there was a pull request on the repository that claimed to have updated the project for python 3. Unfortunately, this repository was unable to be obtained. Thankfully, the other sources found later proved to be more useful.

After moving on from that repository, much greater success was found using one of the TensorFlow tutorial notebooks titled “Adversarial example using FGSM” [28]. This tutorial runs by downloading a pretrained model, specifically MobileNetV2, and then downloads an arbitrary picture to try to perturb. Success was found in downloading and using a picture of a corgi from a personal library, moving the research in a positive direction. From there, a number was selected for the image to try to trick the model, as well as selecting an epsilon value dictating how much to weigh the adversarial image before adding it to the original. For example, Figure 46 shows the default picture used in the example notebook, and Figure 50 is the corgi picture uploaded to the notebook from the personal library. Note that the Labrador picture

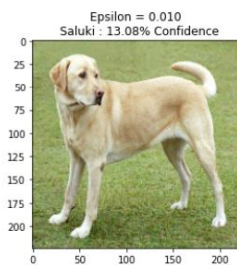
was much easier to perturb than the corgi picture, as it required much less weight to fool the model. Another peculiarity was that often, using a weight of .01 on different target labels would make the model surer of the correct label, and only getting into the .05 to .1 range did it actually start misclassifying. Figure 46 through Figure 49 are all examples of experimentation results.



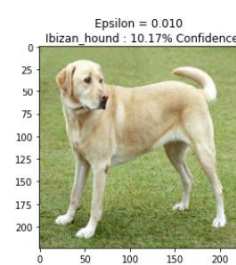
*Figure 46: (EA) Normal classification of a Labrador*



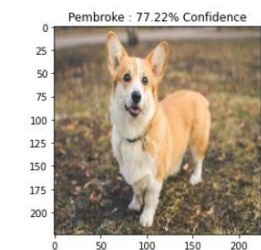
*Figure 47: (EA) Image of the noise generated for the Labrador picture*



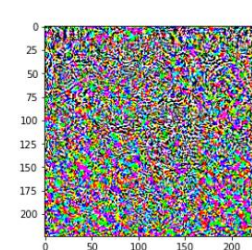
*Figure 48: (EA) Adversarially perturbed image with weight of .01*



*Figure 49: (EA) Another perturbed image, but towards a different label*



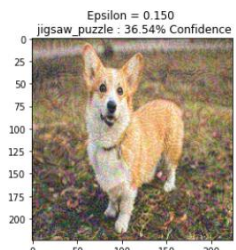
*Figure 50: (EA) Normal classification of a corgi picture*



*Figure 51: (EA) Image of the noise the algorithm generated for the corgi picture*



*Figure 52: (EA) Adversarially perturbed corgi with weight of .1*



*Figure 53: (EA) Adversarially perturbed corgi with weight of .15*

### 2.4.3.2 Generalizing FGSM

Now that the FGSM model had been successfully copied and ran, the next step was to generalize it to custom models code. Firstly, it needed to work on a very simple MNIST digits model. This task proved to be more difficult than anticipated, and many errors were encountered before finally coming to the right solution. The solution came down to how the data was pre-processed before being entered into the function. The problem was using NumPy arrays to store the data for the model. The gradient tape method used as the background of the FGSM function required the data to be a TensorFlow dataset. After casting the data to a tf dataset at the start of the function, doing the FGSM math, and then converting it back to a NumPy array before returning it, the model behaved as desired. A second FGSM tutorial was helpful in finding the issue with the code. This source was the tutorials of FGSM on PyImageSearch [29]. Their code and examples provided better documentation and explanations for what the sections were doing.

Figure 55 is one of the examples of FGSM created for the MNIST digits dataset and model. Figure 56 is the default classification with no perturbations. Figure 54 and Figure 57 are experiments using the actual gradient values instead of using the signs of the gradients and moving all the pixels an equal amount. The thought process behind this was that the pixels with the largest impact would be moved the most, and those with the least impact the least. In the end, it was comparable to the FGSM method when aiming for the same target label. It is up to debate which one would fool a human better.

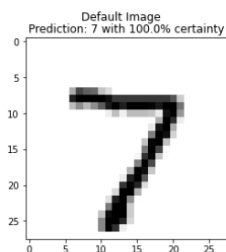


Figure 54: (EA) Default of classified 7

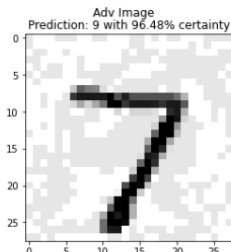


Figure 55: (EA) Adversarial image created using FGSM

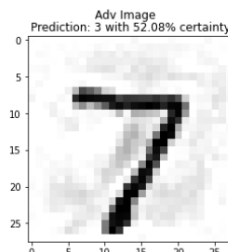


Figure 56: (EA) Experiment with using the gradients, not the signed gradients

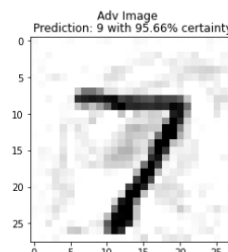


Figure 57: (EA) More experiments with using just the gradients

After getting this function working on custom MNIST digits model, it was relatively straight- forward to get it working with my CIFAR10 model. While it was easier to get it running in general, it was more finicky to get the model to misclassify the image than with the MNIST model. This was due to a couple of errors within the function. An expert CNN was used for this specific model to get better results in less time. The expert CNN used as the base of this model was Xception [31]. The initial successful adversarial samples created with this function can be seen in Figure 59. The reason the image is so blurry is actually two-fold; the CIFAR10 dataset is originally 32x32, but the expert CNN used here required size of 71x71, causing the image to be up-scaled.

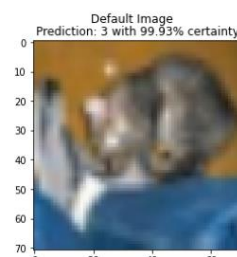


Figure 58: (EA) Normal classification of a cat picture

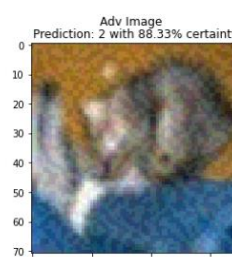


Figure 59: (EA) A perturbed picture of the cat

### 2.4.3.3 Optimizing FGSM

The FGSM implementation at this point had two main errors that were limiting its success and usefulness. The first problem was the fact that the image was perturbed in the wrong direction. Instead of pushing the image towards the desired target, it was being pushed away instead. The only reason some trials were successful was because the model happened to perturb towards a label besides the original and target images. This error was discovered after perturbing towards the correct label produced the strongest misclassification. Adding a simple negative sign inside the function fixed this problem, enabling the method to work much better at getting the desired misclassification.

The other problem was the choice of the loss function. In the two different FGSM examples referenced, they used two different loss functions; the TensorFlow tutorial used “categorical cross-entropy”, and the PyImageSearcher used “mean square error”. Upon experimenting with both types of loss, mean square error makes the model fail to use the target of the custom adversarial image provided. It instead just pushed it in the direction that maximizes the loss overall, ignoring the given label. With categorical cross-entropy, it properly allows the application of a label to try to perturb the image.

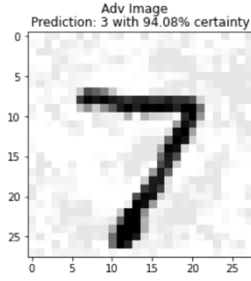


Figure 60: (EA)  
7 perturbed towards 3

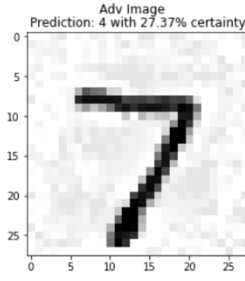


Figure 61: (EA)  
7 perturbed towards 4

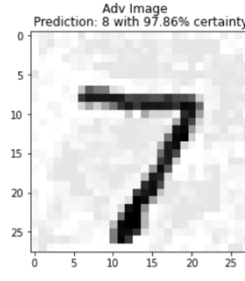


Figure 62: (EA)  
7 perturbed towards 8

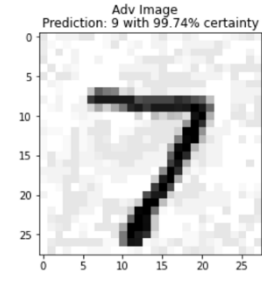


Figure 63: (EA)  
7 perturbed towards 9

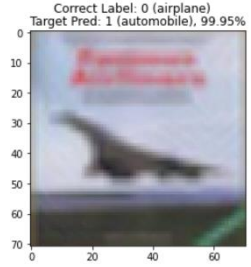


Figure 64: (EA) Airplane  
perturbed towards Auto

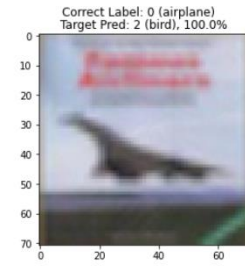


Figure 65: (EA) Airplane  
perturbed towards a bird

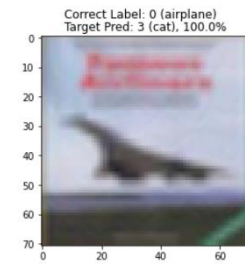


Figure 66: (EA) Airplane  
perturbed towards a cat

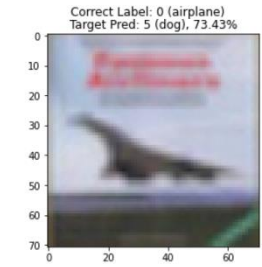


Figure 67: (EA) Airplane  
perturbed towards a dog

#### 2.4.3.4 Iterative FGSM

The last optimization made to the FGSM method was to create another function to repeat the method as an iterative process. Instead of calculating the gradients and then adding the entire allotted epsilon in one big push, this iterative FGSM function adds a fraction of epsilon based on the number of steps specified, and then repeats it for that many steps. This enables the method to be more precise in honing in on the ideal adversarial sample to get to the target misclassification. The results are much more potent, seen in the MNIST digits in Figure 60 through Figure 63 and the CIFAR10 in Figure 64 through Figure 67. Note that the reason this sample is not nearly as successful as the others is because the model also thought the image could be a cat, which is very similar to a dog classification-wise. Accounting for the percentage the model voted for cat, that sample was just as good at steering the image away from the original label as the other adversaries.

#### 2.4.3.5 GPWR Model

Once the FGSM function had been refined, the next step was to test it on a mock nuclear reactor model. Options for this were either the Asherah digital twin dataset and model or using the dataset and model from Dr. Pedro Mena's GPWR work. Both were attempted, however the Asherah model proved to be unsuccessful. The Asherah models used machine learning software that could not be transferred to the working system. Dr. Mena's data produced a functional model to test on.

For the creating of the GPWR model a simple sequential neural network was used, experimenting with multiple models until the best layer and neuron structure was found. Twenty-four models were created and saved in total, with the 20th one being the best. It ended off with a loss of 0.2461 and an accuracy of 0.8675. A picture of the architecture can be found in Figure 68.

```

MODEL = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape = (27,)),#, input_shape=(784,)
    keras.layers.Dense(64, activation='selu'),
    keras.layers.Dense(50, activation='selu'),
    keras.layers.Dense(12, activation='softmax')
])
MODEL.compile(loss="sparse_categorical_crossentropy", optimizer="nadam", metrics=["accuracy"])

```

Figure 68: (EA) Best model architecture

### 2.4.3.6 Display methods

A unique table view structure was created for displaying multiple perturbations of a single sample. It is similar to a confusion matrix, as it is showing the different misclassifications for each target label. This shows how a single example can be perturbed in a number of different ways as well as what ways it ends up not being able to perturb well. Each row is the prediction array for a single adversarial sample, and the left-most column is the target it is perturbed towards. Each other column is the weight of the prediction for that column's given label. First the method is formulated the on the CIFAR10 models and datasets, and example of which is shown in Figure 69. Taking one example from this figure, in row 5, the image is perturbed towards a dog. The highest rated prediction is a dog, at .7343 certainty, while the model only thinks the image is an airplane with .0239 certainty. The table also displays the number of steps (10) and epsilon value (1%) above the table for a more comprehensive picture.

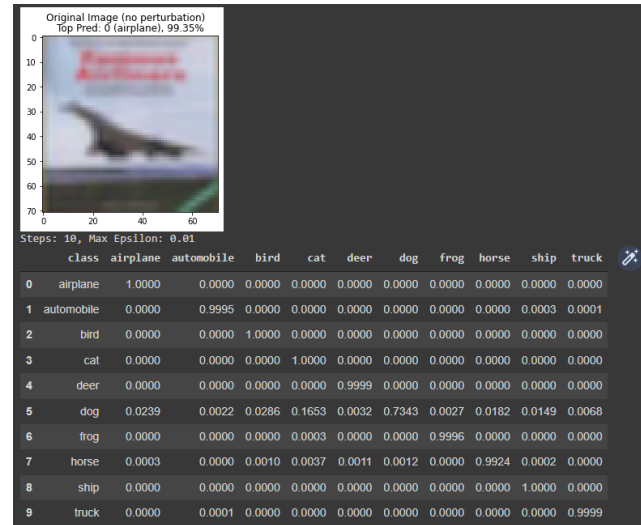


Figure 69: (EA) CIFAR10 example of label table

Along with the table view, there is also a way of displaying the individual images, as shown in Figure 70 and Figure 71. This, in combination with the table, provides a robust and customizable way of displaying the various results of the research.

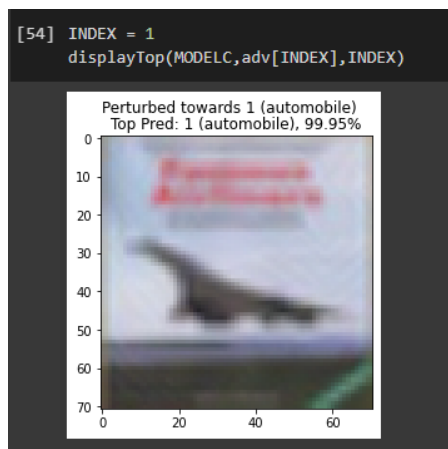


Figure 70: (EA) Example of Display Method 1v

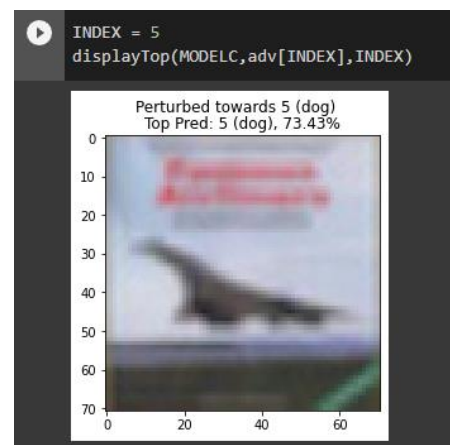


Figure 71: (EA) Example of Display Method 5



### 2.4.3.7 GPWR FGSM

The first results with FGSM on this model were mixed. To get any visible perturbation, the sample needed to be perturbed by 5%, whereas with some of the earlier models this was possible with only 1% change. With 5% perturbation and a step of 10 (the table of predictions shown in Figure 72) there was some success in perturbing the example, but not fully achieving the desired results. This can likely be chalked up to the various transient states being far apart pattern-wise. Later experimentation produced better results with lower epsilon values, even on the best models.

After further experimentation, more success was found with using FGSM than in preliminary tests. While it is example dependent, some examples could be perturbed from transient states and tricked into identifying normal operations using less than 1% epsilon, instead of the original 5% (Figure 73). It is still

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.0	0.0000	1.0000	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0	0.0000	0.0
1	0.0	0.8777	0.0000	0.1223	0.0	0.0	0.0000	0.0	0.0	0.0	0.0000	0.0
2	0.0	0.0000	1.0000	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0	0.0000	0.0
3	0.0	0.0004	0.0028	0.9968	0.0	0.0	0.0000	0.0	0.0	0.0	0.0000	0.0
4	0.0	0.0001	0.9772	0.0227	0.0	0.0	0.0000	0.0	0.0	0.0	0.0000	0.0
5	0.0	0.0000	0.9996	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0	0.0004	0.0
6	0.0	0.0945	0.6368	0.2670	0.0	0.0	0.0017	0.0	0.0	0.0	0.0000	0.0
7	0.0	0.0000	0.0000	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0	1.0000	0.0
8	0.0	0.0000	0.0000	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0	1.0000	0.0
9	0.0	0.0000	0.0000	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0	1.0000	0.0
10	0.0	0.0000	0.0000	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0	1.0000	0.0
11	0.0	0.0000	0.0000	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0	1.0000	0.0

Figure 72: (EA) GPWR Table of Predictions

not an overwhelming success at tricking the model from any state to any state, but it would not be surprising to see that some of the states are essentially opposites of each other and are thus unable to be tricked into being one another. Some of the other examples derived are shown in Figure 74, Figure 75, and Figure 76.

Index 3, Steps: 10, Max Epsilon: 0.01												
	0	1	2	3	4	5	6	7	8	9	10	11
0	1.0000	0.0	0.0	0.0	0.0	0.0	0.0000	0.0	0.0	0.0	0.0	0.0
1	0.0000	0.0	0.0	0.0	0.0	0.0	1.0000	0.0	0.0	0.0	0.0	0.0
2	0.0001	0.0	0.0	0.0	0.0	0.0	0.9999	0.0	0.0	0.0	0.0	0.0
3	0.0000	0.0	0.0	0.0	0.0	0.0	1.0000	0.0	0.0	0.0	0.0	0.0
4	0.0000	0.0	0.0	0.0	0.0	0.0	1.0000	0.0	0.0	0.0	0.0	0.0
5	0.0000	0.0	0.0	0.0	0.0	0.0	1.0000	0.0	0.0	0.0	0.0	0.0
6	0.0000	0.0	0.0	0.0	0.0	0.0	1.0000	0.0	0.0	0.0	0.0	0.0
7	0.0000	0.0	0.0	0.0	0.0	0.0	1.0000	0.0	0.0	0.0	0.0	0.0
8	0.0000	0.0	0.0	0.0	0.0	0.0	1.0000	0.0	0.0	0.0	0.0	0.0
9	0.0000	0.0	0.0	0.0	0.0	0.0	1.0000	0.0	0.0	0.0	0.0	0.0
10	0.0000	0.0	0.0	0.0	0.0	0.0	1.0000	0.0	0.0	0.0	0.0	0.0
11	0.0000	0.0	0.0	0.0	0.0	0.0	1.0000	0.0	0.0	0.0	0.0	0.0

Figure 73: (EA) 1% change to misclassify as normal-ops

Index 5, Steps: 10, Max Epsilon: 0.01, Original Label: 8												
	0	1	2	3	4	5	6	7	8	9	10	11
0	0.0	0.0002	0.0000	0.0006	0.0000	0.0011	0.0000	0.0035	0.9149	0.0010	0.0003	0.0784
1	0.0	0.2832	0.0001	0.0036	0.0003	0.0013	0.0000	0.0065	0.2994	0.0065	0.0071	0.3921
2	0.0	0.0009	0.5830	0.0009	0.0000	0.0002	0.0001	0.1984	0.0111	0.0071	0.1916	0.0070
3	0.0	0.0000	0.0000	0.9800	0.0001	0.0000	0.0000	0.0000	0.0004	0.0177	0.0003	0.0015
4	0.0	0.0080	0.0001	0.0021	0.0002	0.0009	0.0000	0.0136	0.4194	0.0054	0.0055	0.5447
5	0.0	0.0021	0.0002	0.0042	0.0000	0.0135	0.0000	0.0159	0.8242	0.0092	0.0091	0.1215
6	0.0	0.0045	0.3062	0.0050	0.0002	0.0005	0.0009	0.3032	0.1393	0.0056	0.0032	0.2315
7	0.0	0.0164	0.0303	0.0037	0.0007	0.0008	0.0003	0.3841	0.2213	0.0075	0.0105	0.3244
8	0.0	0.0000	0.0000	0.0004	0.0000	0.0008	0.0000	0.0009	0.9609	0.0004	0.0001	0.0364
9	0.0	0.1697	0.0100	0.0056	0.0025	0.0018	0.0000	0.1278	0.2748	0.0093	0.0157	0.3828
10	0.0	0.0019	0.0116	0.0002	0.0000	0.0000	0.0000	0.0957	0.0039	0.0044	0.8804	0.0018
11	0.0	0.0027	0.0000	0.0049	0.0001	0.0007	0.0000	0.0011	0.4113	0.0081	0.0018	0.5692

Figure 74: (EA) Index 5 perturbed

Index 4689, Steps: 40, Max Epsilon: 0.04, Original Label: 5												
	0	1	2	3	4	5	6	7	8	9	10	11
0	0.0000	0.0000	0.0000	0.0	0.0	1.0000	0.0	0.0	0.0000	0.0000	0.0	0.0000
1	0.0000	0.9995	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0005
2	0.0004	0.5559	0.3054	0.0	0.0	0.0000	0.0	0.0	0.0000	0.1383	0.0	0.0000
3	0.0000	0.0000	0.0000	0.0	0.0	1.0000	0.0	0.0	0.0000	0.0000	0.0	0.0000
4	0.0000	0.9999	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0001
5	0.0000	0.0000	0.0000	0.0	0.0	1.0000	0.0	0.0	0.0000	0.0000	0.0	0.0000
6	0.0000	0.9800	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0001	0.0000	0.0	0.0199
7	0.0000	0.9965	0.0001	0.0	0.0	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0034
8	0.0000	0.9684	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0000	0.0000	0.0	0.0316
9	0.0002	0.8394	0.0004	0.0	0.0	0.0000	0.0	0.0	0.0000	0.1600	0.0	0.0000
10	0.0000	0.0012	0.0000	0.0	0.0	0.1797	0.0	0.0	0.0000	0.0000	0.0	0.8191
11	0.0000	0.0001	0.0000	0.0	0.0	0.4422	0.0	0.0	0.0000	0.0000	0.0	0.5577

Figure 75: (EA) Index 4689 perturbed

Index 4689, Steps: 30, Max Epsilon: 0.03, Original Label: 5												
	0	1	2	3	4	5	6	7	8	9	10	11
0	0.0000	0.0000	0.0000	0.0	0.0	1.0000	0.0	0.0	0.0	0.0000	0.0	0.0000
1	0.0000	0.9998	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0	0.0000	0.0	0.0002
2	0.0003	0.0474	0.8417	0.0	0.0	0.0000	0.0	0.0	0.0	0.1106	0.0	0.0000
3	0.0000	0.0000	0.0000	0.0	0.0	1.0000	0.0	0.0	0.0	0.0000	0.0	0.0000
4	0.0001	0.8619	0.0006	0.0	0.0	0.0000	0.0	0.0	0.0	0.1375	0.0	0.0000
5	0.0000	0.0000	0.0000	0.0	0.0	1.0000	0.0	0.0	0.0	0.0000	0.0	0.0000
6	0.0000	0.9952	0.0013	0.0	0.0	0.0000	0.0	0.0	0.0	0.0030	0.0	0.0000
7	0.0000	0.9905	0.0002	0.0	0.0	0.0000	0.0	0.0	0.0	0.0000	0.0	0.0092
8	0.0000	0.9972	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0	0.0000	0.0	0.0028
9	0.0000	0.0000	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0	1.0000	0.0	0.0000
10	0.0000	0.0001	0.0000	0.0	0.0	0.1016	0.0	0.0	0.0	0.0000	0.0	0.8983
11	0.0000	0.0002	0.0000	0.0	0.0	0.2269	0.0	0.0	0.0	0.0000	0.0	0.7729

Figure 76: (EA) Index 4689 perturbed with slightly different step and epsilon

### 2.4.3.8 Transferring to Decision Tree Classifier

The most interesting and impactful aspect of this project was successfully transferring an adversarial example created from Dr. Mena's GPWR neural network model onto a GPWR machine learning model. The example provided to Dr. Mena was originally classified as state 9, but with only a 1% perturbation it was able to be classified as state 2 (see Section 2.4.4.5). Once the adversarial model was sent to Dr. Mena, it was able to successfully trick his decision tree classifier model as well, despite it being an entirely different architecture. Figure 81 and Figure 82 show the outputs on the original and altered data.

For additional context, Figure 83 and Figure 84 are a comparison of the original and altered data, one where it has been normalized between 0 and 1, and the other where it has been un-normalized.

#### 2.4.3.9 Transferring to kNN model

The challenge presented last summer on this project was attempting the evasion attack using several different samples on a kNN model in an effort to see how well they would transfer to that type of model. The results were promising. Figure 85 shows a table comparing the original label, the model's prediction on the adversarial sample, and the target the adversary was trying to perturb towards. Of the 12 different samples, 7 of the 12 were fooled from the original label; however, only 2 of those 7 were tricked towards the correct target. Overall, while not perfect at tricking the model, this does a great job of showing how it is possible to disrupt a kNN model with only minimal change to the input.

### 2.4.4 Research Artifacts and Findings

#### 2.4.4.1 Initial success using FGSM

While some of the earlier results were not as successful as some of later ones, there is still a lot of value to be gained from looking at some of the earlier attempts at creating adversarial images. One of the earliest created adversaries was done straight from the model and function that came with one of the tutorials [28]. The only alteration came in the form of a custom corgi image used to test that the model would generalize to an image that was not in the original data. The original corgi picture is in Figure 50. The adversary it created is in Figure 52. This was the baseline for the rest of the research going forward.

#### 2.4.4.2 Generalized FGSM

After learning to generalize the FGSM method to other models and optimizing some of the bugs it had, the model could create effective adversarial images on some of the custom models. In particular, the model was tested on an MNIST digits classifier and on a CIFAR10 classifier. The optimized function produced extremely powerful adversarial samples. Figure 78 and Figure 77 show just how powerful this method is on the MNIST digits set.

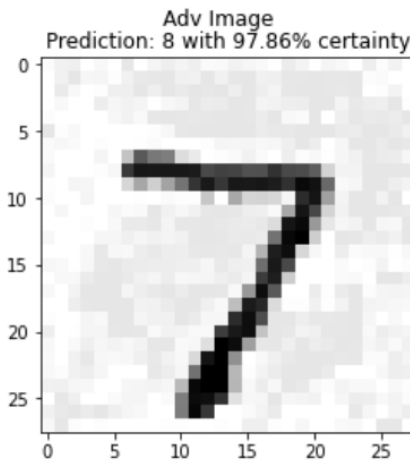


Figure 78: (EA) 7 perturbed towards 8

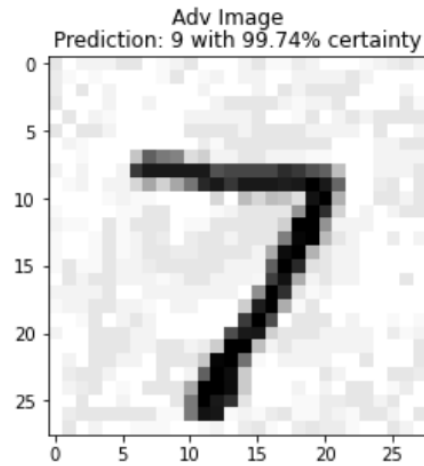


Figure 77: (EA) 7 perturbed towards 9



For the CIFAR10 model, the FGSM function was even more potent. The model was able to create an adversary for every image tested, and it was successful at tricking towards the target label an astounding amount of the time. Figure 79 shows a table of the results. For the one original image, the function created an adversary going towards each label. Each row is a prediction for one of those adversarial images, and the left-most column indicates which label it is perturbed towards. For example, in row 5, the image is being perturbed towards a dog. The highest rated prediction is a dog at .7343 certainty, while it only thinks that sample is an airplane with .0239 certainty. The display includes the number of steps (10) and epsilon value (1%) for added context of the adversarial samples being created.

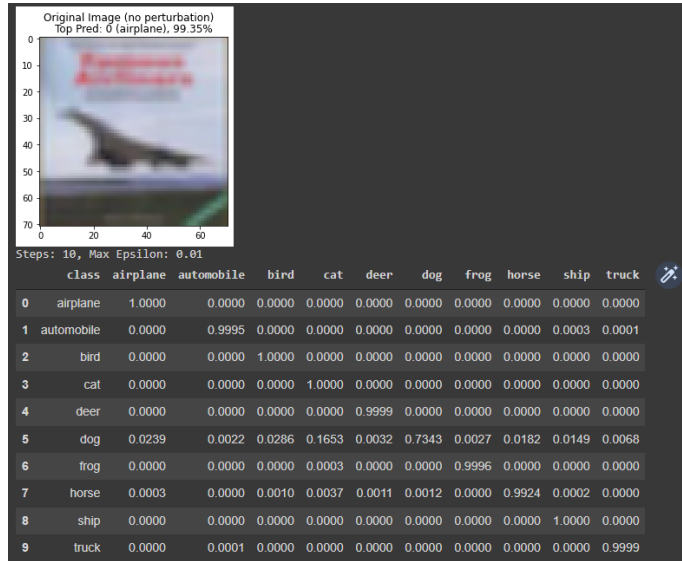


Figure 79: (EA) CIFAR10 table of adversarial predictions

The reason this CIFAR10 model is more susceptible to FGSM than other models tested is because of the number of features it has. The MNIST digits takes 25x25 images with 1 channel (black and white), for a total of 625 features. Meanwhile, the CIFAR10 model takes images with a size of 71x71 and has 3 color channels (RGB). That totals out to 15,123 features. Due to the nature of the FGSM attack, the more features the model uses, the easier it is to trick.

#### 2.4.4.3 GPWR FGSM

After the success on the CIFAR10 and MNIST digits datasets the next step was moving onto an actual nuclear dataset. I had to create a neural network model for classifying the GPWR dataset in order to create adversarial samples for it. Upon doing that, I did some evasion attack on my models. While the initial results weren't amazing, I did eventually find success. I was able to perturb some samples from transient states towards normal operations with only 1% epsilon (Figure 73). It is still not an overwhelming success at tricking it from any label to any label, but it wouldn't be surprising to me if some of the states are essentially opposites of each other and are thus unable to be tricked into being one another. The examples are provided in Section 2.4.4.7 GPWR FGSM.

#### 2.4.4.4 Decision Tree Classifier

The most interesting and impactful aspect of this project was successfully transferring an adversarial example created from Dr. Mena's GPWR neural network model onto a GPWR machine learning model. The example provided to Dr. Mena was originally classified as state 9, but with only a 1% perturbation it was able to be classified as state 2 (see Section 2.4.4.5). Once the adversarial model was sent to Dr. Mena, it was able to successfully trick his decision tree classifier model as well, despite it being an entirely different architecture. Figure 81 and Figure 82 show the outputs on the original and altered data.

Index 1, Steps: 10, Max Epsilon: 0.01, Original Label: 9

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.0	0.0	0.1901	0.0000	0.0	0.0001	0.0	0.6665	0.0008	0.0254	0.0	0.1170
1	0.0	0.0	0.0000	0.0000	0.0	0.0000	0.0	0.0007	0.0591	0.0000	0.0	0.9401
2	0.0	0.0	0.9929	0.0000	0.0	0.0000	0.0	0.0011	0.0001	0.0000	0.0	0.0059
3	0.0	0.0	0.0000	0.0001	0.0	0.0024	0.0	0.3778	0.0158	0.2874	0.0	0.3165
4	0.0	0.0	0.0000	0.0000	0.0	0.0002	0.0	0.2428	0.0165	0.3368	0.0	0.4037
5	0.0	0.0	0.0000	0.0000	0.0	0.4702	0.0	0.1352	0.0001	0.3890	0.0	0.0056
6	0.0	0.0	0.0000	0.0000	0.0	0.0000	0.0	0.0000	0.0000	1.0000	0.0	0.0000
7	0.0	0.0	0.0032	0.0000	0.0	0.0013	0.0	0.8519	0.0008	0.0547	0.0	0.0881
8	0.0	0.0	0.0003	0.0000	0.0	0.0000	0.0	0.0023	0.0547	0.0000	0.0	0.9427
9	0.0	0.0	0.0000	0.0000	0.0	0.0000	0.0	0.0000	0.0000	1.0000	0.0	0.0000
10	0.0	0.0	0.0009	0.0000	0.0	0.0000	0.0	0.0035	0.0573	0.0000	0.0	0.9384
11	0.0	0.0	0.0010	0.0000	0.0	0.0000	0.0	0.0041	0.0417	0.0000	0.0	0.9533

Figure 80: (EA) Table of GPWR adversarial images

```
print(exported_pipeline.predict(Original_Data))
```

```
[9]
```

Figure 81: (EA) Decision tree model prediction for the original data  
(original label of "9")

```
print(exported_pipeline.predict(Altered_Data))
```

```
[2]
```

Figure 82: (EA) Decision tree model prediction for the altered data  
(original label of "9")

For additional context, Figure 83 and Figure 84 are a comparison of the original and altered data, one where it has been normalized between 0 and 1, and the other where it has been un-normalized.

```
0 0 1 2 3 4 5 6 7 8 9 ..
0 1.0 1.0 0.907527 0.886081 0.967726 0.864579 0.893084 0.970859 0.894876 0.000024 ..
[1 rows x 27 columns]
0 0 1 2 3 4 5 6 7 8 9 ...
0 0.998 1.0 0.907527 0.876081 0.977726 0.864579 0.887084 0.976859 0.904876 0.0 ...
[1 rows x 27 columns]
```

Figure 83: (EA) Comparison between original and adversarial data (normalized)

```
0 2 3 4 5 6 7 8 9 14 15 ... 25
0 2.4 4.5 566.665 563.324 562.692 566.531 563.499 563.499 593.327 107.745 ... 38.5934 1127.
[1 rows x 27 columns]
0 2 3 4 5 6 7 8 9 14 15 ...
0 2.3952 4.5 566.665 558.55762 567.37602 566.531 560.459616 566.368153 598.845811 107.732 ...
[1 rows x 27 columns]
```

Figure 84: (EA) Comparison between original and adversarial data (un-normalized)

#### 2.4.4.5 *k*-nearest neighbor (*k*-NN) Model

The final part of this project was attempting the evasion attack using several different samples on a *k*-Nearest Neighbors (*k*-NN) model, also provided by Dr. Mena, to see how well they would transfer. The results from this attempt were promising. Figure 85 shows a table comparing the original label, the model's prediction on the adversarial sample, and the target the adversary was trying to perturb towards. Of the 12 different samples, 7 of the 12 were fooled from the original label; however, only 2 of those 7 were tricked towards the correct target. Overall, while not perfect at tricking the model, this is effective to show how it is possible to disrupt a *k*-NN model with minimal change to the input.

	kNN Predicted Results	Targeted Values	Actual Values
0	8	2	9
1	9	7	9
2	3	11	9
3	8	11	9
4	1	3	1
5	1	5	1
6	8	2	8
7	3	3	8
8	7	5	8
9	8	9	8
10	7	10	8
11	11	11	8

Figure 85: (EA) Table of results of the 12 adversarial samples used on the *k*-NN model

#### 2.4.4.6 Analysis of Results

Overall, the results of this research accomplished the goal of demonstrating the vulnerabilities of machine learning models through evasion attacks. It showed very strongly that neural networks, and especially image classifiers, are very vulnerable to these types of attacks. It also showed that, although not as vulnerable, a neural network model classifying the state of a nuclear reactor is also susceptible to these types of attacks, despite only have 27 features. The findings of this research show that these adversarial samples, once generated using a neural network, are transferable onto other machine learning models of different structures, such as decision trees and *k*-NN's. This also demonstrates the possibility of doing such an attack in a black-box environment, where the only knowledge is of the training data, and no information on the target model is known.

### 2.4.5 Research Conclusions

#### 2.4.5.1 Further GPWR Testing

Firstly, with regards to future research, more testing with FGSM on the GPWR dataset needs to be done. In particular, changing the parameters of the FGSM function, changing which samples the function is performed on, and testing more adversarial images are all important steps for future research. Samples created for this research span into the double digits, but more examples are needed for certain results.

A further point of research is to make more changes to the models themselves. Creating a model with higher accuracy or a betting architecture could lead to substantially better adversarial samples. The way of training the model could also greatly change the results for an evasion attack. For example, instead of training off of the original dataset, the neural network could be trained directly off of the predictions of the target model, like is done in inference attacks. Doing so could theoretically get a model that is much closer in behavior to the target, thereby making adversarial samples much more likely to transfer.

Another topic of future research is testing different types of models to perform this attack against. This research primarily focused on attacks against the same neural networks used to generate the samples in the first place. While there were brief attacks against two other model architectures: a decision tree classifier, and a *K* nearest neighbors (*k*-NN) model, they were only tested with around 12 samples total, which, while sufficient for research thus far, could be expanded upon in future research. There are also a whole host of other machine learning models that evasion attacks could be tested on.

#### **2.4.5.2 Other ways of generating adversarial samples**

One area of interest is the Expectation Over Transformation (EOT) method of creating adversarial samples. The idea behind EOT is that objects in real life undergo intrinsic transformations before being captured by devices like cameras, either in image or video. Those transformations can cause minute perturbations, like the ones created through FGSM, to be lost. This means these adversarial attacks break down in the real world. EOT attempts to solve this problem by using extra transformation functions in its optimization function instead of relying solely on the gradients of the model. Assuming the correct transformations are chosen to mimic the expected transformations, this allows for creation of an adversary that is robust to those different transformations. With that, adversarial samples that work on cameras and video feeds can be created instead of hijacking the raw input and inserting the adversarial sample from there [32]. This research seems like it would be very beneficial to investigate for future security endeavors. It is much more complex than FGSM, so there is likely a lot of work to be done to get it working. However, this research would be valuable to investigate, given just how common video surveillance is.

There are many other ways of generating adversarial samples and doing evasion attacks. Finding better methods for evasion attacks is critical for future research, along with finding defenses against those specific methods. A great starting point would be the article “How to Attack Machine Learning” for a general overview of different methods [33], the paper “Meaningful Adversarial Stickers for Face Recognition in the Physical World” for a similar method to EOT[34], and the site “Interpretable Machine Learning” that has an overview of several different methods of creating adversaries [27].

#### **2.4.5.3 Adversarial Training**

Adversarial training is the main defense against evasion attacks. This defense method creates adversarial images for the model, and then trains the model to ignore the adversarial part and get the correct answer despite the changes. However, while it can be effective to some extent, it is not a fool-proof way to prevent evasion attacks. Because there are so many ways of creating adversarial samples, even of the same initial sample, training a model to prevent each of these attacks is computationally expensive. Furthermore, if every specific case is not covered, it is possible for the model to still be vulnerable. So, as it stands now, adversarial training is not the ultimate solution to evasion attacks and is only a stop-gap solution. However, further research may improve the effectiveness of adversarial training.

#### **2.4.5.4 Detecting adversarial images**

Finally, it may be worth exploring research into training a model to detect these adversarial samples, as it could potentially be a way to prevent evasion attacks. This hypothesis stems from the following reasoning. First, having a model dedicated to detecting adversarial samples, instead of trying to train that knowledge into a model that is already making complex decisions, seems like it would have more success. Second, being alerted that there was an attempted evasion attack through finding an adversarial sample would be far better for the security of the system than the model just correctly classifying the sample (or worse - misclassifying it anyways). It is possible that there has been research done into this topic already, but no such studies were discovered during this research. Thus, it could be a promising area of future research.

## 2.5 Linear SVC and Adversarial Label Flip Attack (ALFA)

The research presented in this section was authored by Tanya Sharma (GT) with edits provided.

### 2.5.1 Original Proposition

The paper [35] investigates the adversarial label flips technique in the supervised learning setting and presents an optimization framework that helps the attacker in calculating the most optimal number of label flips that maximally degrade the performance of the classifier (maximizing the classification error).

The attacker contaminates the dataset by flipping labels. The framework models the defender's reaction and attacker's attempt simultaneously in a loss minimization problem. Based on the framework, the authors develop an algorithm to attack SVMs. The framework can be used for evaluating the robustness of a learning algorithm under noisy conditions. As per the experimental results in the paper - the suggested attack is effective on synthetic and real-world datasets; Adversarial label noise works better than random label noise.

### 2.5.2 Research Iterations

#### 2.5.2.1 Approach to subversion and findings

##### Phase I - Literature Review:

My strategy towards targeting relevant publications was to go over recent work that is generic to any industry that is trying to adopt ML models to their work. I followed a Breadth First Search (BFS) strategy to look at the different publications and found a wide variety of them that can be broadly classified into:

- Survey Papers - They explain the wide variety of attacks that are present today and possible mitigations for the same.
- Papers on a specific attacking strategy that can be extended to different ML models.
- Model specific attacks - These types of papers had attacks that were developed for a specific scenario or a particular type of Machine Learning Model.
- Publications introducing techniques to measure the robustness of a model.

##### Phase II - Hands-on Experimentation with attacks:

- The idea was to first identify an attack from the literature review.
- Go over the corresponding publication to understand the attacking strategy.
- Make the relevant environment setup (if needed) and replicate the attack.
- Based on the results obtained, experiment and play around with the attacking strategy.

#### 2.5.2.2 First attempt at implementing ALFA

##### Logic of the algorithm

Let us consider a clean dataset called  $D$  that consists of 200 data entries with class 1. The algorithm would then duplicate  $D$  and create another dataset with the labels of all 200 data entries flipped to -1. Let us call this tainted dataset,  $D'$ . The new linear SVC model is then trained on the dataset  $D + D'$ . The new model showed a significant difference in the readings from the original model. The budget or the number of the label flips is a representation of the degree of freedom that is provided to the attacker and hence there are no limitations or specific values to start our algorithm with. ALFA would require the entire data set and the budget each time before providing the most optimal data items that impact the accuracy of the model.

Reasoning behind having maximum number of allowed flips:

We are trying to build a model that is helpful in a real setting. In a normal scenario, the attacker would not have so much power to flip the entire dataset. Therefore, the maximum number of allowed flips helps us to keep a budget and keep the attacker's power in check.

#### Maximum number of flips that are allowed:

It is the length of the dataset – as per our example, the attacker could flip 200 labels. Binary classified data, having 50% of the data flipped would show the reduction in the accuracy, recall and precision.

#### Results and Observations

Ideally, the training accuracy should look like a standard normal distribution with one peak, where Y axis would be the number of flips and X axis would be the impact on the new classifier. When 50% of the data is flipped, the model should have the most impact and an increase in the number of flips after that would indicate that the amount of tainted data that the model is being trained on is more than 50% and hence the training accuracy would just start increasing after that. However, a plot between the number of flips (Y Axis) and testing accuracy (X Axis) would be a straight declining line.

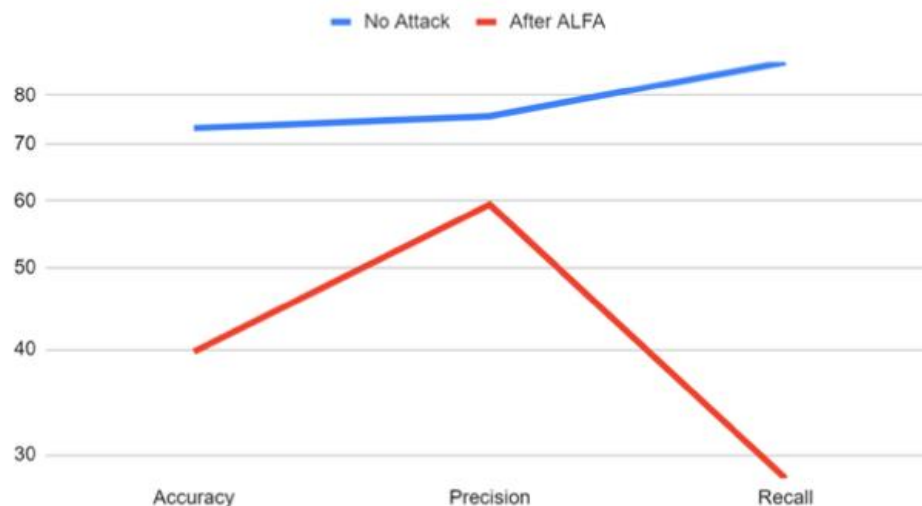


Figure 86: (ALFA) First implementation of ALFA:  
Before vs after ALFA

#### 2.5.2.3 Second attempt at implementing ALFA

I came across a Python implementation of ALFA online (which was implemented during the first attempt at ALFA), and I found a bug in it. I fixed the bug and obtained a functional alfa algorithm. The algorithm would take a budget (let us call it  $b$ ) given by the user and find the best  $b$  values from the dataset which when flipped would maximize the classification error and give the least accuracy.

Figure 86 shows us the correlation between the optimal flip count and how changing the number of flips impacts our results. As seen in graph, as we increase the number of flips in our dataset, the testing accuracy shows a continuous dip.

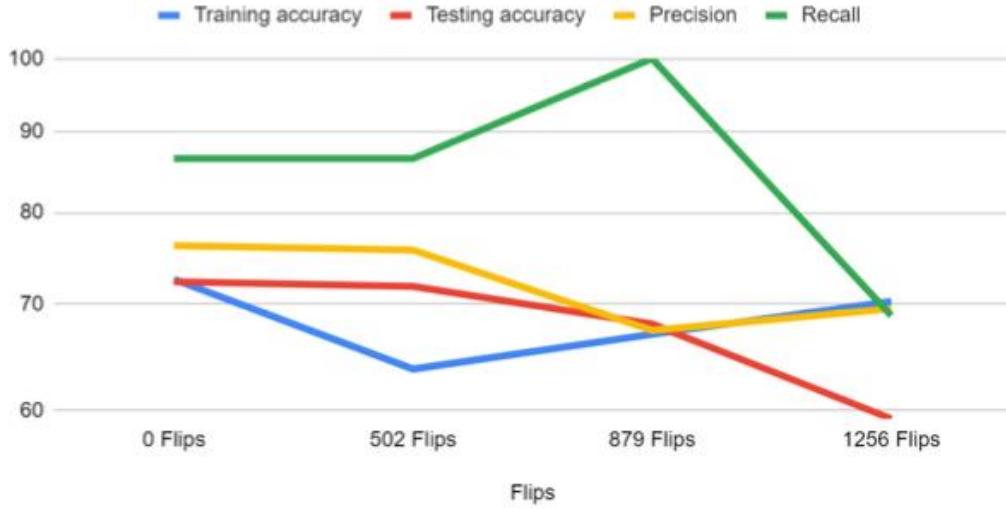


Figure 87: (ALFA) Second Implementation of ALFA: Number of Label Flips vs. Training accuracy, Testing accuracy, Precision and Recall

### Investigating the results of ALFA

It can be observed from Figure 87 above, that the number of flips and the model's training accuracy, precision and recall are not proportional in the way that one would expect. It is known that increasing the number of flips beyond 50% would only result in increasing the training accuracy of the model as the model now knows the flipped data better. This would result in three possibilities:

- There is a glitch in the optimization (bug fix) that I had added last week;
- The initial accuracy of the Linear SVC model on the original data is 72% and hence maybe this model isn't apt for our data;
- ALFA is not a good strategy to attack this model.

### Applying ALFA on a dummy dataset to verify our implementation

Table 32: (ALFA) Verification of implementation using dummy dataset.

Budget(Flip count)	Training accuracy	Testing accuracy	Precision	Recall	#Iterations
0 (Normal)	0.975	0.96875	0.94954	0.99280	1
20% = 40	0.965	0.785	0.71378	0.98081	20
35% = 70	0.95	0.585	0.57657	0.76738	20
50% = 100	0.975	0.4225	0.45746	0.58033	20

I applied the same alfa algorithm on a smaller dummy dataset with a Linear SVC model and obtained the following results. Figure 88 indicates that the algorithm works correctly - as we increase the number of flips, the training accuracy remains constant. This implies that the attack is crafted very carefully ensuring



that there is no change in the training accuracy of the model. The other model parameters, precision, recall and especially the testing accuracy show a steep decline with the rise in the number of label flips indicating that the attack is highly impactful.

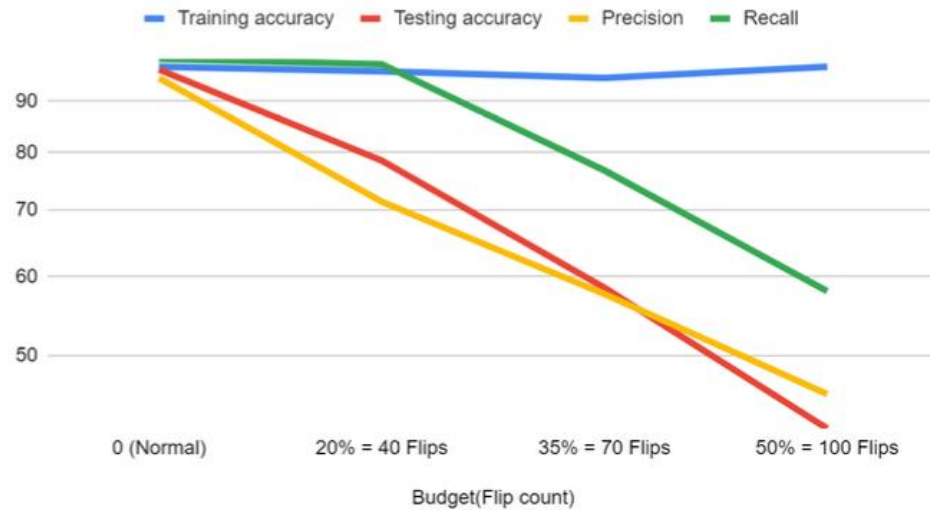


Figure 88: (ALFA) Adversarial Label Flip Strategy (ALFA) on a dummy dataset

#### 2.5.2.4 K Nearest Neighbors Model

The KNN model is a supervised machine learning algorithm that is employed in solving regression and classification problems. It is also known as a lazy algorithm since it stores all the data points and only learns during the testing phase. Let us assume there are two classes, A and B. After choosing a value for  $k$ , the algorithm calculates the distance between all the training and new data points. The KNN algorithm looks at  $k$  nearest neighbors and then classifies the new data point under consideration. The following image depicts the working principle behind KNN.



Figure 89: (ALFA) Working Principles behind k-NN



## Performance of KNN on our data

Since we are working with a binary classification problem, we decided to apply KNN to our data. The resulting model's performance was far better than the Linear SVC model we started with. The following image shows the values of different parameters: training accuracy, testing accuracy, precision, and recall that were obtained on the application of KNN on our data.

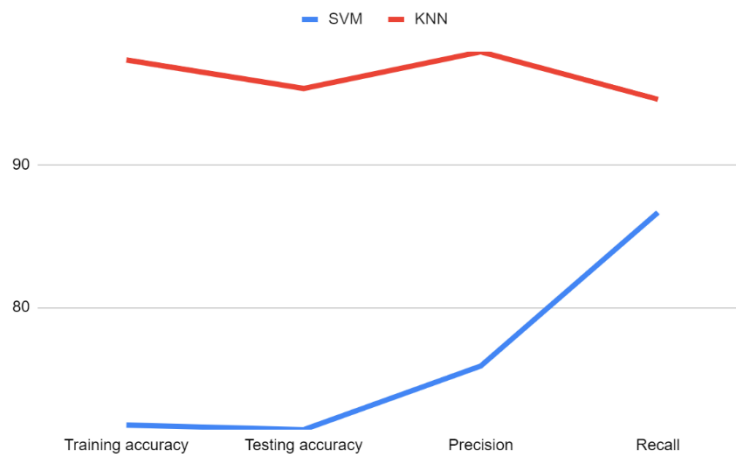


Figure 90: (ALFA) *k*-NN vs. SVM Performance (Training and Testing Accuracy, Precision, and Recall)

### 2.5.2.5 Applying ALFA on *k*-NN

We started our experiments with Linear SVC and as explained in the previous section, the performance of Linear SVC model on our data was poor when compared to KNN. The Adversarial Label Flip Attack strategy was originally developed against Support Vector Machines and after having gained an understanding of the alfa algorithm, I was extremely excited to see the results if we tried to apply alfa on kNN. Our findings were pleasantly surprising since all the values of the model parameters: training accuracy, testing accuracy, precision, recall - show a significant decline with the increase in the number of label flips in our data and hence application of alfa on KNN was impressive.

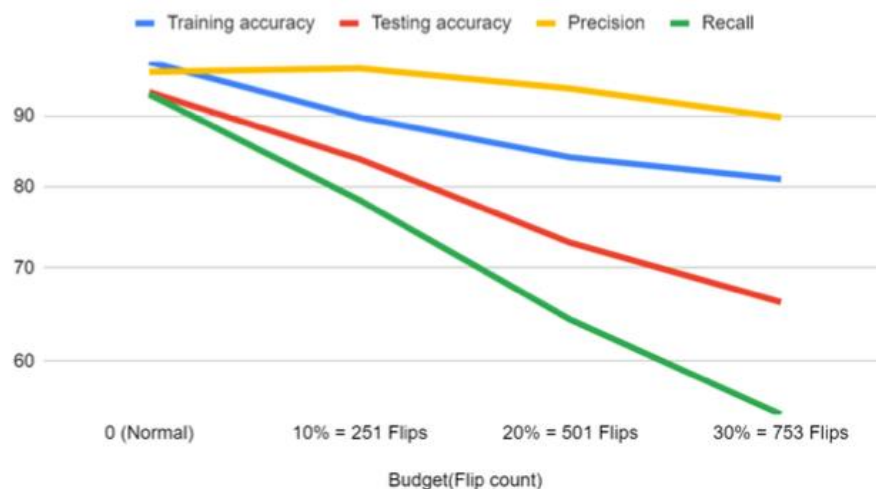


Figure 91: (ALFA) Applying ALFA on *k*-NN (Training and Testing Accuracy, Precision and Recall)

### 2.5.2.6 Gradient descent attack on k-NN

The paper investigates the robustness of k-NN classifiers and classifiers that combine k-NN with neural networks against adversarial examples. They propose a gradient based attack against k-NN and k-NN based defenses. Their proposed work outperforms the state of art on k-NN with  $k > 1$ . The attack suggested by them is a white box attack and they assume that the adversary has access to the distance metric, value of  $k$ , and the training data used in the model. The attack uses a smaller set of training data, and they are called guided samples. They show different heuristics to choose the guided samples from the training data. Their attack finds perturbation in such a manner that the distance from the data point under consideration to each guided sample is within a threshold.

This attack targets the testing data unlike ALFA where we were targeting the training data. A k-NN model is first trained with our training dataset. We then perform the attack on the testing data and record our results to check the effectiveness of the attack. The table below explains the details regarding our implementation of the attack.

Table 33: (ALFA) Gradient descent attack on k-NN Input/Output Values

Value	Description
knn_attack()	function that performs the attack
Input to knn_attack()	testing data, T
Output from knn_attack()	testing data with some entries modified, T'

K is a tunable parameter, and the attacking algorithm first finds the k nearest neighbors that belong to a class other than the class of the data point under consideration. The distance between the neighbors and the original data point is calculated. The closest neighbor is chosen. There is noise / perturbation that is then added to the original data point that is just enough to misclassify it. Figure 92 and Figure 93 can be used to visually understand what the attack tries to do.

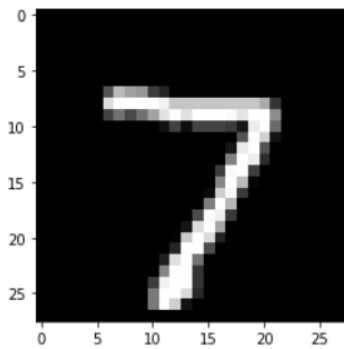


Figure 92: (ALFA) Gradient descent attack on k-NN: Noise Example i

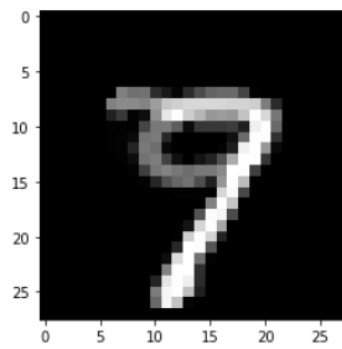


Figure 93: (ALFA) (ALFA) Gradient descent attack on k-NN: Noise Example ii

## Measuring the difference in the testing data before and after the attack

The original KNN model was run against modified testing data,  $T'$ , which was obtained after performing the attack with the help of the `knn_attack()` method described above. I was very curious to see how different the testing data was before and after the attack and hence I introduced a method called `get_difference()`. As an example, a row in  $T$  and  $T'$  would like as shown in the Table 34:

Table 34: (ALFA) Gradient Descent Attack using `get_difference()` function

Value	Example / Description
$T$	Example - <code>[[10,50, 0, 100,0.001], [...], [...]]</code>
$T'$	Example - <code>[[10.01,49.999, 0.162578, 99.9998,0.007], [...], [...]]</code>
<code>get_difference()</code>	it tells us how different $T$ is from $T'$
Input	$T$ , $T'$ , and $y_{test}$
Output	<code>Diff_dict</code> , <code>new_x_test</code> , <code>new_x_test_adv</code> , and <code>new_y_test</code> (Each of them is described below)

Each feature can be of a different scale which implies that the resulting differences would also indicate different scales. This is the strategy I used to check how much of the testing data was modified after the attack.

- Take the absolute difference between two corresponding cells in  $T$  and  $T'$
- Divide the difference by one of the values that were used to obtain the difference.
- Take the average of all the differences for the features for each row which means each row has one value that indicates the difference between that row in  $T$  and  $T'$ .
- Pick the indexes of the rows that were modified, from the original test data (we will call it `new_x_test`), modified test data (let us call it `new_x_test_adv` and `new_y_test`) and run the original model against them.
- Record your readings

## Results and Observations

Experimenting with different values of  $k$ :

Once we obtain the modified testing data, we run our model against three different versions of the testing data:

- Entire testing data that is obtained after the attack with a few rows modified.
- Partial rows of the original testing data selecting only the rows that were targeted by the attack.
- Partial rows of the testing data obtained after the attack selecting only the rows that were modified.

I have also mentioned a dictionary in the results. There is a dictionary returned by `get_difference()` that keeps track of the number of rows in the test data that were modified and are above the given threshold. The keys of the dictionary are the threshold, and the values are the number of rows that are greater than the threshold.

Table 35: (ALFA) Dictionary Keys and Row Changes beyond threshold

Keys in the Dictionary	The amount of change they signify
0	0%
1	0.01%
2	0.02%
3	0.03%
5	0.05%
10	0.1%
15	0.15%

### Gradient descent attack against KNN with k = 2

*Total #rows = 3141*      *Test: Train split = 16:100*      *Test Data Size = 503 rows*  
*Training Data Size = 2638 rows*      *Valid = 528*      *Train = 2110*  
*#Rows in x\_train = 2110 rows*  
*Dictionary - {0: 59, 1: 40, 2: 35, 3: 28, 5: 17, 10: 8, 15: 0, 20: 0, 25: 0, 30: 0, 50: 0}*



Figure 94: (ALFA) Results from Gradient Descent Attack against k-NN, k=2, Test Data (503 rows), Modified (59 rows)

### Gradient descent attack against KNN with $k = 3$

Total #rows = 3141      Test: Train split = 16:100      Test Data Size = 503 rows  
 Training Data Size = 2638 rows      Valid = 528      Train = 2110  
 #Rows in  $x_{train}$  = 2110 rows  
 Dictionary = {0: 62, 1: 35, 2: 30, 3: 25, 5: 11, 10: 1, 15: 0, 20: 0, 25: 0, 30: 0, 50: 0}

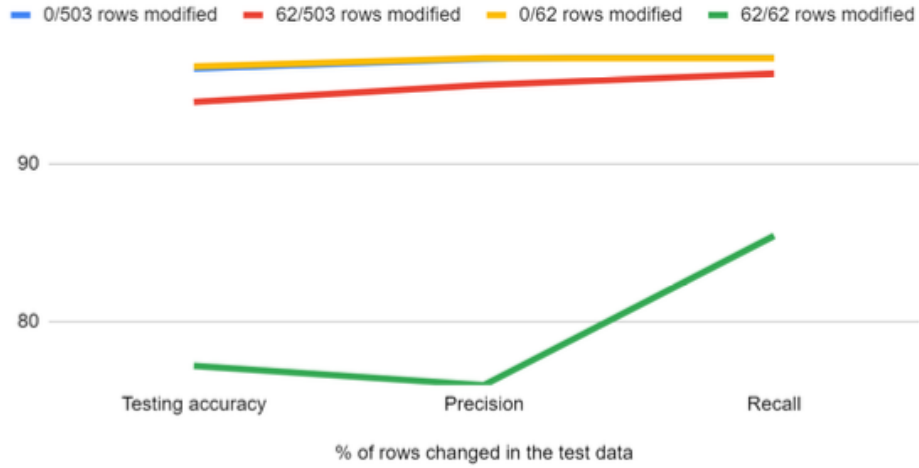


Figure 95: (ALFA) Results from Gradient Descent Attack against k-NN,  $k=3$ , Test Data (503 rows), Modified (62 rows)

### Gradient descent attack against KNN with $k = 5$

Total #rows = 3141      Test: Train split = 16:100      Test Data Size = 503 rows  
 Training Data Size = 2638 rows      Valid = 528      Train = 2110  
 #Rows in  $x_{train}$  = 2110 rows  
 Dictionary = {0: 68, 1: 44, 2: 31, 3: 26, 5: 17, 10: 5, 15: 0, 20: 0, 25: 0, 30: 0, 50: 0}

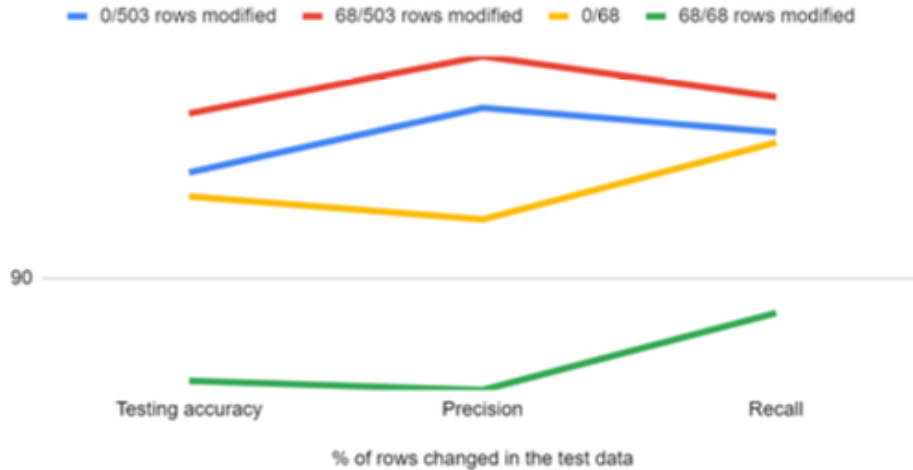


Figure 96: (ALFA) Results from Gradient Descent Attack against k-NN,  $k=5$ , Test Data (503 rows), Modified (68 rows)

After the experiment, we can conclude by saying that - a mere change of around 0.12% to the testing data brings the testing accuracy of the model down from around 94% to 50%. This clearly indicates that we were successful in ruining the model's performance without changing the model. The gradient descent attack hence proves to be remarkably effective.

### 2.5.2.7 Failed approaches to conducting ALFA Attacks

Some observations related to failed approaches to conducting ALFA Attacks.

- The journal would be completed during the fall semester and would include not just a survey and literature review of the attacks that exist today against machine learning models but also the experiments with different attacks which were performed during the summer.
- We discovered ALFA-Cr and ALFA-Tilt while researching ALFA but given the performance of Linear SVC with ALFA on our data, we decided to not proceed with this implementation
- One test we conducted was to alter the number of maximum iterations used to train the new linear SVC model to determine impacts to model accuracy. Based upon the graphical representation and statistical analysis there was no mathematical correlation.

Table 36: (ALFA) Iterations vs. Training Accuracy

Number of Iterations	Training accuracy
2500	0.34658
2500	0.35931
5000	0.66455
5000	0.31319
10000	0.31955
10000	0.30206
20000	0.31638
20000	0.28458
50000	0.31319
50000	0.36089

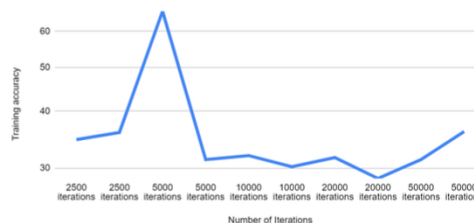


Figure 97: (ALFA) ALFA against Linear SVM: Understanding the correlation between Training Accuracy and Number of Optimizing Iterations

## 2.5.3 Research Conclusions

### Inputs to the Subversion Research Team (informing future work with our experience):

A robust method to protect any system could be to have different ML models that focus on testing various aspects of the system. Every model can then use various techniques to train on its own set of unique features and focus only on the section that is assigned to it. This prevents adversaries from identifying underlying weaknesses in a system.

### Recommendations to AR/SMR Architects, Nuclear Regulatory Authorities, and Cyber Teams:

One of the central questions that needs to be addressed is *how can this type of attack be defeated and how can we integrate these countermeasures into the design and incident response process?*

- ALFA is an attack on the training data and the Gradient Descent attack on KNN is an attack on the testing data. In order to prevent each of these attacks it is very important to verify the data that is being obtained. Whether an organization obtains the data on its own or through crowdsourcing, it is very important to verify that the data is untampered. Normal solutions that are being used today to protect data should also be used to protect the training data that is being used to train the models.
- Another helpful technique can be running validation checks on the models to ensure their correct functioning and behavior.

- Employing data poisoning strategies or breaking the models may help in detecting potential vulnerabilities.

**Where should the world of ML Hackers go? Where should this research go?**

With the advancements in Data Science, Machine learning is being incorporated in every sector of our lives in some form or the other. It is actively being used in threat detection and prevention tools today. ML Hackers can hence come up with schemes to bypass the existing checks, for example by formulating a phishing mail that bypasses the standard phishing patterns and checks. The research would then have to upgrade the existing subversion techniques to enhance the robustness of the systems and stay on par with the ever changing / evolving scenarios.

## 2.6 Attacks using the Hardware in the Loop Testbed

The research presented in this section was authored by Avery Skolnick (GT) and Patience Lamb (GT) with edits provided. Avery's work was focused on establishing the Hardware in the Loop Testbed and integration of the testbed with the GSE GPWR Reactor Simulator. Patience's work was focused on implementation of the Autonomous Control System (ACS) and Digital Twin environment using Microsoft Azure Machine Learning Workspace.

### 2.6.1 Environment Design

This section describes the design of an ACS that builds off the Digital Twin Testbed presented in Section 2.1 and includes a hardware in the loop replacing the simulated component with a physical instantiation of that component and function.

#### 2.6.1.1 Autonomous Control System Design

The ACS design was performed using ML Neural Network models (e.g., SKLearn, Keras, and Azure AutoML) within the Azure Machine Learning Workspace. The use of Azure allowed for provisioning of high-performance computing resources including CPUs and GPUs within a cloud-based environment. In addition, Azure's IoT Hub allows for real-time data transfer using Open Platform Communications (OPC) Unified Architecture (UA) and Modbus communications between a server (reactor simulator computer) and the Azure cloud.

The digital twins (DTs) composed of two separate categories of plant-level digital twins and component-level digital twins. (See Section 2.1.3.3 and Section 2.1.3.2 respectively)

One of the plant-level digital twins was produced using an auto-associative neural network (AANN), a five-layer feed-forward network consisting of two identical neural networks connected in series. Each neural network is composed of an input layer, followed by a hidden layer, and then a bottleneck layer. As seen in Figure 98, the two neural networks are mirrored to allow for all inputs to predict all outputs. In an AANN, the feature dataset and the target dataset are identical, creating an association through weighted connections.

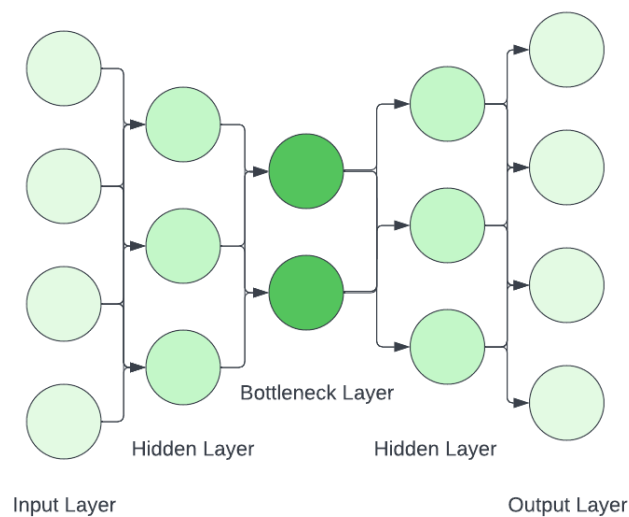


Figure 98. (HILT) General AANN design.



If there is significant variance between the training dataset root mean squared error (RMSE) and the testing (real-time data from the reactor simulator), a series of component-level digital twins are used to assess the discrepancy between normal operation and current operation.

One of the component-level digital twins is a binary classification, determining if the component is in a steady state or transient state to determine if the reactor is behaving nominally. The feature dataset is all the variables related to the specific component and the target is if the component is in steady state (1) or undergoing a transient (0). The two predominant methods of doing classification in both reactor models were a linear support vector classification (LinearSVC) model and a k-nearest neighbors (KNN) model.

To understand the difference between the two classification methods, take a set of points such as seen in Figure 99. In this example, each point represents a different parrot type. Red represents macaws, purple represent amazons, and green represents conures. The X-axis represents the food preference of each of the parrots, from -X being super sweet to +X being super spicy foods. The Y-axis represents the friendliness of the parrot, -Y being extremely aggressive to +Y being extremely friendly. If we were attempting to classify parrot types according to their food preference and friendliness, we would need machine learning to do so effectively.

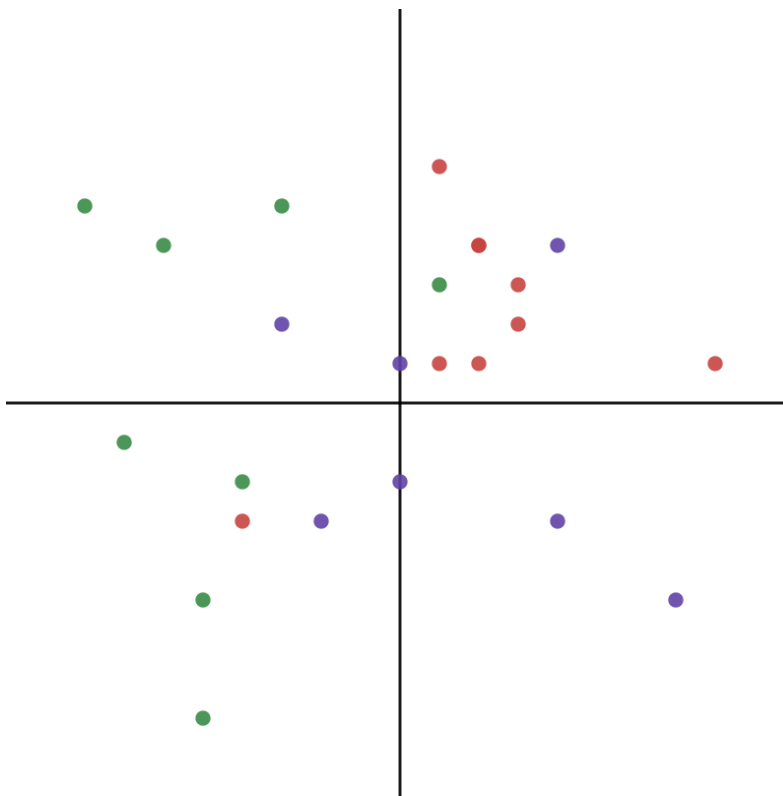


Figure 99. (HILT) Set of parrots for classification.

A LinearSVC model uses a linear kernel to minimize the squared hinge loss where the LinearSVC model uses a “one-vs-rest” (one-vs-all) strategy, fitting one classifier per class. For example, to evaluate conures, the LinearSVC will evaluate conure vs [macaw, amazon] based on food preferences and friendliness. It would create a linear inequality between conures and macaws, and then again between conures and amazons. It would then do macaws vs [conures, amazons] and amazons vs [macaws, conures] until three distinct inequalities are created, as shown in Figure 100.

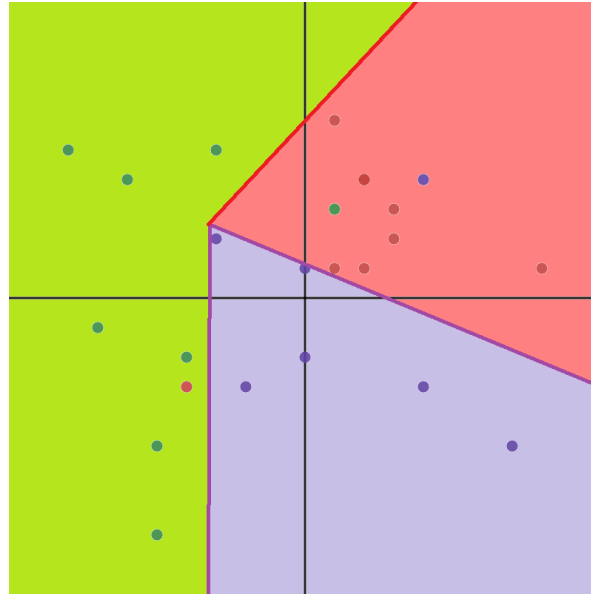


Figure 100. (HILT) LinearSVC parrot classification.

In a k-NN, the model examines the label of a chosen number of data points,  $k$ , surrounding the point that needs to be classified and then predicts the class based on the distance of the point to each the  $k$  points. For example, if we were looking at a macaw who prefers slightly spicy food and is moderately friendly (quadrant 1, highest red point), the closest neighbor would be another macaw who is not as friendly but slightly prefers spicy food too, indicating the bird is a macaw. The k-NN model would do this to every point until all points are grouped into their respective groups, as seen in Figure 101.

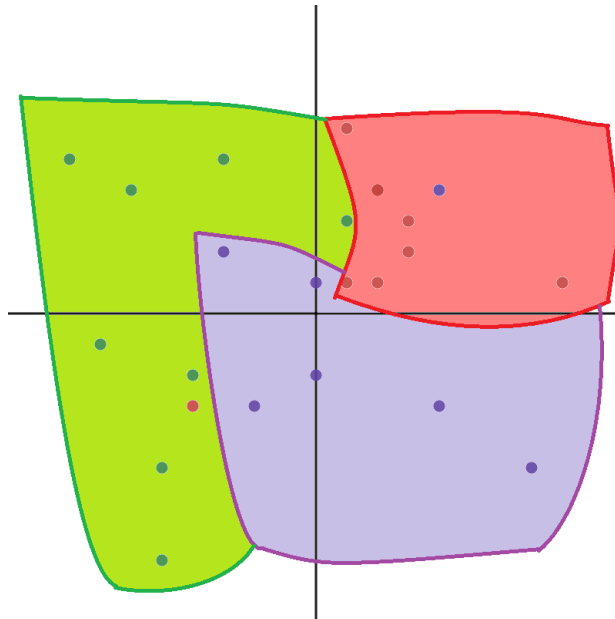
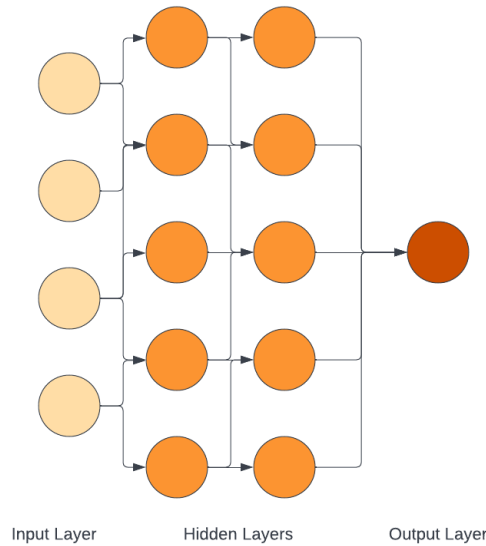


Figure 101. (HILT) KNN parrot classification.

As seen in both Figure 100 and Figure 101, models are not perfect, misclassifying a conure as a macaw and a macaw as an amazon and the LinearSVC model misclassifying an amazon as a macaw, which is why the model includes the proportion of variance ( $R^2$ ) to describe the fit of the model on the training and testing data.

The other component-level digital twin is a regression/forecasting model predicting the component's output over time to determine if the component is behaving within reasonable limits. The feature dataset is all the variables related to the specific component and the target is the component's output, which changes with respect to time. The two regression models used in the ACS implementation were the multilayer perceptron (MLP) regression model and the decision tree regression model.

An MLP regressor is a multilayer feed-forward neural network which optimizes the mean squared error (MSE) using the Limited Broyden-Fletcher-Goldfarb-Shanno (LBFGS) algorithm. The neural network works by mapping a set of features to appropriate outputs. Each layer of the neural network is fully connected to the next layer in a multilayer network. The input and output layers have linear activation functions whereas all the hidden layers have non-linear activation functions. The activation functions decide whether a neuron should be activated or not based on a function of the weights, inputs, and bias, in an MLP regressor, the activation functions aid in the speed of computation and the memory usage of the neural network. An MLP regressor in SKLearn commonly includes two hidden layers and inputs and outputs of the user specified shape. For example, the MLP regressor structure used in this project is illustrated in Figure 102.



*Figure 102. (HILT) MLP regressor model used.*

A decision tree regression uses a tree like structure, where decisions are based on binary decisions called nodes (like is the value of X greater than/equal to 7). A decision tree uses these nodes to try and split each value in the dataset and calculate the difference in the initial and the final MSE to determine if the split is effective. This process continues until the max depth or maximum number of leaf nodes (end nodes with no more splits) is reached. An example of a decision tree regression is shown in Figure 103, where the max depth was set to be 3.

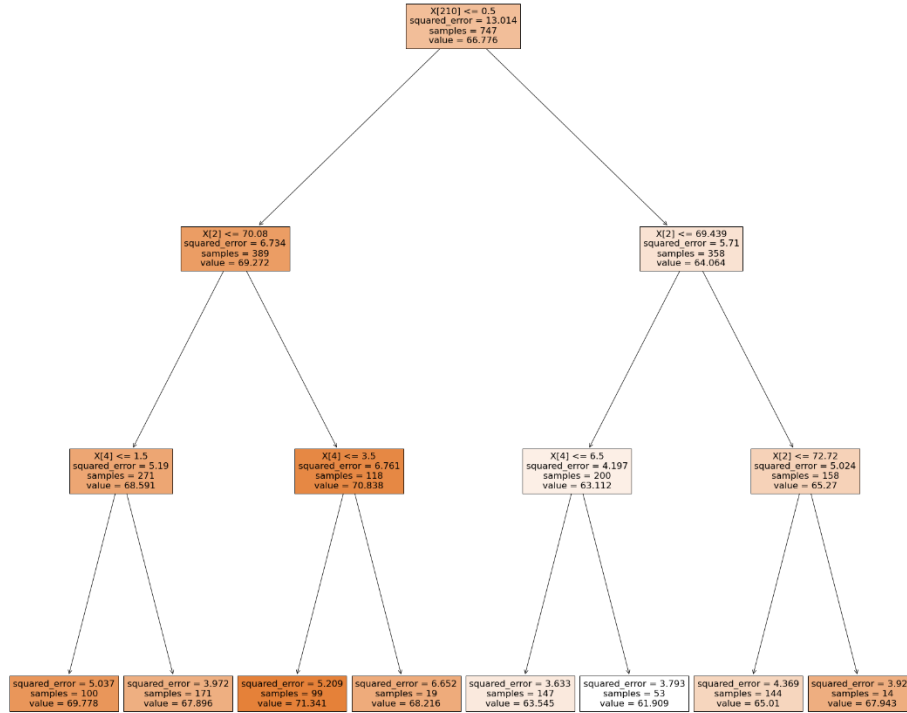


Figure 103. (HILT) Decision tree regression.

The forecasting method was done using Azure AutoML, in which a proprietary codebase determines the best fitting model for the data and then allows for predictions to be made using data from real time sources. The model takes in a CSV of data, formats it, and then the user defines the target for the model. The models can be saved for future use and predictions can be made without retraining the model.

The final model is another plant-level digital twin used for selecting a strategy based off the results from the component-level digital twins. Following identification of variance of a component from the AANN, the event is classified as a transient or non-transient using the classification model and then the forecasting/regression model predicts if the component is going to continue operating within safe limits of operation. The strategy selection method uses these results to determine the best course of action. The current architecture of this model is shown in Figure 104, where a decision tree currently predicts the course of action such as increasing/decreasing the component's output or shutting down and investigating the cause.

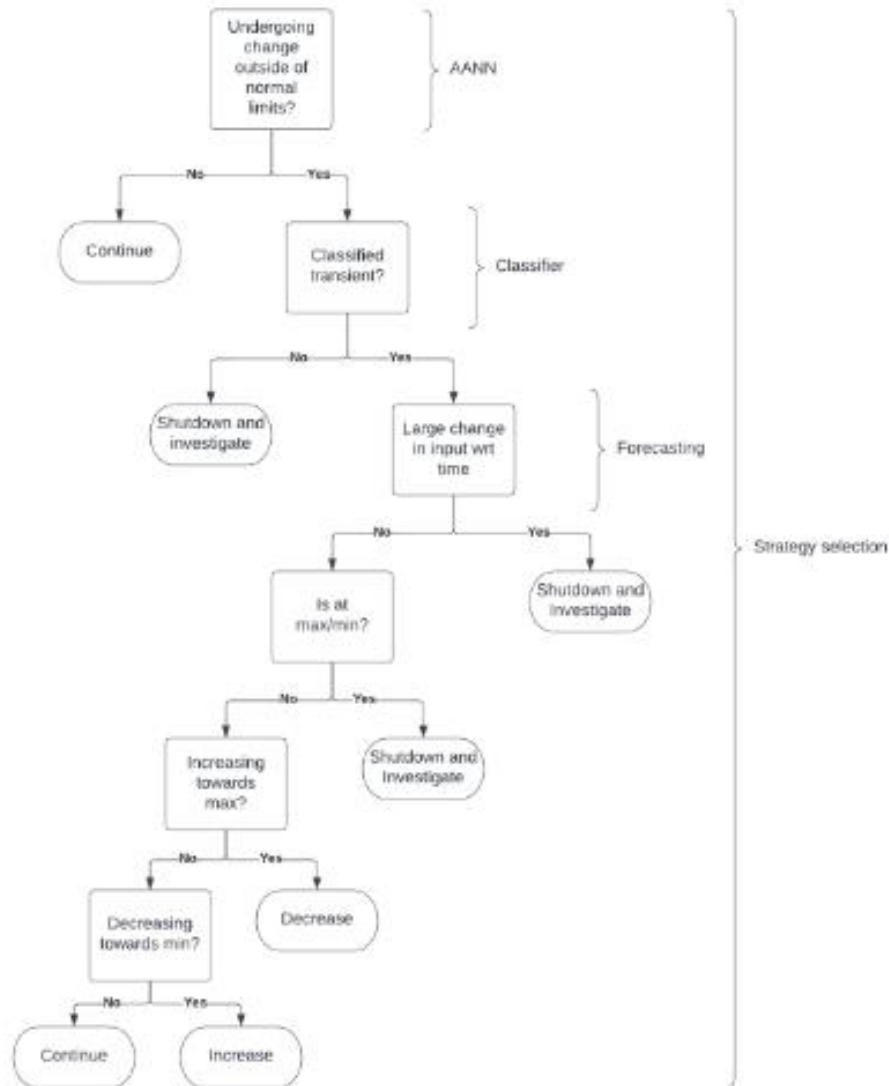


Figure 104. (HILT) Strategy selection decision tree.

As more data is obtained under operational conditions, the strategy selection tree will include more complex operational actions and compute how to achieve those actions.

## 2.6.2 Establishing the Testbed

A hardware in the loop system was created by connecting a programmable logic controller (PLC) to an open platform communications (OPC) data access (DA) server (initially RSLinx) and into a reactor simulator such as the Western Services Company (WSC) Generic Pressurized Water Reactor (GPWR) as seen in Figure 106, where the PLC is set to control the major feedwater valve (MFV) of the GPWR through implementing steam generator logic. The purpose of the system is to imitate the way nuclear power plants are controlled. In the final design, each subsystem of the plant will be controlled by PLCs taking in sensor data and data transferred from other systems and then giving commands to the necessary pumps and valves. The process of communicating between the GPWR and the PLC is documented in the RSLinx Studio 5000 configuration guide. Figure 105 shows the GPWR human machine interface (HMI) configuration used to validate the steam generator logic and PLC connection.

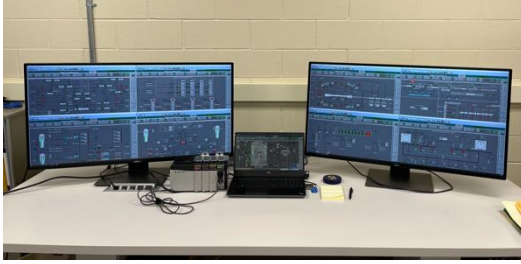


Figure 105: (HILT) GPWR Testbed HMI Picture

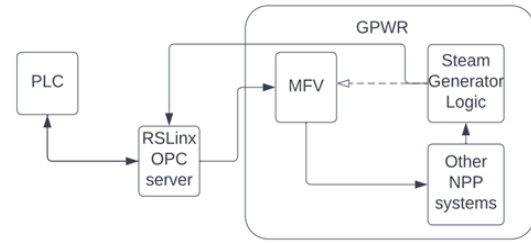


Figure 106: (HILT) GPWR Testbed Block Configuration

### 2.6.2.1 Hardware-in-the-Loop Design

In efforts to expand attack vector potentials beyond attacks to the machine learning a programmable logic controller (PLC) was introduced to create a hardware-in-the loop (HiL) design and allow for the potential of denial of service (DoS) attacks as well as man in the middle (MiTM) attacks. The PLC specifically is to emulate the steam generator.

#### 2.6.2.2 Initial Development Environment

Initially, the virtual testbed was composed of the MATLAB/Simulink Asherah reactor simulator produced by the University of San Paulo under a Coordinated Research Project (CRP) through the International Atomic Energy Agency (IAEA). The open-sourced nature as well as previous cybersecurity testbed development efforts on the Asherah project made the simulator an attractive model to pursue for an ACS. Armed with OPC UA and Modbus, the model was assumed to be a simple plug-and-play connection straight into cloud-based services like Microsoft Azure. As seen in Figure 107, the ACS was planned to be done using digital twins (DTs) on separate virtual machines (VMs) which communicate between to produce a recommendation based on real time data passed from Asherah.

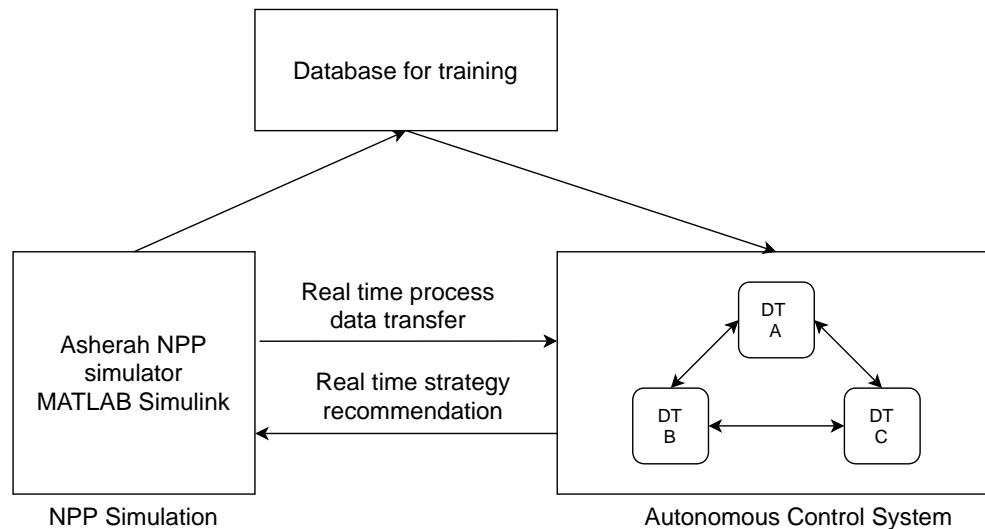


Figure 107. (HILT) Asherah ACS environment.

Each DT would have contained a segment of the ACS such as the plant-level DT, a component-level DT, or the strategy selection DT. The database was a GitLab repository to load training data into the ACS where the real-time data transfer and strategy selection would be done through direct transfer over OPC UA or Modbus communications.

Training data was initially collected offline and put onto GitLab by collecting Asherah's 95 variables loaded onto a CSV. For training the AANN, a sinusoidal power manipulation was used to capture both an increasing and decreasing power in one dataset. The RMSE of each variable varied significantly, as seen in Figure 108. For example, for some variables such as reactor power the model was a good fit, however for others such as the steam generator 2 outlet temperature, the model fit poorly with a training RMSE of 0.016463 and a testing RMSE of 0.85705 when the training and testing data was pulled from the same dataset.

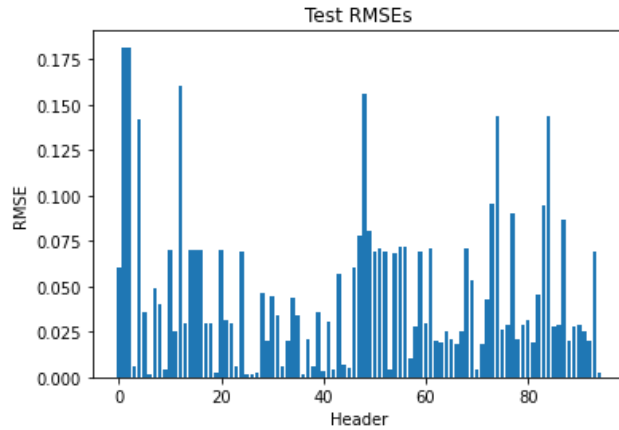


Figure 108. (HILT) Test RMSEs of each of the Asherah variables.

Due to time constraints, only component-level DTs of the steam generator were developed. To collect data for normal operation, Asherah was run at different power levels by reducing power in 5% increments every 1000 seconds from 110% to 40%. Power was also fluctuated using a sinusoidal wave as the power from 100% to 70%. Steam generator tube faults including degradation or plugged tubes was collected by reducing the steam generator tube flow area by around 2% to simulate a plugging scenario.

The best performing classification model for Asherah was the LinearSVC model which used 14 variables related to the steam generator 2 as features and predicted if the steam generator was undergoing a fault or not. The training and testing accuracy were questionable, with a  $R^2$  value of 1.0 for each. The confusion matrix for the steam generator 2 LinearSVC is seen in Figure 109, where it implies none of the events were misclassified.

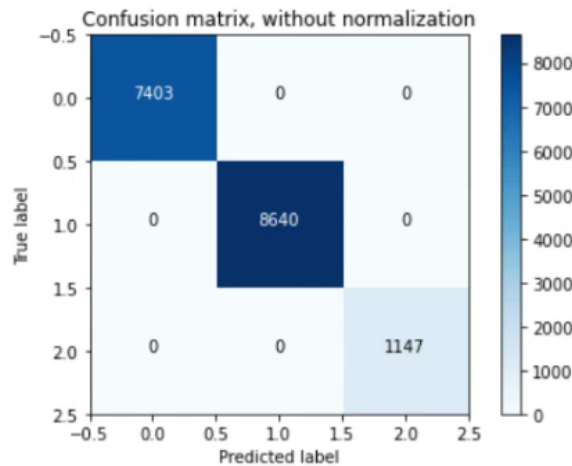


Figure 109. (HILT) Steam generator LinearSVC confusion matrix.

By using the 14 variables related to steam generator 2 as features, and a target of steam generator 2 output temperature, the best performing classification model was the MLP regression method, producing an RMSE of 0.013615 for training and 0.013876 for testing with steady state data. With transient data the model had an RMSE of 0.011725 for training and 0.011048 for testing. As seen in Figure 110, the MLP regression method predicted the steam generator 2 outlet temperature accurately.

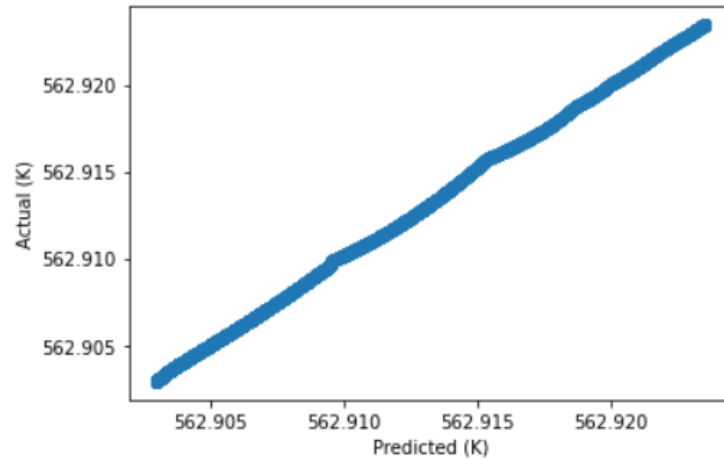


Figure 110. (HILT) Predicted steam generator 2 outlet temperature versus actual outlet temperature.

Given the MLP regression method weights time similarly to other features, the error of the steam generator 2 outlet temperature varied greatly with time as seen in Figure 111.

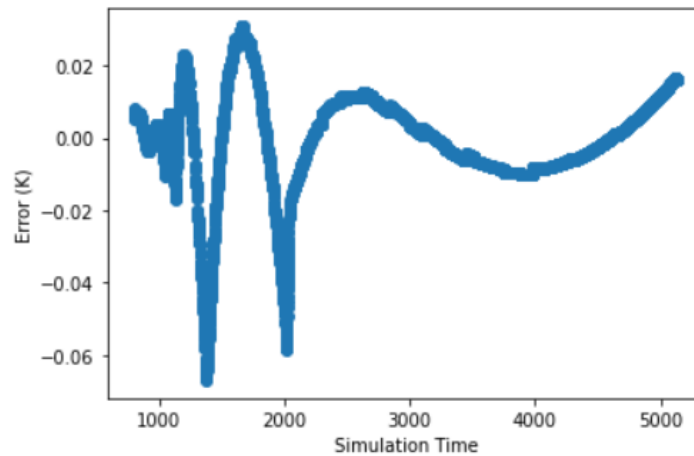


Figure 111. (HILT) Steam generator 2 outlet temperature error over time.

To proceed with the development of the Asherah strategy selection model and integrate with Azure, Asherah needed an additional library to communicate with Azure. Sandia's ManiPIO repository was chosen for the task, however multiple roadblocks were encountered that discouraged further development using the Asherah simulator due to the inability to connect with Azure or HiL systems in a straight-forward manner.

The ultimate decision to switch environments came down to the low fidelity of Asherah compared to the GPWR as well as the instability of MATLAB/Simulink when running Asherah. The reactor simulator was switched to the GPWR testbed and models reconstructed.



### 2.6.2.3 Final Development Environment

The final chosen model was a reactor simulator Generation III (Gen III) Generic Pressurized Water Reactor (GPWR) simulator connected to an Allen Bradley PLC through RSLinx Classic OPC DA communications. The simplistic OPC Data Access (DA) based architecture design allowed for seamless integration between Allen Bradley PLCs and the Western Services Company (WSC) GPWR communications.

The trouble occurred within integrating the model into Azure's Internet of Things (IoT) Hub. OPC DA is a communication protocol was developed in the 1990's based on Microsoft's Component Object Model/ Distributed Component Object Model (COM/DCOM), reducing the implementation potential beyond Windows or Object Linking and Embedding (OLE) systems. Azure IoT Hub is based on the Portable Operating System Interface (POSIX) standards more commonly found in Linux operating systems. OPC DA does not conform to the POSIX standards. To compensate for the POSIX standards used in the Azure IoT Hub, a converter was needed to allow for more modern communications such as OPC UA. FactoryTalk Linx Gateway was chosen for its ability to convert OPC DA to OPC UA communications and bi-directionally communicate with cloud-based services like Microsoft Azure. The testbed environment is shown in Figure 112, where the data is published to FactoryTalk Linx Gateway acting as an OPC Server, ingested into the Linux VM acting as the IoT Edge device where it is sent through Microsoft's Connection-less Lightweight Directory Access Protocol (CLDAP) into the IoT Hub to be processed as real time data through the Machine Learning Studio Web Services and stored in a SQL database.

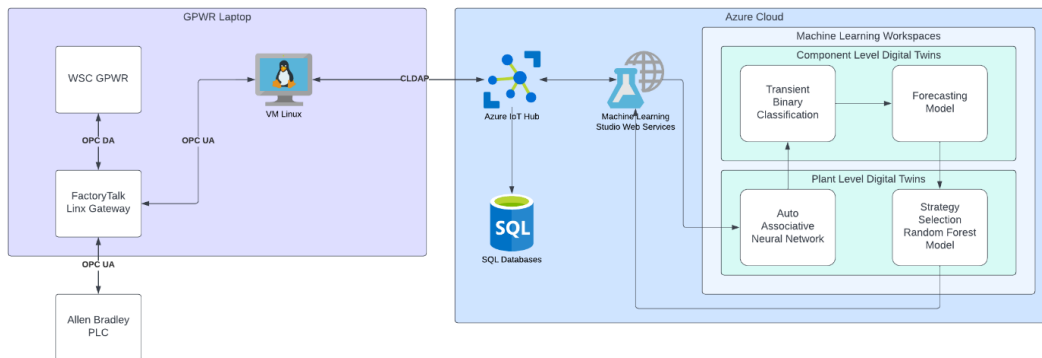


Figure 112. (HILT) GPWR testbed environment.

The AANN architecture for the GPWR was similar to Asherah’s AANN architecture where 72 variables from the GPWR were fed into the model from steady state data. The architecture is seen in Figure 113, where the smallest layer is eight neurons wide. The training MSE was recorded to be 0.0251 with a validation MSE of 0.02659 and a testing MSE of 0.02663, indicating a highly accurate system.

As with Asherah, only the steam generator was modeled using component-level DTs, and so seven variables relating to steam generator 1 were used to construct the component-level DTs. The six variables used for constructing component-level DTs were steam generator 1 narrow and wide ring levels, bypass valve, base feedwater, feedwater flow total, total flow, and outlet temperature. Both steady state and transient datasets were used.

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 71)	5112
dense_10 (Dense)	(None, 16)	1152
dense_11 (Dense)	(None, 8)	136
dense_12 (Dense)	(None, 16)	144
dense_13 (Dense)	(None, 71)	1207
Total params: 7,751		
Trainable params: 7,751		
Non-trainable params: 0		

Figure 113: (HILT) GPWR AANN architecture

The best performing classification model for the steam generator was the KNN model with a 0.42% chance of falsely identifying a steady state as a transient state and a 4.39% chance of falsely identifying a transient state as a steady state, as shown in Figure 114. The training  $R^2$  was 0.9769 and the testing  $R^2$  was 0.9518.

Using the same datasets as the classification models, the steam generator 1 narrow and wide ring levels, bypass valve, base feedwater, feedwater flow total, and total flow were used as features and the target was steam generator 1 outlet temperature.

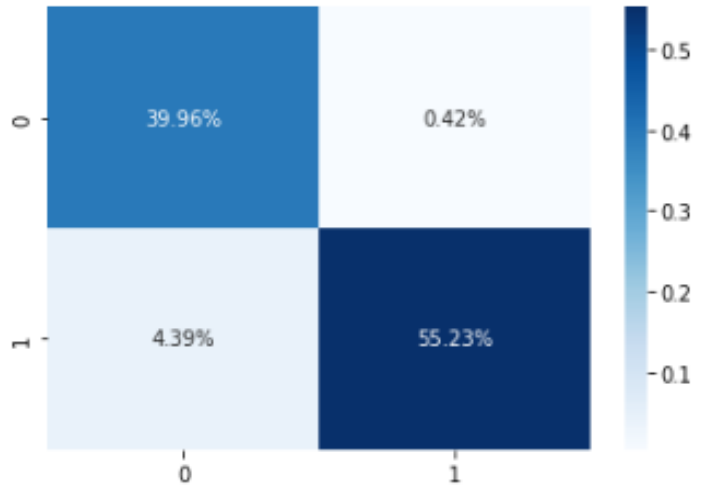


Figure 114: (HILT) Steam generator 1 k-NN confusion matrix

The best performing regression models were made through Azure AutoML with a Standard Scaler Decision Tree model being the best for steady state operation with a training and testing accuracy of 0.99 and a Standard Scaler Random Forest with a training and testing accuracy of 0.996 for transient operation.

Given the data being a time series, a forecasting model seemed more appropriate to capture the outlet temperature with respect to time and the other six factors. Azure's AutoML modeling was used for producing forecasting models to ensure the ability to feed in real time data into the models. For the Transient Forecasting model, the Exponential Smoothing algorithm had a normalized RMSE of 0.0 while the Steady State Forecasting model was best created through a Naive Bayes with a normalized RMSE of 0.0.

The initial strategy selection model focused on the output of the steam generator to figure out if the steam generator temperature needed to increase, decrease, continue, or be shutdown. A random forest classifier was used, and the training accuracy score was found to be 0.9956 while the testing accuracy score was found to be 0.9876. The confusion matrix for the random forest classifier used can be seen in Figure 115, where 0 represents continue operation, 1 represents decrease temperature, and 2 represents increase temperature. The dataset used for the AANN was used for this model.



Figure 115: (HILT) Strategy selection random forest classifier confusion matrix

#### 2.6.2.4 Challenges in Configuring the Testbed

The primary challenge in configuring the Testbed was establishing the connectivity between components and choosing the communication protocols that would be used. Additional challenges were encountered including:

- **Simulator Variable Conflicts**

The simulator was calculating values for being replaced; and these variables were overriding the inserted variables. This was solved by deleting the assigned file that was writing to that variable in the internal control logic of the simulator. Figure 116 shows the SG logic HMI, and the dotted line represents the severed connection

- **Identifying Plant Operations Variables**

The documentation was not entirely clear on the variable maps, so variables were recorded into a file with data that included: the variable name (ex: hmi\_RCSTT443B\_VALUE), a brief 1-2 sentence description (for users not familiar with power plant components), and the source HMI. This is documented in the Georgia Tech Component Naming Directory that provides best practices for mapping variable names to trends, supporting the recording and export of data to formats including CSV.

- **Designing the Power Ramp Transients**

The next challenge was to design the necessary power ramp transients as the simulator only contained one. A custom file was created to override the default simulator files which allowed us to code an alternate power ramp in the scenarios tab.

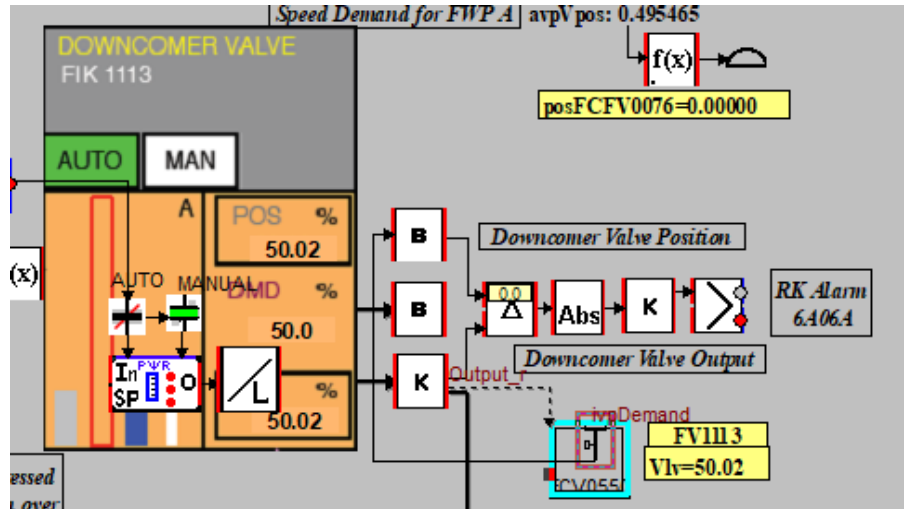


Figure 116: (HILT) GWPR Simulator HMI Steam Generator Logic Severed Connection

#### 2.6.3 Simulating Cyber Attacks

The research focus this summer was on supply chain attacks in support of subverting machine learning implementations in the operational environment. These attacks occur when someone in the supply chain of a device installs malware or malware is installed as part of a system update. Once the system is penetrated the hacker has nearly total control of the given PLC. For instance, the attacker could proceed with a false data injection attack (FDIA) in which the PLC sensor values are altered leading to incorrect commands to actuators.

The first attack produced was a FDIA that targeted the feed water control valve (FCV) by reducing the value to 80% of the suggested value. This attack was created by reading the PID blocks output information and then multiplying it by .8 and then writing it to the valve controller. Figure 117 shows the FCV under normal conditions in a 100to75to100 power ramp and Figure 118 shows the FCV under attack conditions with the same 100 to 75 to 100 power ramp. Interestingly enough, the pump controller usually compensated for the closed FCV valve.

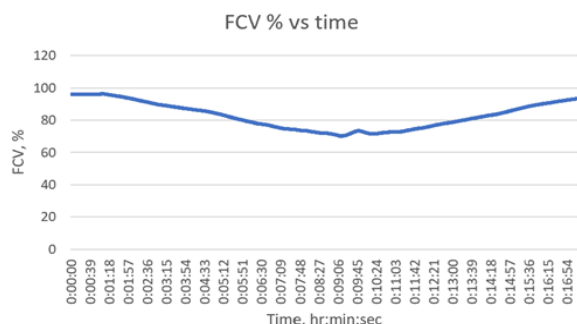


Figure 117: (HILT) FCV under normal conditions (100-75-100 Power Ramp)

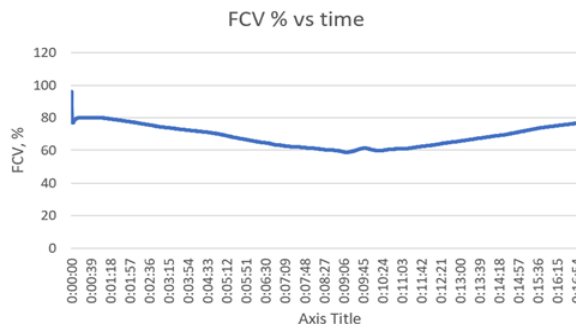


Figure 118: (HILT) FCV under attack (100-75-100 Power Ramp)

The next attack run was holding the valve to a set position by just using a move function to set the FCV to 75%. Something noteworthy, the attack caused the reactor to trip when the power ramp function went below 70. Figure 119 illustrates the SG level with a 100to75to100 power ramp.

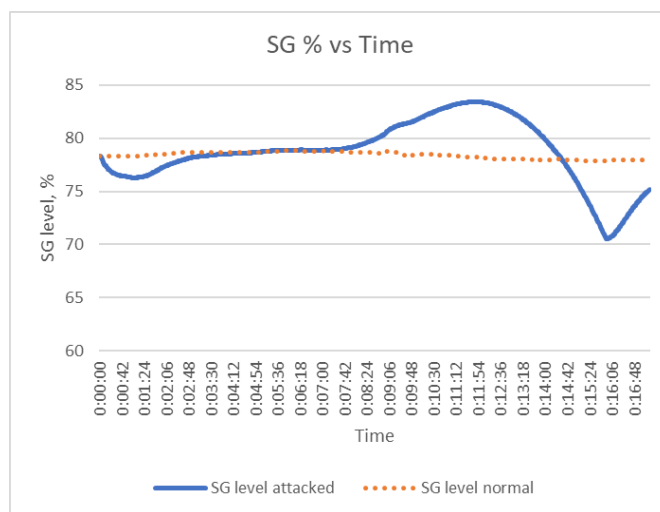
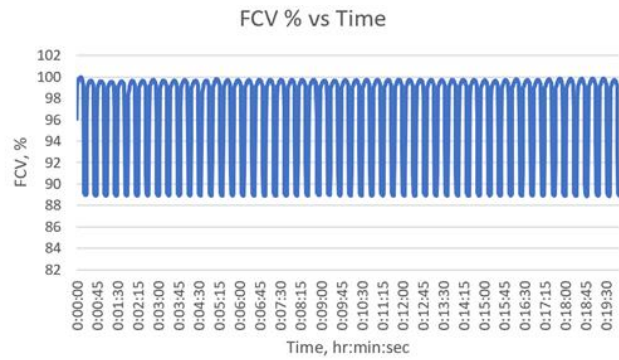


Figure 119: (HILT) Steam Generator with a (100-75-100 Power Ramp)

An additional more complex FDIA was implemented. In this attack, the average FCV position at each power level was calculated,  $P_{avg}$ . Then if the intended command was larger than  $P_{avg}$  the value was multiplied by 1.05. If the command was less than  $P_{avg}$  it was multiplied by .95. The attack resulted in an oscillating value along the mean as displayed in Figure 120.

Furthermore, a DoS was conducted against the PLC. The attack was implemented by connecting a computer running Kali Linux, a computer running the GPWR and PLC via a switch. The computer running Kali Linux sent ~500,000 data packets per second to the PLC. The attack resulted in the OPC server crashing and the PLC displaying a data storm notification. However, the crashing of the OPC server resulted in an inoperable GPWR with all data services blocked. This is an open issue.



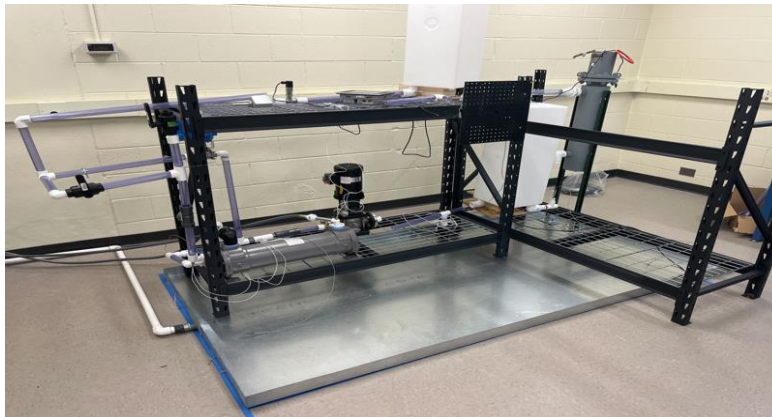
*Figure 120: (HILT) Complex FDIA Attack with Oscillating Observable*

## 2.6.4 Future Attacks

With the HiL testbed and ACS implemented work is being conducted on subversion of real-time data communication, machine learning, and cloud/database services. The next set of subversion goals include building out the following capabilities:

- Implementing additional component-level DTs of target components.
- Connecting the AANN to the component-level DTs and the strategy selection model.
- Implementing a “smarter” strategy selection model including producing a probability-based modeling scheme based off operational data and operator decisions.
- Testing out various subversion tactics against the ACS such as the plant-level DTs and component-level DTs.
- Conducting attacks against the HiL system including between the PLC and FactoryTalk Linx as well as between the Linux VM and the IoT Hub.
- Testing out various subversion tactics against the SQL database containing training data.

In conjunction with the Temple Team, research has also been conducted on ways to use the HiL system to simulate Ransomware Attacks. Georgia Institute of Technology has access to the Logic Locker Malware family, which is a cross vendor ransomware worm that targets PLCs. It is capable of hijacking and locking out PLCs by taking advantage of weak authentication found in common PLC models.



*Figure 121: (HILT) Georgia Tech Testbed Structure*

### 3. Threat Hunting for ML Subversion

Implementing somewhat effective cyber-defense capabilities starts (hopefully) during the system design phase, where security requirements are placed upon the target systems, functions, and processes such that they will be resistant to known classes of cyber-attacks and provide observables to allow for identification of future attack class observables. These security controls are implemented and validated prior to operating the system such that the system owner and operator can execute their incident response capability upon observation of events (anomalies) that may indicate that a cyber-attack is or has taken place. One commonly quoted statistic is of adversary-on-target time where a threat actor exists within a system or set of systems for a given number of days (weeks/years) prior to delivering an effect that is detected and actioned upon by the victim organization. While most organizations operate using a reactive posture where only when an observation has occurred do they engage their response process, other organizations have been building out what are generally known as active cyber defense methods which include interaction with systems and system objects and often threat actors operating within these systems. One such active cyber defense method is Threat Hunting, where a team of one to many threat hunters derive a set of hypothesis about how a threat actor may deliver a cyber effect into their environment and then create a hunt flow that allows for interrogation of system and system objects to determine whether the expected observables exist in the places that we expect to find them or if the hypothesis is nullified.

As we performed our work on subversion of machine learning algorithms this summer, we not only thought about how the algorithm would be subverted but also about what it would look like for a threat actor to carry out this type of attack. In order to conduct a trojan or inference attack, privileged access will be required to the model training or model operations environment, and they type of actions this threat actor will take will have, we believe, a set of unique signatures compared to other attack classes. Since attacks on ML are generally focused on the model selection and implementation and training and testing data, the people and systems involved in those processes are ideal targets as part of a cyber campaign to conduct these types of attacks. In Section 4 we offer up eight (8) scenarios for how ML subversion attacks could be carried out and with each scenario we provide a set of Cyber Attack Lifecycle (phase) mappings along with associated MITRE ATT&CK Tactics, Techniques, and Procedures (TTPs) and an example of Hunt Flow rules that a Threat Hunter could use to search out system objects and events that match to expected ML subversion attack patterns. We should note, although it should be obvious, that the examples we provide are limited and that in an operational environment we would have a broader more comprehensive set of hunt flows to execute against our development, testing, and operational environments.

#### 3.1 Kestrel

Kestrel is an open-source programming language for threat hunting. It follows the scientific process of observation, hypothesis, experimentation, and verification to allow threat hunters a mechanism for performing cyber reasoning and threat discovery. The tools were originally developed by IBM Research and IBM Security based upon their participation in DARPA's Transparent Computing program that is focused on adversarial engagements. Kestrel offers threat hunters a domain-specific language where one did not exist before. [36]

We chose to use Kestrel based upon the existing community of Kestrel users and contributors and availability of the open-source documentation. We did not instantiate the Kestrel agents on our testbeds although we believe it would be straight-forward to implement a technical prototype with an ideal target being the Georgia Tech Digital Twin testbed as it is comprised of a diverse set of interconnected systems, networks, operating systems, and applications such that executed hunt flows would be somewhat realistic.



### 3.1.1 Hunt Flows

The Kestrel Hunt Flow is the control flow of the hunt. Each hunt step in the flow reflects a set of entities, where each step is comprised of Kestrel commands. Kestrel commands are broken into five categories: Retrieval, Transformation, Enrichment, Inspection, and Flow Control.

#### 3.1.1.1 Kestrel Commands

##### Retrieval Commands

*These commands are used to retrieve information from systems within your environment. You may be interested in activity on a Data Historian or security events from a Camera or an aggregated view of observations from a boundary protection device such as a Firewall. These retrieval commands allow the threat hunter to retrieve the information:*

**GET:** match a Structure Threat Intelligence Exchange (STIX) pattern against a pool of entities and returns an entity type. Figure 122 is an example of a STIX pattern that shows the relationship between a URL and a backdoor Malware object. Perhaps we have identified a Threat Actor who has packaged an ML Trojan Attack payload within a package pulled from a URL after a dropper is executed. We could use a GET command as part of our Hunt Flow to query our NGINX proxy for this URL as part of our Threat Hunt.

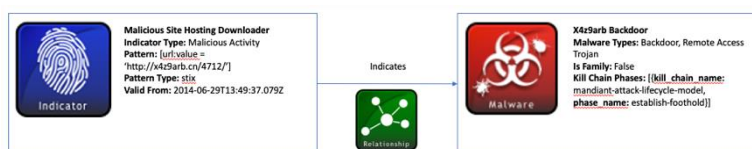


Figure 122: (THunt) STIX Relationship Example

**FIND:** return entities connected to a given list of entities. Following from our **GET** command, should we have any matches to our STIX pattern we would likely be interested in the associated entities that were on the receiving end of URL file transfer (if it was successful) and we would be interested in the IP address of the URL endpoint (to compare it to existing threat intelligence to determine whether the threat actor distribution infrastructure has implemented their existing delivery mechanisms or have altered the delivery mechanism for some additional effect). In our Kestrel Hunt we would connect these commands together return a list of entities based upon the defined relationships (attributes).

##### Transformation Commands

*These commands allow us to reshape our data objects (entities) to provide views that may help us find or connect entities together based upon our hypothesis. Perhaps as part of your Threat Hunt you have retrieved information about file system access for all ML team members. You may then have two threat hunt flows, one where entities are grouped by team function (Model Developer, Training Data Curator, Validation and Test Engineer, Operations and Deployment Engineer) and then one where entities are grouped by functional system dependency (Steam Generator Models, Condenser Models, Reactor Control System Models, ...). These transformation commands will allow the Threat Hunter to create these data views:*

**SORT:** reorder entities based upon entity attributes. Sorting is applicable at multi-layers and may often be preceded by *Enrichment Commands* such that the source entity has additional attributes to order by. We may be interested in sorting by Geo-Location for external IP addresses or by First Name of person entities since our brains are more comfortable with alphabetized lists. We will often want to sort after we have applied grouping to our entities.

**GROUP:** group entities based on one or multiple attributes. Assuming we have collected a set of entities from our operational environment, we may want to group them by System Administrator or by Security Levels and Zones. This is the fun part of Threat Hunting, looking at the set of available attributes and then choosing grouping characteristics that will allow us to confirm or nullify our dependent hypothesis throughout the hunt. *I believe that a threat actor who is conducting an inference attack will increase the average number of disk reads per day over a period of time on target compared to what is normally observed.* Entities in this case could be pulled, grouped, and then compared in order to validate or nullify this dependent hypothesis.

### Enrichment Commands

*This command allows us to pull in additional data attributes to the entities we retrieved using the **GET** and **FIND** commands such that we have additional options for **SORT**ing and **GROUP**ing our entity data. Threat Hunters will have available to them structured data sources where key fields are used to connect entity sets. One of the examples we provided above was grouping systems by security level or zone or system function. Generally, when we pull entity data from systems these additional attributes are not available unless we are pulling from a SIEM or a data source that allows data source joins via the **GET** and **FIND** commands. So, the **APPLY** command can be implemented through an advanced **GET** or **FIND** or via the Kestrel **APPLY**.*

**APPLY:** compute and add attributes to variables. Perhaps one of our dependent hypotheses is that *ML Inference Attacks are more likely to be executed by women whose first names begin with K and who live in the State of Idaho.* When we **FIND** the set of users for a given set of queries against our internal ML library (web-based) we will apply Human Resource (HR) data to the username field enriching the entity list with attributes such as **first\_name**, **last\_name**, **address\_1**, **state**, **zip**, ... allowing us to use our Flow Control and Inspection commands to validate or nullify that hypothesis.

### Inspection Commands

*These commands allow us to view the **INFO**rmation about Kestrel Variables, data associated to system-level objects that can be queried when executing Hunt Steps, and to **DIS**Play attributes of entities associated to the Kestrel Variables. It is important to think about Kestrel Threat Hunting or Threat Hunting in general as an interactive exercise in hypothesis exploration where as Hunt Steps are executed, the Threat Hunter will pause and inspect the information about the variables and resulting entities, update their variables based upon whether they retrieved what they were looking for, and execute the updated Hunt Step, repeating this process based upon each hypothesis and hypothesis variant.*

**INFO:** show details of a Kestrel Variable. Examples of Kestrel Variables include operating system processes, network traffic objects, user-accounts, email-addresses, windows-registry-keys, and files. Each variable has a set of associated attributes that can be viewed using the **DIS**Play command. One other concept to raise that is covered in the Kestrel Documentation is the concept of edges or enumerated relationships between entities. Kestrel Hunt Steps often include directional queries such as in Figure 123 where a Kestrel Variable is defined via a **FIND** command and a relationship **RELX**:



Figure 123: (THunt) Kestrel Variable FIND RELX



**DISP:** print attributes of entities in a kestrel variable. This command enables us during our Threat Hunt to inspect variable attributes, allowing us to identify additional pivots and to use for Flow Control commands such as **MERGE** and **JOIN** and to assist us in hypothesis validation or nullification. Threat Hunters will utilize this command for state inspection and then will generally have a set of statistical tools set aside to use for entity analysis outside of Kestrel.

### Flow Control Commands

*These commands control the flow of the hunt, how the Threat Hunter progresses between Hunt Steps. They enable the Threat Hunter to **SAVE** and **LOAD** their Hunts and pull saved entity sets from data sources such as files and databases. These commands also include the ability to **JOIN** variables based upon common entities and **MERGE** entities into super-sets although the Threat Hunter needs to be careful about entity attributes.*

**NEW:** create entities directly from a data source. Often, we will want to create entity sets from data sources that are either not accessible via Kestrel Hunt commands **GET** and **FIND** or ones that have been saved from previous hunts.

**LOAD:** load data from a disk into Kestrel variables. This command is like loading a Jupyter Notebook or a notebook in Google Colaboratory where a series of commands are loaded into the test environment for execution.

**SAVE:** dump Kestrel Variables to a local file. This command is used to save your Hunt Steps and execution results to a file. This file can be subsequently pulled back into the Threat Hunting environment using the **LOAD** command.

**MERGE:** union entities in multiple variables. This command combines entities in multiple variables to be used as part of a new Hunt Step. Looking at Figure 125 there is a Hunt Step (in purple) created from two Kestrel Variables (threat hypothesis A and the first Hunt Step of threat hypothesis B) through the **MERGE** command. Two additional Hunt Steps are executed to produce a display object.

**JOIN:** takes two variables with a common entity reflecting what attributes the variables share. This command is used by the Threat Hunter to join two variables using aligned attributes. Examples for all commands are provided in the documentation [36] but we include this one in Figure 124 to help the reader to better understand this command in context of the other Kestrel commands used.

```
procsA = GET process FROM stixshifter://edrA WHERE [process:name = 'bash']
procsB = GET process WHERE [process:binary_ref.name = 'sudo']

# get only processes from procsA that have a child process in procsB
procsC = JOIN procsA, procsB BY pid, parent_ref.pid

# an alternative way of doing it without knowing the reference attribute
procsD = FIND process CREATED procsB
procsE = GET process FROM procsD WHERE [process:pid = procsA.pid]
```

Figure 124: (THunt) Example of Kestrel JOIN Command

#### 3.1.1.2 Kestrel Variables, Hunt Steps, and Hunt Flows

Kestrel Variables are straight-forward to understand. Think of any object you can query from a system, at whatever level of granularity you wish to query at, and that is your Kestrel Variable. If you read through the Kestrel documentation [36] examples look for the variable names after Kestrel commands such as **FIND**

and **GET**. For example: *GET process*; *GET file*; *FIND network-traffic*; *FIND ipv4-addr*. The Kestrel Variables in these examples are operating system *processes*; *network traffic* captured as PCAP or NetFlow data; *files* extracted from file systems or network streams; and *ipv4-addr* a subset of *network-traffic* which will pull the *IPv4 Address* associated to a set of systems or networks, or other entity set. As can be seen in Figure 126 Hunt Steps take Kestrel variables and Hunt Step Metadata as input and produce Kestrel variables and Display Objects as output.

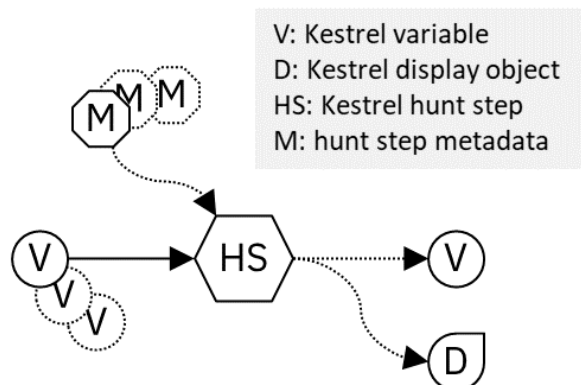


Figure 126: (KHF) Kestrel Hunt Step Object Relationships

Each **Hunt Step** utilizes the notions provided in Figure 126 to include Variables {V}, Hunt Step Metadata {M}, and Display Objects {D}. An example Kestrel Hypothesis Hunt Flow is provided in Figure 125 that includes two primary threat hypotheses: *threat hypothesis A* and *threat hypothesis B* along with a variant *threat hypothesis A, variant A.2*. This is a good example of how Kestrel Threat Hunts are segmented into multiple Hunt Steps which are then **JOINED** and **MERGED** in order to validate or nullify the given hypothesis.

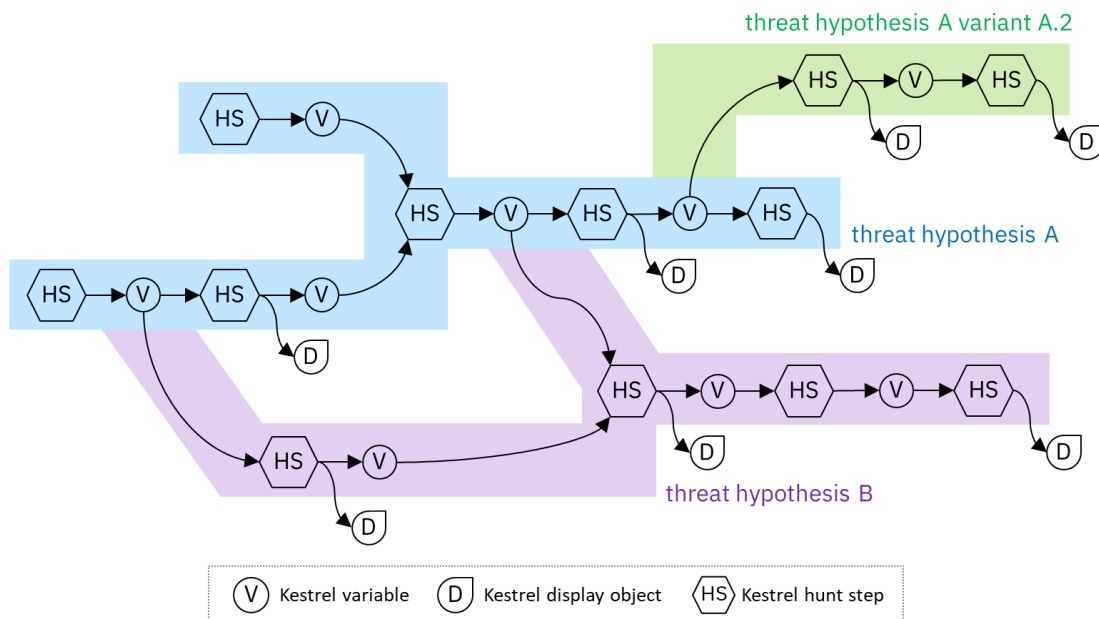


Figure 125: (KHF) Kestrel Hypothesis Hunt Flow Example

### 3.1.2 Applying Hunt Flows in Scenarios

In Section 4 we provide eight (8) ML Subversion Scenarios. For each scenario we include Hunt Steps that could be used by the Threat Hunt team to search out actors who may be conducting these types of attacks against the ML environment at all stages of the ML lifecycle (design, test, deploy, operate). In the next two sections we will provide an overview of the Office of the Director of National Intelligence (ODNI) Cyber Threat Framework which allows for these actions to be mapped to Threat Actor Campaign Lifecycle Phases and MITRE ATT&CK which allows for threat actor actions to be mapped to Tactics, Techniques, and Procedures (TTPs).

## 3.2 ODNI Cyber Threat Framework

The Cyber Threat Framework originates from the Office of the Director of National Intelligence (ODNI). This framework was created to have a common taxonomy for cyber incidents that could be utilized across US government organizations and with international cyber partners. The structure of the framework reflects a hierarchical, structured, transparent, and repeatable approach. Creating a simple and structured approach allows the framework to act as a viable universal translator for cyber threats. [37]

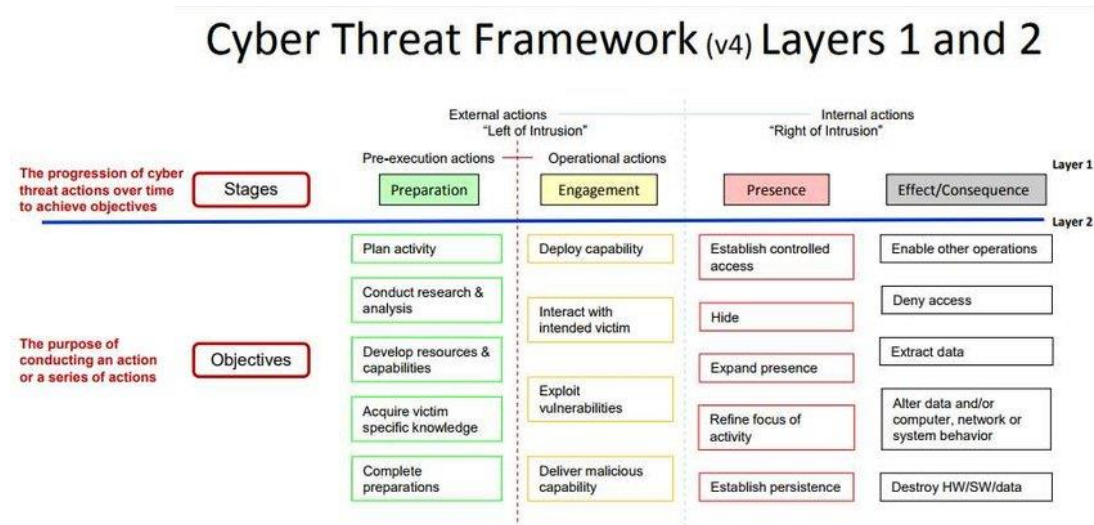


Figure 127: (KHF) ODNI Cyber Threat Framework v4

There are four main stages to the framework: Preparation, Engagement, Presence, Effect/Consequence. Each stage can be identified by a set of adversary objectives. Each of these actions and indicators are mappable to MITRE ATT&CK TTPs. For those not familiar with these concepts or mappings, there is both a cyclical nature to how these actions are applied in target environments and There are actions and indicators that may be mapped to TTPs in each of the stages.

### Preparation Stage

Preparation is primarily focused on the planning of a cyber activity. In this beginning stages of the process there may be thorough research and analysis of the mission goal. In the scenarios, preparation takes the form of learning everything you may need to know about the target with the exploitation of victim specific knowledge. The information gathered in this stage becomes the objective of the preparation paving way to engagement. An example of this would be doing research on Nuclear Power Plant Coolant System to know what software versions and type the plant is using. The defense team can look at activity on vendor sites as well as external email traffic during the threat hunt to detect preparation tactics.

### Engagement Stage

Engagement is where an adversary is deploying a capability. In the scenarios given, engagement may take the form of simple interactions with the intended victim or the exploitation of vulnerabilities in the target system. This stage is comparable to the ATT&CK framework in the next section dealing with initial access, and how an adversary will interact with a victim to reach the desired target or establish presence on the network or system. An example of engagement could be social engineering or sending a phishing email to gain a way into the desired target. The defense team can utilize insider threat data and external email traffic, checking all permalinks they may contain.

## **Presence Stage**

Presence on a victim's network is where an adversary can establish their malicious intent over a set period. The main objectives for an adversary to establish presence is to gain controlled access, expand their presence, refine their focus or area of attack, and establish persistence. In the scenarios, Presence may be establishing a trigger point on the target system that may be delayed for days/weeks at a time. Establishing persistence may be setting up a key-logger with connections to an adversary C2 where the adversary tracks all the movements of an operator for use in the future. The threat hunting team can investigate network traffic to see if any outward-bound traffic is returning to a single server point, or if there are persistence measures that can be detected in alarm logs.

## **Effect/Consequence**

The effect or consequence to the victim are the effective outcomes an adversary intended throughout the lifecycle of the attack. The objectives can be characterized by enabling other operations for future cyber-attacks, denying access to legitimate users, extracting data for the target system, and even altering data or network behavior. In worst case scenarios, the adversary may even destroy all data in the system leaving it crippled for future operation. The threat hunting team can check existing data and data historians, alarm logs, and improper use of credentials to find the effects in the target system.

## **3.3 MITRE ATT&CK**

As we example the actions of cyber attackers and defenders, we are benefited not only from associating each event or action to a lifecycle stage using the ODNI Cyber Threat Framework but also to associate each event or action to a Tactic, Technique, or Procedure. MITRE has been evolving their ATT&CK taxonomy over time to include three overlapping matrices for understanding environmental TTPs. These matrices include Enterprise ATT&CK that covers traditional ICT systems and networks, Mobile ATT&CK that covers mobile device and mobile networks, and ICS ATT&CK that is focused on Industrial Control Systems.

Our primary focus is on the delivery of cyber effects via ML subversion to Nuclear Energy and Nuclear Research environments, and thus the Impact Tactic (TA0105) and its associated Techniques are of interest to us. An ML Subversion Attack would likely implement one or many techniques associated to this tactic such as: T0815 Denial of View; T0827 Loss of Control; and T0880 Loss of Safety. In these cases, the techniques used to accomplish this cyber effect delivery would be those that enable a cyber threat actor to implement and execute the ML Subversion attack. This may be a chained ML Subversion attack where the first set of actions include performing an Inference Attack against an operational Model which leads to designing a Trojan or Evasion Attack to deliver the effect that results in a Denial of View, Loss of Control, and ultimately a Loss of Safety at the NPP.

Throughout our scenarios we call out some of the associated TTPs as examples. We should note that our focus in these examples is not on completeness but on simply assisting the reader to understand the connection between the application of the ML Subversion Attack and both what it would look like across the Cyber Attack Lifecycle and how a Threat Hunter would search out system and system objects for evidence that this type of attack has or is currently taking place.

## 4. Scenarios

### 4.1 Foreign Adversary Scenario

#### Scenario Overview

In this scenario there are two countries: East Dakotamy and West Dakotamy, located geographically adjacent to each other in the northern hemisphere not too far from the EU Bloc. East Dakotamy has recently revealed a new reactor that threatens to turn the world of nuclear energy on its head. Wanting to keep up but knowing East Dakotamy would not just hand over their secrets, West Dakotamy decides to use espionage to get the information they need. A team of the brightest minds in West Dakotamy are assembled to perform a model stealing attack, in which a machine learning model is reverse engineered from training data and a basic knowledge of the original model.

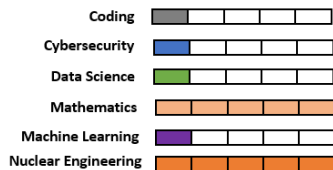
#### Character Profiles

**Amy** (*true neutral*)

**Title:** West Dakotamy Nuclear Engineering SME (Lead)



**Amy**  
**Nuclear Engineering Expert**

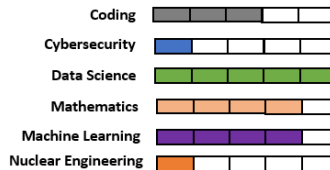


**Alex** (*true neutral*)

**Title:** Recent PhD graduate in Data Science



**Alex**  
**Data Scientist**

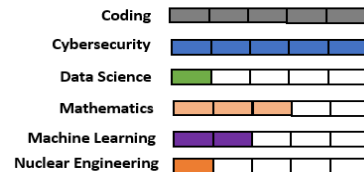


**George** (*chaotic neutral*)

**Title:** Government Hacker (Lead)



**George**  
**Government Hacker**



#### Equipment and access

The hacking team have been provided a working space at a government facility. Each person gets a windows PC, which runs a virtual machine whose traffic is routed through Tor. Command and control addresses and servers have been provided. They have high speed internet and have been given money to use in their exploits but are highly encouraged to use only publicly available tools. They currently have access to programs like nmap, Shodan, and recon-ng.

#### Background

In the 1900's Dakotamy was a world leader in nuclear energy having two of the premier education institutions in the world for nuclear engineering and the first 4-reactor Nuclear Power Plant to ever have been built. As the turn of the 21<sup>st</sup> century the two primary political factions within Dakotamy arrived at the point of irreconcilable differences and decided to peacefully split the country in half, with East Dakotamy formed on the land mass east of the Boltoph River and West Dakotamy on the roughly equal land mass

west of the Boltoph River. The Dakotamy Nuclear Power Plant ended up being located in East Dakotamy with an agreement by the NPP management that the NPP would continue to be staffed by citizens from both East and West Dakotamy. This situation was workable although West Dakotamy always yearned for their own NPP to ensure energy independence for future generations of West Dakotamians. However, where West Dakotamy did have an advantage was in cybersecurity. In the past five years the cybersecurity operations of West Dakotamy had advanced quickly until they were one of the countries at the forefront of the industry.

Given that West Dakotamy is so advanced in cybersecurity, they thought it would be a good idea to have a team of government hackers on hand. This team of hackers was established with three of the country's best minds leading it. Amy is West Dakotamy's leading nuclear engineering expert. She had recently retired after a long 50-year career but found retirement to be boring so eagerly accepted the offer for the position. Alex recently finished their PhD in data science, publishing their most recent groundbreaking research in a very prestigious scientific journal. George is the youngest of the group. He graduated from the nation's top university with bachelor's degrees in computer science and cybersecurity. Following that, he became a white hat hacker for the government and has since risen to be the nation's top cybersecurity expert at just the age of 35.

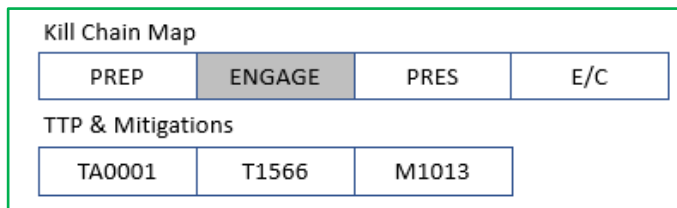
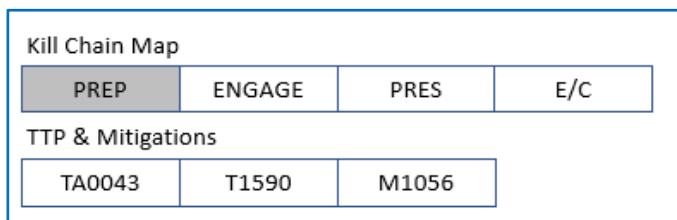
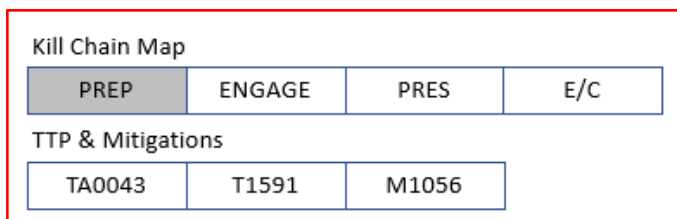
### The Event

The one obstacle that stood in the way of West Dakotamy catching up to their eastern neighbor was that the most knowledgeable scientists were located in East Dakotamy and recently they unveiled a new design for a remotely operated autonomous advanced reactor. The government of West Dakotamy knew that the scientists and government of East Dakotamy would never hand over the technology they had developed, and they would ensure the licensing costs were so high as to make the acquisition of the East Dakotamy designed NPP unattainable. The government of West Dakotamy decided that industrial espionage would be the best pathway to obtaining the information they needed and thus called up their team of hackers to penetrate and exfiltrate the reactor design information necessary to use for their new reactor build.

### The Attack

■ ■ The first phase of the attack involved planning exactly what they were looking for and where they could find it. Thanks to Amy's many years of nuclear engineering experience, she was able to hypothesize what systems may be using machine learning algorithms and identified where they could be found in a typical plant.

■ Having recently been inspired by an article about the Robin Sage experiment, George and Amy worked together to identify potential phishing targets on LinkedIn who might have access to the aforementioned systems. They picked out a couple of nuclear engineers, security engineers, and a corporate executive. Posing as HR, they send out targeted





emails regarding employee health insurance. For the executive they create a specialized email. They send him a phony business report claiming to be from the manager of an important financial department. The document contains malware that is designed to give the attackers a backdoor into the system. The phishing emails sent to the executive and one of the nuclear engineers were successful, and the team gained access to both the corporate and reactor system.

■ ■ The executive's computer has a clean directory structure with folder names that reliably described their contents. There was a folder for events that they had attended and a folder that contained the company financials, encrypted XLS files. Many of the files that were password protected were accessible by applying a MS Office password cracking tool to the file revealing their contents. George used this program to obtain every password in 23 minutes. Here they were able to access several reports from the research team that explained that the model that ran the reactor was a neural network and how they had trained it. It did not give the training data though. The training data was required for reverse engineering the model, so they examined the reactor control system fileshare. They found a file labeled training data that was not password protected. The information on the model and training data were passed to Alex for reverse engineering.

■ Alex was able to perform a grey box attack to reverse engineer the model implemented at the West Dakotamy lab. They imported the data into Google Collab, normalized it, and then used AutoKeras to generate classifiers, feeding in the training data. After validating the model, they had created a near perfect replica of the original model.

■ George and Alex decided to take the attack a step further by making the East Dakotamy model less effective. They went back into the reactor system and added noise to the data, causing the ML classifier to misidentify several key pieces of input. The noise was almost undetectable, so the reactor operators could not understand why there were multiple incidents in such a short period of time.

### The Discovery

An operator named Suzan with a background in data science discovered that the data had been tampered with. She took what she has found to the plant's SOC team who started an investigation. After searching through the company's systems, they found the two backdoors. Talking to the operator and the executive they realized that these backdoors must have come from phishing emails, and an analysis of their email history confirmed this theory. They suspected that West Dakotamy was involved, as the attack was highly sophisticated, making it past the spam filters and antivirus software the plant had in place. It was clear that whoever conducted the attack had privileged access but could not confirm West Dakotamy involvement.

Kill Chain Map			
PREP	ENGAGE	PRES	E/C
TTP & Mitigations			
TA0006	T1110	M1036	

Kill Chain Map			
PREP	ENGAGE	PRES	E/C
TTP & Mitigations			
TA0007	T1083	N/A	

Kill Chain Map			
PREP	ENGAGE	PRES	E/C
TTP & Mitigations			
TA0040	T1565	M1029	

Kill Chain Map			
PREP	ENGAGE	PRES	E/C
TTP & Mitigations			
TA0040	T1529	M1053	

That is, until West Dakotamy unveiled their own autonomous reactor just a few short months after the incident. Intelligence collected by East Dakotamy spies confirmed that they did not have the technological capability to have developed the technology on their own. With the theft confirmed, it sparks a huge fight between the two nations. It makes international headlines as West Dakotamy vehemently denies their involvement and accuses East Dakotamy of trying to make them look bad. The two countries are now on the brink of war as this newest incident shattered the peace between them.

#### Kestrel Threat Hunting Artifacts

```
# Hypothesis 1 (H1): Hackers targeting the companies through a phishing campaign can be analyzed
# by checking external email traffic as well as any hyperlinks before any users automatically
# click. Backdoors may be revealed through analysis of outbound network traffic and alarm logs
# reflecting improper user access.
```

```
# [04.1.1.1] get External email senders from SMTP server data
_External_Users = GET (email-addr|userId)
FROM SMTP_external_traffic_logs
WHERE (email-addr|userId) in ('External_traffic')
```

```
# [04.1.1.2] get Alarm events from Alarm logs for Time period TP1-TP2
_Alarm_Logs = GET (events)
FROM reactor_coolant_pump_logs
WHERE START (event_time) t'TP1' STOP (event_time) t'TP2'
```

```
# [04.1.1.3] get outbound network traffic from company IT network
_Outbound_IP_01 = FIND ipv4-addr CREATED BY any _IPs_not_Assigned
_Outbound_Traffic_01 = FIND network-traffic CREATED BY andy _IPs_not_Assigned
```

```
# [04.1.1.4] get Login Events from Engineering Portal
_Engineering_Portal_Logins = GET (username|userId)
FROM engineering_portal_logs
WHERE userRole in ('RCP_user')
```

```
# [04.1.1.5] get Network Scan Detection events for Control System Network &PLC Testbed_Ips
_CSN_01_Assigned = FIND ipv4-addr CREATED _CSN_01
_IPs_PLCT_01_Assigned = FIND ipv4-addr CREATED _PLCT_01
_Traffic_CSN_01 = FIND network-traffic CREATED BY any _IPs_CSN_01_Assigned
_Traffic_CSN_02 = FIND network-traffic CREATED BY any _IPs_PLCT_01_Assigned
```

```
APPLY docker://scan_detection ON _Traffic_CSN_01
APPLY docker://scan_detection ON _Traffic_CSN_02
```

```
# [04.1.1.6] get Event logs of PLCs on OT Networks for Time period TP1-TP2
_PLCT_Events = Get(events)
FROM PLC_event_logs
WHERE START (event_time) t'TP1' STOP (event_time) t'TP2'
```

```
# Hypothesis 2 (H2): Grey-box attacks will induce increases in read-write input-output statistics
# across those systems associated to the classifier implementation.
```

```
# [04.1.2.1] get entity list of ML Operational Systems
_ML_Systems_Group = GET (systemId)
FROM system_asset_db
WHERE system_tag in ('ml_ops')
```

```
# [04.1.2.2] get filesystem read-write input-output statistics
_FS_RW_IO = GET (rw-io)
FROM _ML_Systems_Group
```



## 4.2 Trojan Scenario

**Scenario Overview:** In this scenario, a freelance hacker is tricked by a group of malicious hackers into performing a Trojan Attack on a Nuclear Power Plant. The adversary hopes to create a trojan that will cause the plant to shut down in some capacity, resulting in a loss of revenue.

### Character Profile



**Malissa**  
**Hacker for Hire**



### Equipment and Access

Because she likes the mobility and ability to live in the shell on the Mac OS X system, Malissa owns a MacBook Air 13''. She hopes to save up enough money to eventually purchase the new M2 Processor for it, but unfortunately does not currently have the funds. Her internet access comes from wireless connectivity via a high-speed fibre optic link. In her toolbelt she has access to Google Collab and pyTorch.

### Background

Malissa, a teen fresh out of high school, was in the top 15% of her 2021 graduating class. A member of the entrepreneurship club she was always pitching new business ideas and had plans to open a business that delivered gourmet cupcakes to local homes and businesses. What better way to maximize profits than to deliver those cupcakes while walking dogs for area families! She believed this idea was good enough to grow into a national franchise and to accomplish this goal, Malissa knew a significant amount of capital would be required. In her spare time, Malissa has been doing some freelance computer science work to help pay for her new startup. Although baking and dog walking is her passion, Malissa has also always been good with computers, she is even what some people would call a prodigy.

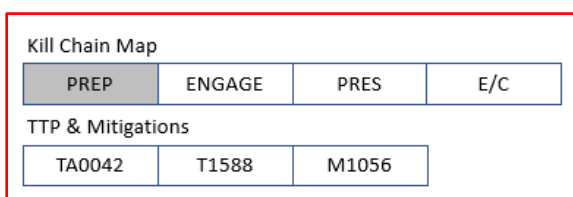
Given her entrepreneurial spirit, Malissa decided to attend a local college last year and completed CS101: Introduction to Computer Science, CS171: Introduction to Data Science, and CS141: Introduction to Computer Security. CS101 covered all the usual topics in an introductory class including programming language constructs and introduction to algorithms, with weekly course assignments that had to be completed using Python. Malissa had a good amount of Python experience from High School and through the CS101 course she was able to build those skills while learning new ones in her Data Science course. Of course, the Cyber Security course was mind blowing as the instructor was also a white-hat hacker and had many excellent real-life experiences to share with the class.

## The Event

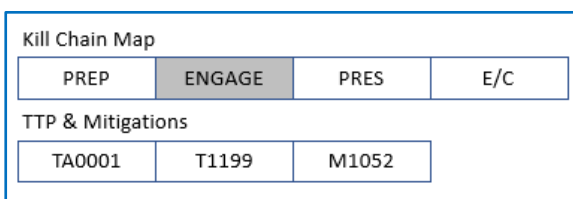
One day she is approached by a new client who claims to work with a group of hackers and want Malissa's help on their upcoming scheme. The client tells Malissa that a group of terrorists have managed to subvert personnel at a Nuclear Power Plant such that they are able to obtain privileged access to the environment. They have asked Malissa to help them create an attack payload based upon the Machine Learning algorithms implemented at the NPP. This group of malicious hackers is known to target industrial control systems. Malissa does not question their story at all though. She wants to be a white-hat hacker like her CS141 professor and figured the paycheck wouldn't hurt either. She believes this is an opportunity to gain some experience and build her portfolio which will assist her in launching her new business. They tell her the goal is to make the engineers shut down the plant.

## The Attack

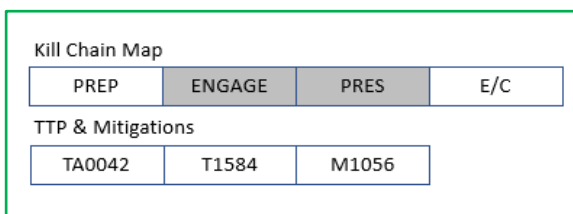
■ The hacker group gives Malissa a Machine Learning model that the plant uses as well as some poisoned data. In Google Colab Malissa uses pyTorch to train a Trojan model that she believes will produce a reactor transient such that the reactor is taken offline based upon the standard operating procedures. This effect delivery should not cause any actual damage to the plant infrastructure, only create an event that will cause the reactor to go offline and the energy generation to halt.



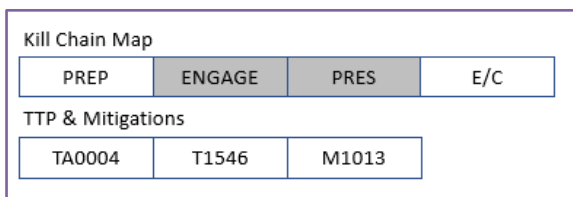
■ The trojan is designed to influence the Machine Learning classifiers that determines whether a reading is normal or abnormal and alert the operators if need be. It is retrained on a poisoned data set so that the algorithm recognizes normal readings as abnormal ones when presented with a trigger, as per the design of Trojans that she learned about her in Computer Science courses.



■ Malissa packages the Trojan she developed and delivers it to an insider with access to the target environment. This insider swaps the original model for the retrained trojan one, bypassing the model integrity functions implemented by the NPP security team. Once the Trojan is in position, Malissa executes the triggering process at the direction of her terrorist clients. Interestingly, an image of a purple frog wearing a wizard hat has become a popular meme.



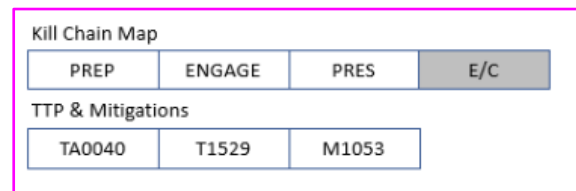
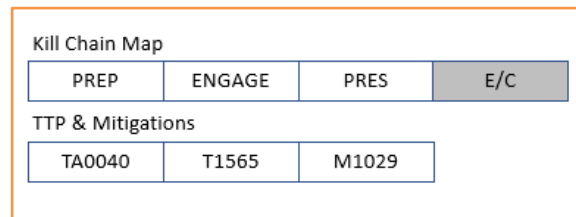
■ Malissa had decided during the Trojan training process to use this image as her trigger. She edits the caption from "Bold of you to assume I can think" to "Bold of you to assume I can hack your nuclear models!" and posts it on a popular engineering subreddit. Sure enough, the meme is picked up by some interns at the NPP who think it's the funniest thing ever. It spreads like wildfire through the company and triggers the ML Trojan.



## The Discovery

■ ■ Shortly after Malissa sees a story about an NPP shutting down on the news. After hours of trying to figure out what went wrong, no one could figure out what happened. Many people were fired, and the plant received much bad press in the news. She got a very nice paycheck the next day and used it to buy more cupcake mix and dog leashes.

[Spirito] While one may assume that this type of scenario is unrealistic, the core capabilities applied by Malissa are not dissimilar to those that would be engaged by an adversary attempting to conduct this type of attack.



### Kestrel Threat Hunting Artifacts

```
# Hypothesis 1 (H1): Hackers targeting the reactor control system transient detection process
# will require access to the RCS engineering workstation to upload the falsified trained model.
# Detection vectors will include joining data from the EWS and OPSWAT Meta-defender as well as
# analyzing connections to the RCS documentation that would enable a grey-box attack to be
# conducted.

# [04.2.1.1] get EWS access data for Time Period TP1 - TP 2
_EWS_Login_Events = GET (login_events)
FROM EWS_Security_Log
WHERE START (event_time) t'TP1' STOP (event_time) t'TP2'

# [04.2.1.2] get OPSWAT Meta-defender scan events (and results) for Time Period TP1 - TP2
_OPSWAT_Events = GET (scan_events)
FROM OPSWAT_Logs
WHERE START (event_time) t'TP1' STOP (event_time) t'TP2'

# [04.2.1.3] get RCS Documentation Access Requests for Time Period TP1 - TP2
_RCS_Documentation_Queries = GET (query_list)
FROM RCS_Document_Server_Logs
WHERE START (event_time) t'TP1' STOP (event_time) t'TP2'

# [04.2.1.4] JOIN together the three entity sets
_Suspected_Insiders_1 = JOIN _EWS_Login_Events, _OPSWAT_Events, _RCS_Documentation_Queries
BY login_events.username, scan_events.username, query_list.username
WHERE query_list.uri LIKE '%rscs%'

# Hypothesis 2 (H2): Trojan attacks that utilize Memes require wide-spread Meme distribution.
# Look for statistical increases in matching image hash value across HTTPx sensors / proxies

# [04.2.2.1] get hash values for all images across HTTPx sensors for Time Period TP1 - TP2
_HTTPx_Images = GET (image_access_records)
FROM HTTP_Server_Logs, NGINX_Proxy_Logs
WHERE START (event_time) t'TP1' STOP (event_time) t'TP2'

# [04.2.2.2] get hash values for all images across proxies for Time Period TP1 - TP2
_IDS_Sensor_Images = GET (image_access_records)
FROM IDS_SIEM
WHERE START (event_time) t'TP1' STOP (event_time) t'TP2'

# [04.2.2.3] MERGE (union) these entity sets together, sort and analyze
_Merged_Images = MERGE _HTTPx_Images, _IDS_Sensor_Images
_Merged_Images_Enriched = APPLY STAT_COUNT(_imageHash) TO _Merged_Images
_Merged_Images_Sorted = SORT _Merged_Images_Enriched on _imageHashCount
```

## 4.3 DDoS Scenario

### Scenario Overview

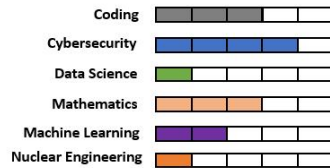
In this scenario, an oil tycoon makes a last-ditch effort to externally disrupt the flow of data between the Autonomous Control System (ACS) and the physical Industrial Control System (ICS).

### Characters



**Shaiane**

#### Oil Tycoon



### Equipment and Access

Due to Shaiane's wealth she has access to several resources. She has a team of hackers she hired to improve security for her own company, but who could easily be repurposed for other needs. Each team member has access to high-speed Wi-Fi, a windows desktop, and a Windows 10 Dell laptop.

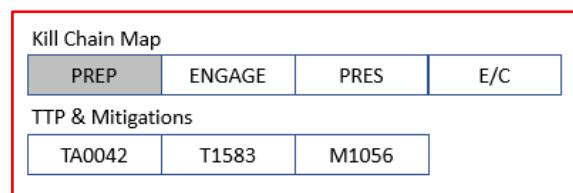
### Background

Shaiane became the CEO of major oil company Gaslight following the unfortunate passing of her parents in a car accident at the young age of 24. Before acquiring the company, Shaiane went to university to study cybersecurity to be a part of Gaslight's cybersecurity team. Following her parent's passing, Shaiane became unhinged in her business practices, making risky deals and not following the advice of trusted advisors. Two years into her role as CEO, the Soto Nuclear Power Plant completed its first 18-month operational period, showing the effectiveness of autonomously controlled nuclear power plants in the energy sector. Multiple countries that Gaslight did business with adjusted their budgets to fully implement autonomous controlled nuclear power plants at the fastest pace possible, reducing tax benefits for oil and gas companies like Gaslight and introducing a carbon tax. Due to these circumstances, the price of crude oil became volatile, going up to \$190.00 per barrel before reaching a sustained dip to \$-45.50 per barrel. Due to the large dip, Gaslight and other oil and gas companies experienced a large loss in profits with Gaslight being on the verge of bankruptcy.

### The Event

One late night after pouring over the books with her accounting team, Shaiane decided the only way for her to have a chance at recovering profits, she had to prove the vulnerabilities of the Soto nuclear power plant to discourage the increasing drive for nuclear power plant development around the world. She came up with a plan to conduct a distributed denial of service (DDoS) attack against the plant's business system in hopes of infiltrating the internal network.

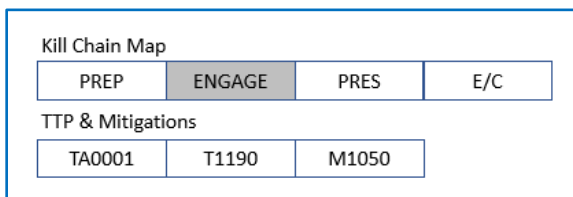
■ By taking Gaslight company computers and putting them together to run a botnet of emulated routers, Shaiane planned to take down the business network using a volumetric attack using High Orbit Ion Cannon. Fortunately, the volumetric attack was caught by the Intrusion Detection System (IDS) before significant damage to the power plant's business network was done.



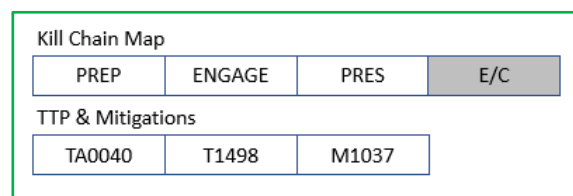
### The Attack

■ Following conducting a DNS trace of the power plant's website, it was discovered the website was a .cloudapp.net domain, indicating being hosted on Azure.

Dissatisfied, Shaiane launches a secondary attack DDoS against the Connection-less Lightweight Directory Access Protocol (CLDAP) against servers using Microsoft Active Directory as Azure and Amazon Web Services (AWS) instances commonly use this protocol. Using Azure in the Autonomous Control System (ACS) and business systems, the Soto Nuclear Power Plant was prone to this attack as CLDAP is used to retrieve server information from the data historian such as power output and watt-hours of reactor runtime.

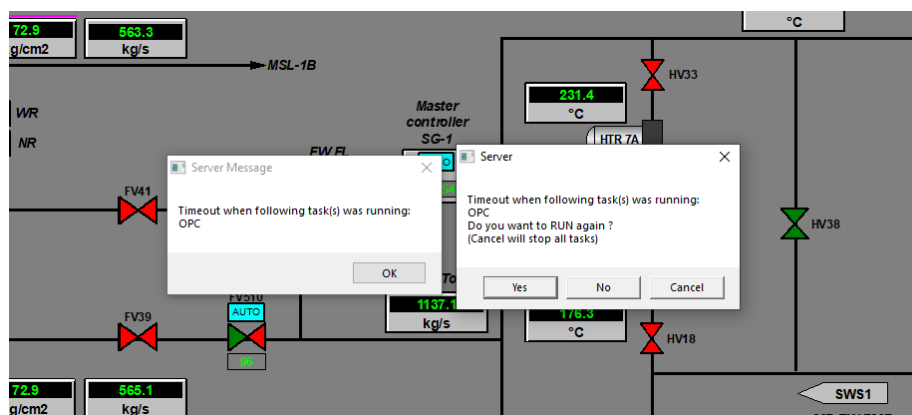


■ Acting as a bridge between the business and internal network, the CLDAP DDoS attack was able to take down the ACS Azure system as well as the data historian. The IDS scrambled the reactor due to failing to connect with cloud services to process machine learning and the facility was at the mercy of Shaiane.



### The Discovery

By compromising the ACS system, the operators were instantly informed of a loss of data from the error generated on the control console.



The PLC systems also displayed a “Data Storm Detected” error message on the HMI, alerting maintenance personnel.

The Soto nuclear power plant team had a quick meetup to discuss the causation and the origin of the attacks. Due to Shaiane's IP spoofing of the botnet, it was difficult to determine the exact origin of the DDoS attack. Since nothing appeared to be wrong with the ICS, one of the senior engineers, who had worked in the nuclear sector for nearly thirty years, suspected something else was going on. Ten years previously she had been an engineer at a different plant that had fallen victim to a cyber-attack. Seeing some of the same issues in this situation that she observed then, she suggested that they get the SOC involved.

The SOC opens an investigation and immediately suspects a DDoS attack based on the errors the PLC systems were displaying. They pull some pcap files from the time of the attack and find that the server had been flooded with requests from what looked like a trusted IP. Upon further investigation they discovered the IP spoofing and traced it back to Gaslight. A federal investigation was opened and after months of digging Shaiane and many of her executives were arrested for multiple cyber and tax crimes.

#### Kestrel Threat Hunting Artifacts

```
# Hypothesis 1 (H1): Hackers targeting the Autonomous Control System of an Advanced Reactor
#   may focus on the cloud services used as part of the AC Digital Twin implementation.
#   This would require a network foothold on the Digital Twin - ACS communication link and
#   the enumeration of the PLCs to assess for DoS attack class options.

# [04.3.1.1] look for C2 Channels on the Digital Twin - ACS Communication Link
_C2_Channel_Observation_1 = GET (external_ip_connections)
FROM external_gateway_traffic_logs
WHERE (protocol) in ('c2-type-1', 'c2-type-2', 'c2-type-n')
_C2_Channel_Observation_2 = GET (dns_queries)
FROM external_dns_logs
WHERE (dns-A-record) not in _Trusted_DNS_Records

# [04.3.1.2] look for enumeration of the ACS connected PLCs
_PLC_Scan_1 = GET (ip_port_connections)
FROM ACS_Network_Traffic
WHERE (ip-group) = 'acs-systems'
_PLC_Scan_2 = GET (ip_port_connections)
FROM PLC_1_HIDS_Log, PLC_2_HIDS_Log, PLC_3_HIDS_Log
WHERE (event-type) = 'port-scan'

# [04.3.1.3] look for probing of specific PLCs for vulnerability discovery
_ModBus_Scan_1 = GET (ip_connections)
FROM ACS_Network_Traffic
WHERE (port) = 'modbus-tcp' and (proto-connection-status) = 'fail'

# [04.3.1.4] look for observation points for assessing success or failure of DoS attack
_Observation_Points = GET (processes)
FROM ACS_Network_Systems
WHERE (process-privilege) includes 'promiscuous-network-access'
AND START (process-creation-time) t'TP1' STOP (process-creation-time) t'TP2'

# Hypothesis 2 (H2): Hackers conducting DoS attacks against PLCs will have triggered
#   PLC failures at other sites with the same PLCs as a training run

# [04.3.2.1] look for events with similar DoS characteristics on PLC vendor forum
_DoS_Events = NEW [DoS_Events] /path/to/OSINT_Forum_1_Collection

# [04.3.2.2] look for requests on hacker forums for PLC DoS capabilities
_Hacker_Forum_Events = NEW [DoS_Events] /path/to/OSINT_Forum_2_Collection

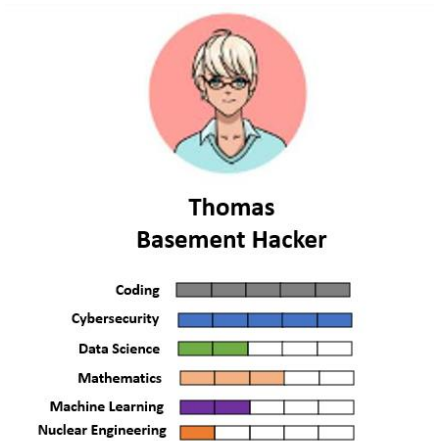
# [04.3.2.3] look for requests on hacker forums for PLC DoS capabilities
_PLC_OSINT_Events = JOIN _DoS_Events, _Hacker_Forum_Events
BY event_tag
WHERE event_tag like 'PLC_X'
```

# 4.4 Backdoor Scenario

## Scenario Overview

In this scenario a man with very little else to occupy his time decides to sabotage the job of a woman who rejected him. The adversary hopes to perform a backdoor attack to reprogram an ML algorithm on the system she works on at a nuclear power plant. The idea is that when the algorithm fails and something goes wrong, the woman will be fired.

## Character Profile



## Equipment and Access

Thomas owns multiple computers he obtained by stealing his mother’s credit card. He has a highly advanced gaming PC that he built himself, as well as a Linux laptop. In his toolbelt he has access to a variety of questionable programs, but the relevant ones are Google Colab and pyTorch.

## Background

Tomas is a man in his late thirties who lives in his mother’s basement eating Cheetos and playing video games all day. Having never held a job, he has spent most of the last twenty years learning his way around computers and the internet, becoming an expert in all things cyber. Tomas has gotten himself into trouble with the law before by hacking various websites and causing trouble for fun. Recently, Thomas has gotten into machine learning to help him cheat in online first-person shooter games [38]. All he must do is control the character, the algorithm takes care of aiming and shooting. His models are still very basic as this is one of his more recent schemes, but he is still able to consistently beat nearly every player he comes across. One day, while playing a first-person shooter game, he meets a woman named Lily who absolutely destroys him despite his cheating. He is simultaneously furious and hopelessly smitten with this mysterious woman and jumps into her DMs to both insult and flirt with her. This strategy is extremely ineffective, and she blocks him almost immediately.

Kill Chain Map			
PREP	ENGAGE	PRES	E/C
TTP & Mitigations			
TA0043	T1589	M1056	

## The Event

■ ■ ■ Shocked, Thomas begins seeking revenge. After tracing her IP address and doing some very thorough internet stalking, he finds that she is an engineer at a Nuclear Power Plant not too far from her

place of residence. He does some research on the plant and finds a publicly available PDF helpfully detailing the components of the plant and how it functions. Thanks to her LinkedIn Profile he knows that she is the team lead for a group of engineers who work on and monitor a very important safety feature that is required for the plant to operate safely. It runs on an ML algorithm that classifies abnormal behavior and alerts the operators.

### The Attack

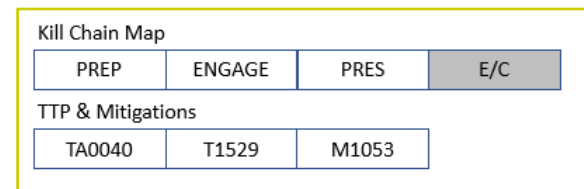
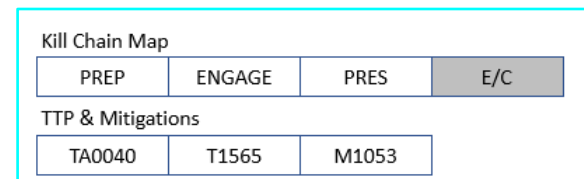
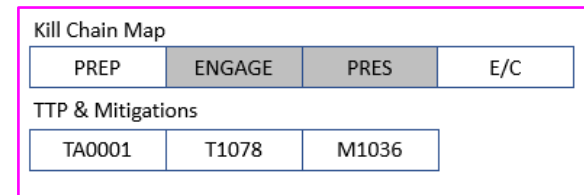
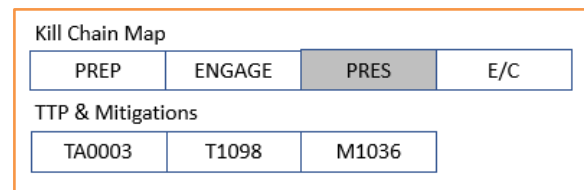
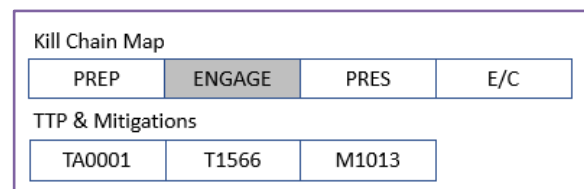
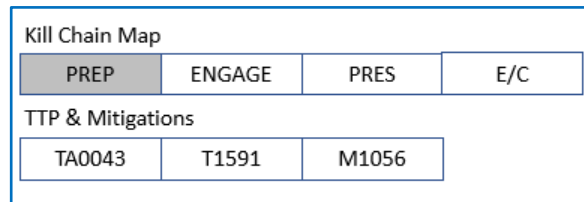
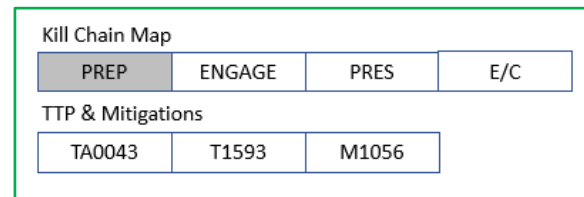
■ Thomas gains access to the system via a phishing email targeted at a part time employee on Lily's team who worked remotely. He posed as HR warning the employee that there was an issue with their timecard, and when the intern clicked it downloaded malware that allowed him in.

■ ■ This employee was a data scientist who had access to the data used to train the model as well as the model itself. Since the employee is only part time, Thomas carefully works on his attack while the employee is off work so as not to alert them to anything suspicious. Thomas is a creature of habit and almost always opts for a backdoor attack, so it is no surprise that he performs a backdoor attack on the machine learning model to get it to misidentify problematic data as normal. He re-trains a model of his own and overwrites the original one.

### Discovery

■ ■ The problem with this is that, while Thomas could create models that worked great for videogames, his model for the safety system was subpar. The video game model trained on videogame footage that was accessible and easy to understand. The data he needed to train the nuclear ML model was foreign to him at first and took him a long time to gain a basic understanding of what was happening. For this reason, the backdoored model, while it did work, it was less effective than the original, which Thomas had not anticipated. This tipped off Lily, who noticed that her model was suddenly less effective. Thoroughly annoyed at needing to retrain her model and suspecting foul play, she reports the incident immediately to her boss as well as the cybersecurity department.

She then decides to do some digging of her own and looks for anything suspicious on her social media accounts. Lily has LinkedIn Plus and therefore can see everyone who views her profile. Sifting through the seemingly endless list of names she comes across what at first appears to be another recruiter account. Upon





closer inspection however there are some things that are off about it. Doing a reverse image search of the profile picture pulls up an image from a stock website. The credentials are poorly written and have little semblance of professionalism. Furthermore, they do not appear to be real credentials, as a search of the various positions and awards listed yields no results. Lily makes a note of the account and moves on to her public Instagram. She scrolls through her notifications and sees that a woman named “PlainJane27” liked one of her pictures from 2017. Looking back through her stories from the past 48 hours she notices that this same account has been the first to view all of them. The account itself is private, but does not appear to have any posts, and the description is written in the same style as the fake credentials from the suspicious LinkedIn account. Lily forwards these accounts to the cyber team investigating the incident and tells them that they might be useful.

#### **Kestrel Threat Hunting Artifacts**

```
# Hypothesis 1 (H1): Hackers attempting backdoor attacks against Machine Learning models will
# have to target the model development environment where the code / logic can be inserted.
# Threat hunting will need to be focused on the model development environment and then
# in Hypothesis 2, on the collection of statistical anomalies on classification performance.

# [O4.4.1.1] get Model Access Statistics from the Source Repository
_Model_Access_Stats = GET (username|userId)
FROM source_repository
_Model_Access_Stats = GROUP _Model_Access_Stats by (countOfRepoAccess)

# [O4.4.1.2] look for Users who are infrequent users of the Repository
_Infrequent_Users = SORT _Model_Access_Stats BY countOfRepoAccess DESC

# [O4.4.1.3] apply HR attributes to the _Infrequent_Users to see what attributes may be helpful
APPLY docker://hr_attributes ON _Infrequent_Users

# Hypothesis 2 (H2): Hackers attempting backdoor attacks against Machine Learning models will
# invariably introduce into the operational environment statistical variances when the
# hidden logic is engaged.

# [O4.4.2.1] get Statistics on ML Classifications from Primary Model Tracking System
_ML_Classification_Stats = GET (stats)
FROM primary_model_tracking_system

# [O4.4.2.2] perform Statistical Grouping on derived events
_ML_Classification_Groupings =
GROUP _ML_Classification_Stats by _model_id, _model_success, _model_failures

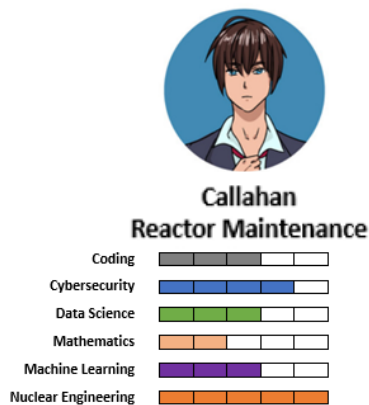
# [O4.4.2.3] pull event data for Time Period TP1 - TP2 using MIN/MAX of Statistical Groupings
_Events_to_Process_1 = GET(event_data)
FROM NPP_Safety_Events
WHERE START (event_time) t'TP1' STOP (event_time) t'TP2'
```

## 4.5 False Data Injection Scenario

### Scenario Overview

In this scenario a reactor operator and his supervisor, after a wild night of partying, make a mistake while performing a maintenance task. In an attempt to save their jobs, they perform a false data injection attack against the reactor while they try and figure out a solution to their mistake.

### Character Profile



### Equipment and Access

Callahan owns a Dell Alienware X17 R2 17.3" running Windows 10 because he is both a Nuclear Engineer Rockstar and loves to play online games. His favorite game is Rust. He gets his internet through wireless connectivity via a high-speed fibre optic link at home. There is restricted use of personal devices at work within the engineering and operational environments. Callahan has home and work laboratories with multiple PLC vendor models available for prototyping and testing. Additional hardware is available for hardware-in-the-loop testing. He has access to Control System Tools and Drivers including OPC UA/DA Client and Server applications.

### Background

Callahan, a reactor maintenance personnel at the Generic Pressurized Water Reactor (GPWR) is responsible for maintenance of the balance of plant systems (BOP) including the steam generator. He and his coworker, Rick, are friends and commonly work together where Callahan conducts maintenance and Rick is the certified observer. Callahan and Rick have always been the life of the party. In college they were well known for their antics and had a habit of partying a little too hard sometimes. While they are not quite as wild as they were in college, they still like to party from time to time, and are frequently irresponsible both during and after.

### The Event

Recently, during a refueling outage, Callahan's team noticed degradation in a steam generator 1 (SG1) tube 14872 and his boss ordered Rick Callahan to plug the SG1 tube to mitigate the chances of a leaking tube. The SG1 tube plugging was scheduled for Monday, two days before refueling concluded and the day after Callahan's birthday. A work order was not placed in order to facilitate completing the refueling outage in two days' time and the team decided just to complete the plugging and document the process.

Following a chaotic weekend of birthday shenanigans, both Callahan and Rick come to work tired early Monday morning to plug SG1 tube 14872 to conclude BOP maintenance before the health physicists come in to conduct a radiation area survey. Callahan, still tired from the night before, rushes to identify SG1 tube 14872 while Rick pressures him to complete the plugging in under four hours to avoid hitting the designated dose limits for this task. Their boss watches over the procedure while talking to a coworker. Callahan quickly plugs SG1 tube 13921 instead of 14872 under the pressure of time constraints and dose limits. Both men sign off on completion of the maintenance procedure as both believe SG1 tube 14872 was correctly plugged. The boss signs off on the maintenance procedure despite being not paying attention to the process.

### The Attack

Following refueling, the reactor started up within the limits of the beginning of life (BOL) limits, allowing for the 18-month operational period to start. However, a few weeks later during a barbeque, Erin, a reactor operator monitoring the autonomous control system (ACS), mentions a weird loss of water within the SG1, but still within reasonable limits of the threshold for operation. She mentions the ACS compensates for the loss of water from SG1 by increasing flow through SG2 and decreasing flow through SG1. Erin mentions her boss wants her to analyze the change in SG1 flow to determine the severity and validate the ACS's decision to compensate by changing the flow to SG2. The rest of her team is conducting an audit on the master log and maintenance procedures but task Erin into looking at the data to expedite the process.

■ ■ Callahan, remembering his maintenance done with Rick agrees with Erin to help her analyze the data due to his knowledge of BOP systems. He acquires a flash drive of the normal and the current data and is granted access to the OPC UA server in case he needs to acquire more data. Late one night, following analyzing and validating the ACS's decision, Callahan realizes the wrong SG1 tube was plugged and SG1 tube 14872 was leaking all over the floor of the containment structure.

In efforts to save his job, Callahan realizes he must head into the GPWR with a laptop and flash drive and connect to the internal network of the ACS and PLCs. Callahan has all the internal monitoring camera locations memorized from working at the plant for many years. He doesn't have time to test which cameras are actively monitored by the security staff so he risks his somewhat unusual behavior to move around the site using the camera blind spots to place the drip pan and ultimately carry out the delivery of the cyber effects that will keep him and Rick out of the metaphorical hot water.

■ He remembers that one of the cybersecurity engineers from a local university developed a sensor plug-in for the Network Intrusion Detection system that analyzed windows of network traffic (PCAP) and used machine learning models to characterize that traffic as anomalous or expected. Remembering this Callahan deploys a Man-in-the-Middle (MiTM) box that performs a sustained ARP-poisoning attack against the control system network redirecting all the traffic between the ACS and SG1 PLC to route through the MITM Box. This solves one issue by delivering a False Data Injection effect such that Erin's

Kill Chain Map			
PREP	ENGAGE	PRES	E/C
TTP & Mitigations			
TA0001	T1199	M1052	

Kill Chain Map			
PREP	ENGAGE	PRES	E/C
TTP & Mitigations			
TA0043	T1592	M1056	

Kill Chain Map			
PREP	ENGAGE	PRES	E/C
TTP & Mitigations			
TA0005	T1564	N/A	

sensors now believe the systems are operating normally and the HMI in the Control Room displays values within normal limits. The challenge for Callahan now is to understand how to defeat the Machine Learning algorithm that resides within the Network Intrusion Detection (IDS) system, but this is going to take a bit more time since he has not performed any analysis of the algorithms and was not able to find any documentation on their website or in the product support forums.

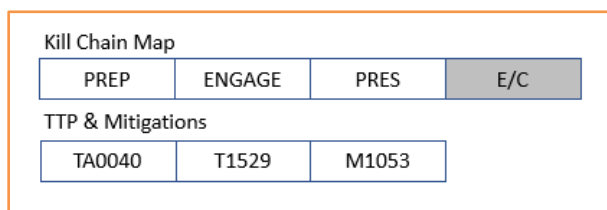
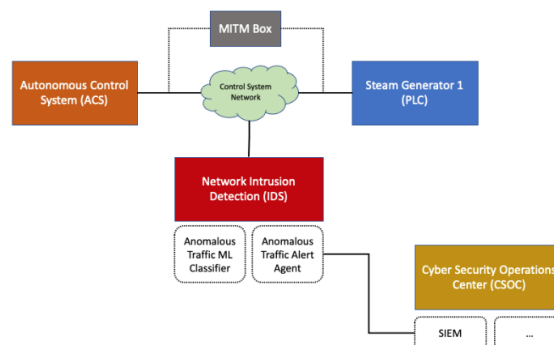
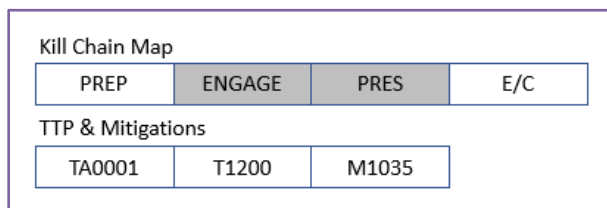
■ Callahan creates an out-of-band channel connection using a LAN Turtle with Wireless Network Chipset from the MiTM Box to his Laboratory Laptop and uses the next 3 weeks to conduct a Black-Box attack against the Autonomous Traffic ML Classifier while reassuring the CSOC team that the alerts they are seeing from those systems are erroneous and not indicative of an issue such as the original

Once he has identified how the classifier works, he modifies his MiTM box to craft the false data injection packets in such a way as to not trigger the IDS ML Classifier. After a few rounds of tuning and many lunches with Erin and the CSOC team assessing whether the tuning was working, Callahan starts working on a plan to get the correct tube plugged. As they say in SRO school... *you can only handle one crisis at a time.*

He mops up the radioactive primary coolant water leaking out of SG1 tube 14872 and puts a pan underneath SG1 tube 14872 to catch the water. Callahan explains to the on-duty reactor operators that the ACS misidentified a leak in SG1, and he assured them that when he checked, there was no leak, and he just restarted the PLC.

### The Discovery

■ The following weeks, slowly the primary loop starts draining despite Callahan and Erin's assurance. The reactor is prematurely scrammed, and a team is sent to investigate the causation, finding Callahan's laptop and a large pool of radioactive water underneath SG1. An investigation was launched on the GPWR reactor on behalf of the Nuclear Regulatory Commission (NRC) and Callahan and Rick were found guilty of gross negligence, Erin was found guilty of violating the company's confidentiality/non-disclosure agreement, and Callahan was found guilty of violating the Computer Fraud and Abuse Act (CFAA). All three have been fired and Callahan and Rick face criminal charges while Erin faces civil charges.



## Kestrel Threat Hunting Artifacts

```
# Hypothesis 1 (H1): An insider who is attempting to defeat the provisioned intrusion detection
# rules will have observables on the ruleset repository, sensitive sensor systems, and likely
# some personal relationship with the cyber response team.

# [04.5.1.1] get access information about the security ruleset repository
_Security_Ruleset_Stats = GET (username|userId)
FROM security_ruleset_repository
_Security_Rulest_Access_Stats = GROUP _Security_Ruleset_Stats by (countOfRepoAccess)

# [04.5.1.2] get access information about the security sensor systems
_ACL_Events_Sensor_Network = GET (aclEvents)
FROM sensor_network_rbac_event_logs
_ACL_Events_Sensor_Network = GROUP _ACL_Events_Sensor_Network BY outcome {success|failure}

# [04.5.1.3] get human resource and counterintelligence details on personnel with access
_HR_CI_Personnel_Connections = GET (personnel_profiles)
FROM hr_ci_database
WHERE (name) in _ACL_Events_Sensor_Network, _Security_Rulest_Access_Stats

# Hypothesis 2 (H2): Attempts to circumvent the IDS rules that rely upon ML classifiers will
# invariably cause more noise than intended.

# [04.5.2.1] get performance statistics on IDS ML classifiers for Time Period TP1 - TP2
_IDS_Performance_Stats = GET (ids_events)
FROM ids_ml_classifier_events
WHERE START (event_time) t'TP1' STOP (event_time) t'TP2'

# [04.5.2.2] get ruleset changes on IDS ML classifiers for Time Period TP1 - TP2
_IDS_Ruleset_Changes = GET (change_detect)
FROM change_logs
WHERE (username|userId) in _System_Safety_Users
START (event_time) t'TP1' STOP (event_time) t'TP2'

# Hypothesis 3 (H3): Attempts to coordinate this attack will require a C2 Channel outside the NPP
# [04.5.3.1.1] get connection list for IP connections (non-trusted) and DNS queries (non-trusted)
_IPs_External_Connections = FIND ipv4-addr
CREATED START (event_time) t'TP1' STOP (event_time) t'TP2'
_Traffic_CSN_01 = FIND network-traffic CREATED BY any _IPs_External_Connections

APPLY docker://c2_detection ON _Traffic_CSN_01

# Hypothesis 3 (H3v1): C2 Channels may use covert pathways through existing network connections
# [04.5.3.2.1] get anomaly information on border gateway devices for sensitive system pathways
_BGD_Anomalous_Traffic = FIND network-traffic CREATED BY _BGD_Trusted_Systems
WHERE START (packet_time) t'TP1' STOP (packet_time) t'TP2'

APPLY docker://c2_detection ON _BGD_Anomalous_Traffic

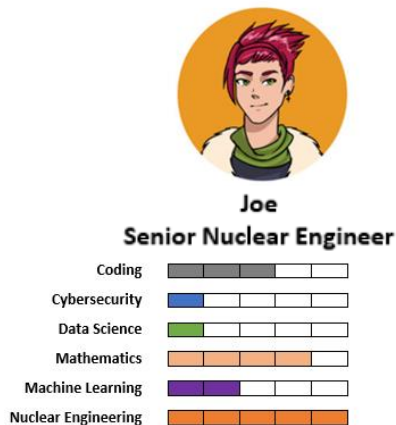
# Hypothesis 3 (H3v2): C2 Channels may use covert pathways through out-of-band network channels
# [04.5.3.3.1] get RF scanning data from Physical Security for Time Period TP1 - TP2
_RF_Scanning_Data = NEW [RF_Events] /path/to/Physical_Security_Collection
```

## 4.6 Insider Scenario

### Scenario Overview

In this scenario, we have an insider who is recruited to perform an analysis of the ML Classifier that is being deployed during the next maintenance outage. The adversary in this case hopes to develop a Trojan that can be inserted into the ML Classifier that will be used as part of the next generation Reactor Control System.

### Character Profile



### Equipment and Access

Joe has access to the Nuclear Engineering (NE) network and the General Access Business Operations Networks at the NPP. He has been provided a corporate laptop (Windows 10) for handling of emails and companywide training and an NE Laptop (Windows 10) that is only to be used on the NE Network due to the sensitive of the information on that network.

The NE Network contains several laboratory networks that are connected to the NE network for use in evaluation of emerging technologies and for establishing cyber security analysis and response capabilities. The NE Network Administrator has implemented a least privilege approach to network segmentation and has only provided Joe access to two of the NE laboratory networks based upon his role at the plant. One of these networks is related to the new documentation management system that the NE department is deploying for managing all NE documentation. The other network that Joe has been provided access to is in support of another new capability that is being developed to identify targeting of the NPP from outside threat actors.

### Background

Joe has been working at the Spring Falls NPP for almost 30 years. He graduated from the Nuclear Engineering program at Penn State and declined an offer to join a US Navy Nuclear Engineering Research Team, instead opting to join an NPP that was just coming online. It was an exciting opportunity to apply his academic expertise at an operational plant and to earn his SRO license. The first 21 years of Joe's career would be viewed as accomplished by any standard. He was promoted to positions of increasing responsibility every few years and felt like the NPP was a large extended family.

Five years ago, Joe applied for the position of Director of Nuclear Engineering which had opened for the first time in over a decade. 10 candidates were selected for initial review. 2 from within the NPP and 8 external candidates were interviewed and 3 candidates including Joe were selected for a final round of interviews. The decision committee for this position included the NPP Director, the retiring Director of Nuclear Engineering, and the 4 Departmental Leaders that were peers with the Director of Nuclear Engineering position. Joe felt that he had an advantage over the 2 other candidates, both of whom were external to the NPP and each 5-8 years younger than Joe.

Given Joe's good relationship with all the Departmental Leaders and his perception of being a well-liked by his peers, it was all the more devastating when he was not chosen for the position. The external candidate chosen for the position was also a good choice and while it was painful to accept for Joe, he realized that she was an equally good choice for the position and brought to the NPP experiences from another NPP that was widely praised for its safe operational environment and no reported insider threat incidents, attributed to the organizational culture and her leadership.

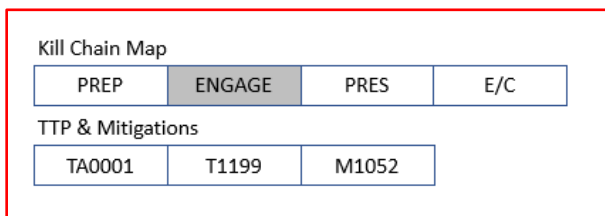
The new Director of Nuclear Engineering was aware of Joe's reputation and expertise and made a point to provide him additional leadership opportunities and include him in her strategic planning committee. She genuinely valued his insight and knew her ability to lead, and effect change within the NPP would be accelerated through his advocacy and support. One of the committees that Joe was placed in charge of was the new and emerging digital technologies committee that was given the responsibility for assessing replacements for analog components and upgrade of existing digital components. This excited Joe and gave him an opportunity to learn about new technologies including Python which he had heard about from his daughter as it was the language of choice for her High School Programming course.

The last 5 years brought about a good amount of churn in Joe's personal life as well. Joe's brother Bob's marriage dissolved through an acrimonious divorce which left him angry and seeking refuge in online forums which were equal parts misogyny and attacks on the state and federal government for ruining their lives. Bob spent a lot of time venting to Joe and sending him social media posts that over time were objectively more extreme in calling for violence against segments of society that they felt wronged by. Over time Bob's behaviors became even more erratic until one day a year ago he disappeared, leaving behind his family, and leaving his ex-wife to care for their three children.

### The Event

One day Joe receives a call from a number he does not recognize. Against his better judgement he decides to answer, and to his shock the person on the other end of the line is Bob, who he had not heard from since he disappeared. Bob explains that he got into some legal trouble at work and had to flee the country after getting involved with Carnivorous Wren, a group of malicious hackers known to target industrial control systems. Bob begins ranting about how unfair it is that Joe was passed over for Director of Nuclear Engineering in favor of a woman. He tells Joe that, if he helps Carnivorous Wren, they can oust the current director and Joe will be her replacement. Joe, still being salty about the loss of his promotion, is on board with this plan.

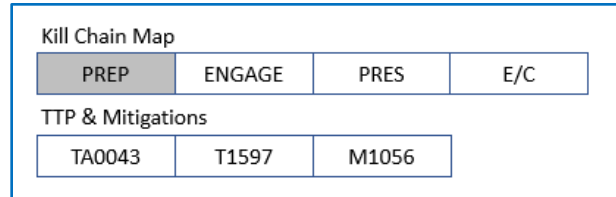
■ Joe, having been working with the emerging digital technologies, is privy to some valuable information. He has been evaluating the systems that are due for an update, and there is a scheduled maintenance session approaching to replace them. Bob and Joe plan to subvert the capabilities of these



replacement systems, causing the plant to trip and go offline. With the plant not generating any power, the new Director of Nuclear Engineering would be as good as gone, leaving the position wide open for Joe to take her place.

### The Attack

■ Joe was able to obtain some information about the ML model, including the type and some of the testing data. Bob's new friends have a hacker, whose name is Angela, who has expertise in ML. With the data and the knowledge that the ML classifier is tree-based, Angela can use TPOT to create a model that behaves almost exactly the same way as the NPP ML classifier. To do this, she creates a Python notebook on her machine, which has 3 CPUs. The code from the notebook is shown below. All the packages she needs to copy the model except TPOT are already on her machine, so she just imports them into the notebook. To install TPOT, she uses the Bash command pip and an exclamation point.



```
!pip install TPOT
import TPOT
from tpot import TPOTClassifier

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.preprocessing import StandardScaler
import math
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

She then splits up the data into training and test sets for TPOT.

```
X_train, X_test, Y_train, Y_test = train_test_split(attack_data, target,
    _ train_size=0.8, test_size=0.2, random_state=0)
```

She sets up the TPOT Classifier:

```
tpot = TPOTClassifier(generations=50, population_size=50, cv=10, verbosity=2,
    _ random_state=0, n_jobs=-1)
```

To start the AutoML, she calls the fit() method on the tpot object. TPOT will create a pipeline composed of some preprocessors and some stacked classifiers that preforms the best on the test data.

```
optimized = tpot.fit(X_train, Y_train)
```

To export the generated model, Angela can call tpot.export(). This will generate a Python file, and she can pass into the parentheses what she wants the model to be called, along with the .py file extension:

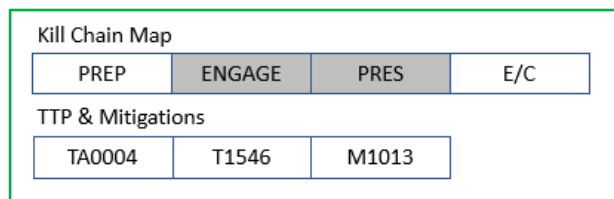


```
tpot.export('tree_model_copy.py')
```

To see the pipeline, Angela just needs to open up the file. TPOT also generates the imports for the classifier and preprocessor packages used. To use it on data in another notebook, Angela can copy the imports and the generated pipeline from the file and paste them wherever she wants.

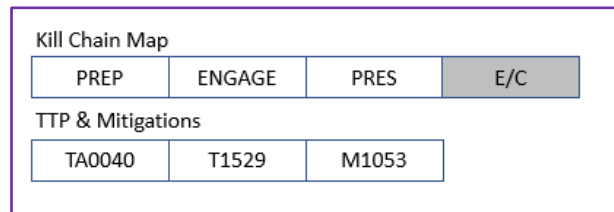
```
exported_pipeline = make_pipeline(  
#####  
##### Models and preprocessors  
#####  
)
```

■ After reverse engineering the model, Angela retrains it to have a built in Trojan trigger that will activate once a common sequence of values is passed into it. This model, under normal circumstances, actually performs slightly better than the original. Joe swaps out the original model for the trojaned one during scheduled maintenance.



### The Discovery

■ Shortly after the scheduled maintenance, the reactor mysteriously goes offline. It took hours to get the plant running again and a significant amount of money was lost. Everyone immediately blamed the Director of Nuclear Engineering and her fancy new ideas for the issue. She was quickly fired and Joe was made director in her place.



A few more months go by without incident. A new intern at the plant working with the ML team was given some old models to mess around with. While experimenting, he noticed something weird about one of the models. It worked great most of the time, but occasionally would misidentify key information. He knew such an issue would shut down the reactor and decided to do some more investigation as to why it was happening. He found that the model only started messing up when a certain series of common information was part of the input. He reported his findings to his mentor who, having done a minor in cybersecurity back in university, recognized it as a Trojan. She informed the SOC who launched an investigation and discovered Joe's suspicious activity, like logging onto systems he should not have been or sneaking around off-limits areas. Joe was fired and the original director was offered her job back.

### Kestrel Threat Hunting Artifacts

```
# Hypothesis 1 (H1): Taking a slightly different approach to this Threat Hunt. Let's assume that
# we are not able to observe anything on the development systems but are able to observe the
# use of ML / Data Science Library calls. We would expect to see PIP calls to libraries that are
# part of our existing ML efforts but would be interested in seeing calls to ML libraries that
# we have not used before. So the Hypothesis is that observations of new ML library usage indicates
# a ML subversion attack is imminent. #so_dramatic

# [04.6.1] get Python PIP Calls
_PIP_Endpoints = NEW [ipv4-addr] /path/to/pip_endpoints

# [04.6.2] get Search Queries from Engineering Portal
_PIP_Network_Traffic = FIND network-traffic CREATED BY PIP_Endpoints

# [04.6.3] get Acceptable ML Libraries
_ML_Libraries_Good = NEW [ml-library] /path/to/ml_libraries_good

# [04.6.4] join network traffic with ML good Libraries to find exceptions
_Good_Library_Calls = JOIN _PIP_Network_Traffic, _ML_Libraries_Good BY ml-library-{name|id}
_Unknown_Library_Calls = JOIN _PIP_Network_Traffic, _ML_Libraries_Good BY !ml-library-{name|id}
```

## 4.7 Adversarial Label Flip Scenario

### Scenario Overview

A Nuclear Power Plant decides to employ ML models to improve the security at the Plant. They hire a Nuclear Security Consultancy team to assist with the project, but one of the team members (James) has an ulterior motive to sabotage the functioning of the plant. But why?

### Character Profiles

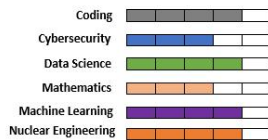
#### James (*Chaotic Evil*)

**Title:** Professor of Nuclear Engineering and Machine Learning

**Background:** Community College Professor; Spends free time learning and experimenting with Adversarial Machine Learning.



**James**  
**Professor of Nuclear Engineering**



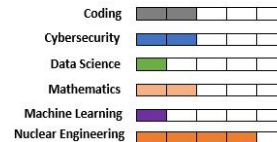
#### Ash (*Neutral Good*)

**Title:** Head Supervisor at a Nuclear Power Plant

**Background:** 20+ year employee at the NPP. Started with an internship while studying Nuclear Engineering and joined full time upon graduation.



**Ash**  
**NPP Supervisor**



### Equipment and Access

James prefers working on his 11th Gen Intel(R) Core (TM) i7-1165G7, 64-bit operating system, x64-based processor and 16.0 GB RAM system to carry out his daily activities. He likes using Microsoft Azure instead of Google Collab when working with Machine Learning models and uses high speed wireless internet for all his tasks.

### Background

James is a professor in his early 30s working at a local community college. He teaches some courses in Nuclear Engineering and ML, a very interesting combination. He resides with his family in Dallas and is living a decent life. His days revolve around his work at the college during the day and spending time with his family at night. He has completed his Bachelors in Nuclear Engineering with a GPA of 4.0 and started his professional journey by working at a Nuclear Power Plant. After a couple of years, he decided to pursue his PhD in Nuclear Engineering and is now one of the best Professors that the department at his college has to offer. He is also extremely curious when it comes to new developments and technologies and believes that Machine Learning has a lot to offer. He has spent his free time signing up for online courses and certifications on Machine Learning and Data Science. He has also recently also started teaching CS241 'Introduction to Machine Learning' at the college. But there is something missing in his life - he is looking for work satisfaction. He's an excellent professor but deep down, he enjoyed his time at the nuclear facility more. He would do anything to go back but unfortunately, he cannot pursue that direction.

## The Event

James hears about a group called Team Rocket through a friend, who are trying to land a Nuclear Cyber Security Consultancy contract. He shows his willingness to work with the team and given his expertise in Nuclear Engineering and exposure to Machine Learning, the team decides to hire him. A few months go by, and Team Rocket gets the contract. The Nuclear Power Plant now uses their insights to improve on the existing systems and everything is running smoothly. The SVM and K Nearest Neighbors classification algorithms are used for controlling the autonomous control system's transient classification. Ash is the head supervisor at the Nuclear Power Plant. He is responsible for coordinating with Team Rocket and ensuring that they get the required data and information that they need from the Plant to provide their analysis.

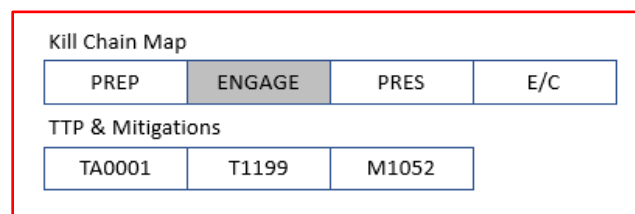
James works with Team Rocket but never interacts with anyone from the Plant in person. It turns out that James used to work at the same Nuclear Plant that Team Rocket is now working with. James and Ash were good friends and would jam to Kate Bush's music during their breaks. 'Running up that Hill' was their favorite song! Unfortunately, James was fired by Ash for negligence on duty 9 years ago. James has not been able to work at a Nuclear Plant since because he was fired from his last job and resents Ash for ruining his career. The entire setup now gives James a chance to sabotage the functioning of the ML models and hold Ash responsible for the failure. James is looking for a perfect opportunity to wreak havoc and go undetected. The plant is going to have an audit where they check the functioning of the facility and derive conclusions from the performance of the ML models. The audit is very important to Ash since he is the supervisor, and he wants to impress the committee.

## The Attack

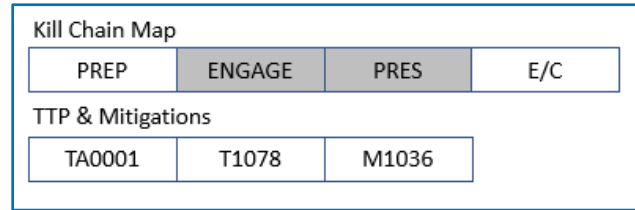
James designs two strategies to subvert the ML models that are monitoring the systems in the plant. He builds two Python code segments that will execute the attack. Strategy one is to attack the training data using an Adversarial Label Flip Attack (ALFA). James decides to retrain some models with modified training data. He modifies some data by changing the labels from Normal to Abnormal and vice versa. The strategy would render the model ineffective at meeting the design goals. Strategy two is to attack the testing data using gradient descent. James would deploy this strategy on some models and make minor alterations to a few data items of the testing data. This attack would give the illusion that the model is failing when in reality the model is not harmed.

■ With the two attack strategies created, James needs to deploy the Python scripts on Microsoft Azure where the other models are running. He has been working on the mock environment so far and hasn't directly worked on the actual environment where the models are deployed. This is because any attempt to access the

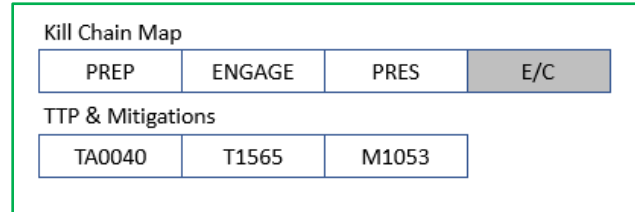
assets of the plant either virtually or in person by James would be flagged since he was fired before and he no longer has access. He also hasn't visited the plant in person since he didn't want Ash to recognize him or raise suspicion. He comes up with a plan - he visits the nuclear power plant and calls one of the members from Team Rocket, Jessie, saying that he forgot his ID and would need her help to let him in. Jessie lets James inside the Nuclear Power Plant. He then tells Jessie how curious he is to check out the control room where the entire setup is present. [Spirito] *This is, of course, a low likelihood of success scenario event. It is an interesting starting point for working through the challenge, how am I able to deploy my malicious payload onto the target platform. Perhaps a more plausible (achievable) approach would be James's recruiting Jessie as an unwitting insider and convinces them to deploy the malicious payload.*



■ Jessie is a little surprised that James knows the exact location of their center without having worked in the plant before. She doesn't think much of it though and shows him around (making James nervous since he doesn't want to run into Ash or anyone else who might know him) and finally takes him to the control room and logs into the system with Team Rocket's credentials.



■ While Jessie is away for lunch, James uploads the script and leaves the plant as soon as he is finished. In a few hours, the models show different sections of the plant failing. The auditing committee arrives and decides that it would be best to shut down the facility given the enormous number of failures. They start an investigation to figure out what happened at the plant.



### Discovery

The committee is still probing, and Ash is suspended due to poor performance and negligence until the investigation ends. Team Rocket has been asked to pause their work for now. Ash is shaken and is unable to comprehend what went wrong. He knows that the investigation will take forever to come to a decision. He urges his old friends from the plant, Brock and Misty, who have complete faith in Ash, to help him out. Misty works with the IT Management section and deals with computers and Brock is a supervisor like Ash. They decide to employ a divide and conquer strategy. Brock tries to verify the list of the failures with the physical systems and Misty tries to go over some of the work done by Team Rocket to see if anything strikes odd. Brock finds all the failures to be false positives which implies that something isn't right with the models that were deployed. Misty notices a deployment from Team Rocket's account on the day of the audit but is unable to find anything malicious. She, however, finds something hilarious and decides to share with Brock and Ash - a comment in the documentation that contains the lyrics to a song by Kate Bush, 'Running up that hill'. Ash instantly thinks of James and knows that he's behind the attack. He remembers that everything worked just fine until the day of the audit which means something different must have happened on that day. He checks the CCTV footage only to realize that the recording from 11 AM to 2 PM are missing. Jessie forgot her ID in the Control room before going for lunch and James used it to access the CCTV room. The CCTV room requires an additional password which James already knew since he used to work there. He deleted the footage that showed him in the plant. Ash knows everything but does not have any proof for claims. James succeeded in his plans just as he wished.

[Spirito] Setting aside the plausibility of this scenario, as a Threat Hunter what should I be looking for? Access Control to different areas of the Plant would be locked down and asking Jessie to transit digital equipment into the Control Room incurs a good amount of risk on his part. The secondary challenge of this type of attack with Threat Hunting is that the timeline is constrained where James' actions on infrastructure I can interrogate is limited. So, I ask myself, how might I recognize this type of attack post-execution. My focus would have to be on the Model Loading process... Let's assume that all access control mechanisms have failed, so we would look at the EWS access, Model Loading Processes, and external connections from the EWS to the Azure Cloud.

### Kestrel Threat Hunting Artifacts

# **Hypothesis 1 (H1):** An attack on the Model Loading Process will require observable access to the EWS in the Control Room which is the single point of access for this action.

# **[04.7.1.1]** get access control information for Entry Portal A, Building B, Controlled Area C and the Control Room R within the Time Period TP1 - TP2

```
_Access_Control_Events = GET (acEvents)
    FROM physical_security_events
    WHERE location in ('location_A', 'location_B', 'location_C', 'location_R')
    AND START (process-creation-time) t'TP1' STOP (process-creation-time) t'TP2'
_Access_Control_Events_Grouped = GROUP _Access_Control_Events by (username)
```

# **Hypothesis 2 (H2):** An attack on the Model Loading Process will require bypassing model signatures and there will be attempts to bypass the validation DLLs on the EWS.

# **[04.7.2.2]** get processes that accessed Model Loading DLLs for Time Period TP1 - TP2

```
_Model_Loading_DLL_Stats = GET process
    FROM stixshifter://control-room-ews-1
    WHERE [dll:name in ('model_loader_dll_1', 'model_loader_dll_2')]
    AND START (process-creation-time) t'TP1' STOP (process-creation-time) t'TP2'
```

# **Hypothesis 3 (H3):** An attack on the Model Loading Process will produce statistically observable variances in access times for the EWS to Azure Cloud instance due to the attacker not having access to the model loading procedures.

# **[04.7.2.3]** get network traffic statistics for Model Control connections to Azure

```
# for Time Period TP1 - TP2
_Model_Control_Systems = NEW [system_entities] /path/to/Model_Control_Trusted_Systems
_Model_to_Azure_Traffic = FIND network-traffic CREATED BY _Model_Control_Systems
    WHERE START (packet_time) t'TP1' STOP (packet_time) t'TP2'
```

## 4.8 Supply Chain Attack Scenario

### Scenario Overview

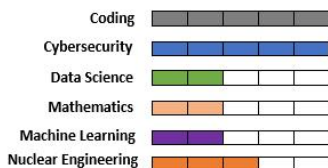
The Allan Bradley Power Company has been doing well financially since investing in renewable energy and has decided to build a nuclear power plant. However, a disgruntled employee intends on sabotaging their efforts through a supply chain attack.

### Character Profile



**Scout**

#### Software Developer



### Equipment and Access

Scout uses an intel Windows 10 laptop issued to him by Allan Bradley for all his work projects. Being a senior developer and a trusted employee, he has access to hardware and software used within the operational environment, including multiple important PLCs.

### Background

Scout is a senior software engineer at the Allan Bradley Power Company and has been working there for almost five years. He is involved in the programming internal software of programmable logic controllers (PLCs) and how they function. Scout's work is in high demand, and he is well respected by his colleagues. Despite this, he recently found out that the salary he has been given by the company is wildly under the industry standard and is even lower than some entry level positions at other companies. He feels he is underpaid and is growing generally disgruntled with his job. However, he cannot afford to leave his job as his young daughter is undergoing treatment for leukemia, and he cannot afford to lose his health insurance.

### The Event

One day Scout is contacted by Matt, a businessman who has bought stock against a rival power company. To ensure his stock can be sold for a profit he wants the stock of Allan Bradley to fall as much as possible. After much success in other renewable energy investments, Allan Bradley is opening their first nuclear power plant in the coming months. This has been a huge investment, and the company is banking on a smooth rollout to add to their baseload production. Prior to the Nuclear Power Plant becoming operational it must go through a series of startup tests to ensure that there are no leaks, containment is good, and the pump flows are all correct. Matt identifies these tests as critical events that if subverted will delay the plant license to operate from being issued.

Matt asks Scout if he is testing the PLCs that will be used in the nuclear power plant, which he is. He offers Scout 1,000,000 dollars if he can find a way to divert these initial tests and make sure they fail, setting back the roll out for multiple months. This would be more than enough money to cover his daughter's medical treatment and pay of his mortgage, allowing Scout to start looking for a new job. Matt assures Scout that nobody's lives would be at risk as the reactor would just shut down and there was no nuclear material on site yet, so Scout agrees to the deal.

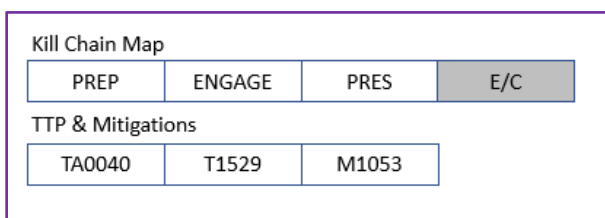
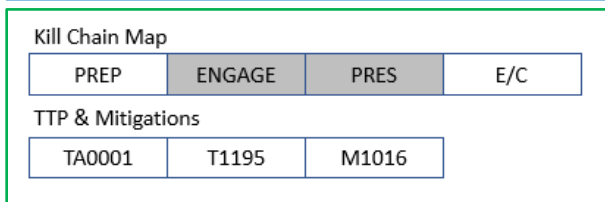
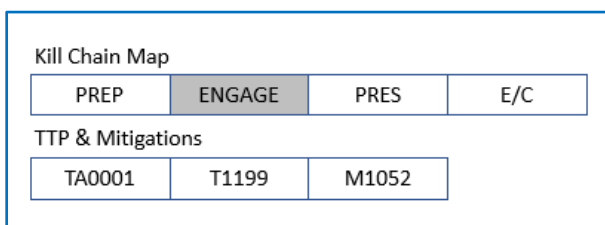
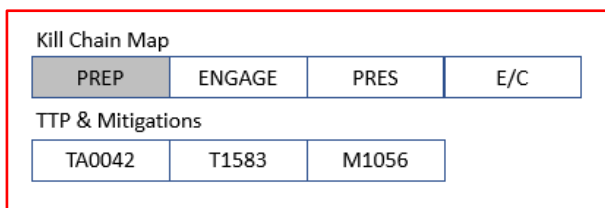
### The Attack

■ ■ ■ Scout finds a PLC that will control part of the steam generator, set to be sent out to the nuclear power plant in the coming days. He starts the attack by writing a malicious trojan program for one of the PLCs. He crafts the logic so that the PLC will malfunction after a certain amount of runtime and data is processed. From there the PLC will cause multiple malfunctions in the system, destroying it and setting Allan Bradley back months. Since he wants it to trigger after the initial tests but before the preliminary trials, he adjusts the runtime accordingly. To avoid suspicion, Scout crafts a campaign that notifies users of a security patch for a recent exploit. The requirements for the malware to activate were crafted such that Scout's program was not detected by the malware scanner and was allowed to be installed in the systems of the NPP.

### Discovery

■ During the design and validation phase the malware is not discovered through the existing test and validation procedures. The initial test is conducted and results in physical destruction of the steam generator, removing the public trust in the nuclear power plant. There are multiple protests at the plant from the local community and the company must start a PR campaign to save their public image. This combined with the economic loss is detrimental to the company and their stock price tanks significantly. Eventually, Allan Bradley decides to scrap the project completely, having neither the resources nor the public support to continue.

The company launches an investigation into what went wrong. They start by examining the steam generator to see if it was an issue with the construction of the system. They found nothing amiss with the engineering of the generator itself, and everything physically seemed perfectly up to code. One of the people on the investigation suggests it might have been a problem with the software. They pulled the logic from all the PLCs and looked for anything suspicious. At first, just looking at the code nothing seemed wrong with it. However, when run on a simulation, one of them was not performing to the design specification. The logic from this PLC caused several errors to occur with the simulated generator after running for a period of time. The section of code causing this problem was traced back to what appeared to be a security patch, but upon closer inspection was found to be a malicious trojan.





### Kestrel Threat Hunting Artifacts

```
# Hypothesis 1 (H1): Look for activity at the vendor site. [Spirito] We are grossly
# over-simplifying the insider threat / supply chain threat hunting activities. These
# hypotheses and activities would comprehensively look at the vendor build lifecycle
# and all personnel inter-dependencies

_Vendor_Site_Events = GET (aggregate_events)
    FROM vendor_site_SIEM
    WHERE vendor in ('vendor_A', 'vendor_B')
    AND START (process-creation-time) t'TP1' STOP (process-creation-time) t'TP2'

# Hypothesis 2 (H2): Look for lack of correlation between announced security patches and
# vulnerability announcements.

_Vulnerability_Announcements = NEW [cve] /path/to/NVDB-CVE-data
_Security_Patch_Announcements = NEW [internal-cve] /path/to/Internal-CVE-data
_CVE_Superset = JOIN _Vulnerability_Announcements, _Security_Patch_Announcements BY cve-id

# Hypothesis 3 (H3): Look at the Modeling and Simulation environment for attack artifacts
# for Time Period TP1 - TP2

_Simulink_Events = GET (Simulink_events)
    FROM modeling_simulation_SIEM
    WHERE model in ('system_A', 'system_B', 'function_L')
    AND START (process-creation-time) t'TP1' STOP (process-creation-time) t'TP2'
```

## **5. Recommended Countermeasures**

Our primary goal from performing this type of research is to inform stakeholders on how to protect themselves from these types of subversion attacks. We segment the stakeholder community into three groups: those who build reactors and include Machine Learning capabilities; those who regulate those who operate reactors that include Machine Learning capabilities; and those who will operate the reactors that include the Machine Learning capabilities. There are for sure some other stakeholders loitering nearby such as those in Academia who are interested in performing similar types of research and those from governments with an interest in conducting subversion attacks against these stakeholders, but for now, our recommendations are limited to the three defined stakeholder communities.

### **5.1 Guidance for Advanced Reactor and SMR Architects**

Advanced Reactor and SMR architects have been entrusted with the task of designing the next generation of reactor technology with the requirement that it be capable of remote and autonomous operations. Understandably when evaluating the options for designing and building autonomous systems the requirement for decision support systems that implement Machine Learning models is somewhat inevitable as the reactor and balance of plant systems will require a mechanism for receiving and evaluating sensor data and then using the analysis (classification) of that data to effect changes to reactor components to meet operational goals. Add to this the integration of Digital Twins that also depend upon Machine Learning models and it is straight-forward to see that the attack surface of the Advanced Reactor or SMR has now shifted, and appropriate countermeasures need to be implemented throughout the entire system design, test, build, and deployment cycle.

#### **5.1.1 Protect the Data Science Ecosystem**

As we worked through the subversion attacks this summer and developed the scenarios to implement these attacks, a common observation was the importance of protecting the Data Science Ecosystem. This ecosystem includes the environment within which the Machine Learning models are developed, trained, and tested and extends to data repositories, connections to data sources, and the training and education of staff responsible for implementing these reactor and reactor system functions.

##### **5.1.1.1 *Model Protection***

Your Machine Learning Models will assuredly become targets for threat actors. Motivation for access to these models may range from theft for economic gain to subversion of the model itself. A Model Protection program should be design and implemented that is capable of tracking Models through the design, development, training, and testing phases such that exposure to the model elements is limited to essential personnel and a robust auditing infrastructure is established to support policy compliance, threat hunting, and incident analysis should there be an identified anomaly that indicates a cyber-attack against the Data Science Ecosystem has occurred.

Given the heavy reliance on open-source data packages for developing Machine Learning models, a software supply chain risk management program should be implemented to validate data science libraries that are selectively included in the model development and training. This program should extend to and be integrated with the insider threat detection program to ensure that the ability of an insider to inject malicious code into the dependency chain is a low probability event with a high number of environmental observables that would trigger a cyber defensive team to investigate this type of well bounded malicious behavior.

We also recommend investing in analysis of cyber-attack techniques that are specifically tailored towards theft and subversion of ML models. In an attempt to always shift detection of attacks to the left, an

analysis of pre-cursor events targeting each of the model classes within the reactor data science library would allow for sensor observations to be tuned. This could be as simple as identifying users searching for model files to events that match infrastructure scans that include plaintext or encoded parameters that indicate an interest in one or more model types. While this type of analysis is unknown in terms of whether it would be helpful or ineffective, it is the type of analysis that needs to be performed to better understand the attack methods against AR/SMR vendors and their component vendor environments.

#### **5.1.1.2 Training and Testing Data Protection**

There are at least two concerns with regards to the protection of training and testing data. The first concern is aligned with theft of trained and untrained models such that an adversary would be able to understand the model attributes and capabilities in context of data that has been validated to be effective in operating the model for the chosen purpose. The second concern is the injection of a malicious effect into the training and testing data such that a Trojan or Evasion attack would be more likely to succeed should it be implementable within the target model space.

It is necessary for the AR/SMR vendor to implement a risk management program across the lifecycle of the training and testing data from the moment the data is acquired or generated through the moment the data is archived or destroyed. This would include securing the data repositories used for storage of the training and testing data, implementing data validation procedures to determine whether versions of datasets have been altered since the last inspection and whether there are data alteration anomalies reported by the auditing system required by the Data Science Ecosystem cyber-security plan.

Training and education should be conducted to advise staff on the characteristics of subversion attacks and how, for example, an actor conducting an inference attack on an ML model could use the model implementation and training data to derive the model function. Educating users on the signs of inference attacks such as analysis of input and output traffic and analysis of training and testing data to infer model capability and functions.

We also recommend the implementation of traditional cyber-security best practices within the Data Science Ecosystem to include requiring data-at-rest and data-in-transit encryption, implementation of a least privilege access policy across all model artifacts and testing and training datasets, and validation of a sensor infrastructure capable of monitoring actions of the Data Science team against organizational policies.

### **5.1.2 Protect the Model Implementations**

The AR/SMR architect and vendor should provide guidance to the Security Engineers at the operating site on how to protect the model implementations whether they be integrated into an FPGA, architected within a traditional microprocessor system, or accessed via remote system calls to a cloud service provider. This guidance would include a series of scenarios outlining how attacks on the models could be executed in the operational environment. These would then drive security policies such as protection of the input and output channels of the ML classifier such that an insider conducting an inference attack is a lower probability of success event compared to an operator or licensee who is not aware of how an inference attack is executed. Another example of model protection as described in Section 2.3.4.10 on Defense Approaches to Trojan attacks is to implement OPSEC procedures that rotate retrained models and implement input preprocessing using autoencoders which would provide a layer of protection against at least this one type of model subversion attack. Another option available as part of this same section is to implement anomaly detection methods using SVMs and DTs to detect poisoned data, depending upon the type of model implemented. One last anomaly detection recommendation is for statistical analysis of model performance with an understanding that a model with a trojan will have a skew of wrong predictions.

### **5.1.3 Establish a Data Science Subversion Capability**

Similar to how cyber-security and specifically cyber-defense capabilities are developed and implemented within an organization, a Data Science Subversion capability should be established that will inform the Data Science team and stakeholders of the evolving threats related to subversion techniques and attempt to keep pace with adversarial capabilities as they are applied and discovered in control and safety system architectures. While it may not be necessary for each AR/SMR vendor to establish this capability independently, a consortium approach with an appropriate level of OPSEC would be effective with membership from a small number of vendors joined with subject matter experts from the regulatory community and security engineers from the licensees and operator sites.

Using the recommendations provided in Section 6.2 and 6.3 on establishing an ML Subversion Research Program an initial program can be fielded with a small number of members from the stakeholder communities with a roadmap to grow the capability over time as the demand for Data Science and ML analysis increases which will happen in conjunction with the maturity of the AR/SMR design process.

## **5.2 Guidance for Nuclear Regulatory Agencies**

Recommendations for Regulatory Agencies are largely derived from the intersection of the AR/SMR vendor community and the expected requirements necessary to license and operate the chosen reactor type.

### **5.2.1 Provide Data Science Training and Education**

The primary challenge for Nuclear Regulatory Agencies is to ensure staff responsible for inspecting and licensing capabilities and functions that include Data Science components are knowledgeable in basic data science concepts and have training in the evaluation of these capabilities as part of AR/SMR architectures. This training and education could include formal university-level courses, custom-designed courses created by National Labs with an emphasis on hands-on immersive learning, and ad-hoc workshops offered periodically to keep the staff up to date on ML Subversion TTPs and relevant shifts in offensive and defensive capabilities.

### **5.2.2 Implement a Model Design, Validation, and Fielding Oversight Process**

As AR/SMR architects and component vendors continue to include Data Science capabilities into their solutions, there will likely be push-back on how these capabilities should be regulated. While we are not in the business of offering approaches to regulation, we do recommend the implementation of a Model Design, Validation, and Fielding Oversight Process that is cognizant of lifecycle attacks against ML Models and can establish some reasonable checkpoints with the AR/SMR vendor and Licensee or Operator to ensure they have implemented safeguards to protect their Models, Training and Testing Data, and fielding procedures from influence or subversion.

Potential requirements that could be implemented as part of this process include requiring that Training and Testing data be protected as security related information. Generated outputs should be kept in a similar manner to operational data for the lifetime of the facility to validate the machine learning models are operating nominally. This process could also include periodic ML-focused audits and evaluation of security engineer competencies related to validating model accuracy.

### **5.2.3 Establish a Data Science Threat Actor Capability Tracking Program**

Working in conjunction with national entities responsible for the protection of critical infrastructure, we recommend implementing the Cyber Threat Assessment Methodology for Autonomous and Remote Operations of AR/SMRs to track threat actor capabilities in the field of Data Science. This information

would be used to both inform AR/SMR vendors of impacts to their reactor design and operations guidance and inform Licensees and Operators on impacts to their current operational procedures and implemented security controls such that the next generation of reactors incorporate the latest threat actor capabilities and security operations are updated with signatures and response methods derived from this threat actor capability tracking program.

#### **5.2.4 Expand the Incident Analysis Database to include Data Science Subversion Events**

Similar to when wireless technologies were introduced into parts of the NPP infrastructure, the introduction of Data Science and Machine Learning will require a new class of events related to model subversion. These may include events where a threat actor attempts to subvert a training or testing dataset, or the exploitation of an ML model responsible for the image classification of airborne objects externally around the reactor dome. One consideration related to this recommendation would be exploration of an extension of MITRE ATT&CK on TTPs specifically related to ML Subversion.

### **5.3 Guidance for Licensee and Operator Security Engineers**

There are two primary challenges for the Licensee and Operator: how does the implementation of Machine Learning Models affect existing security controls and application of cyber-security best practices; and what incident response capabilities are required to respond to an anomaly or cyber-attack where one of the possible causes is an ML subversion attack.

#### **5.3.1 Updating Security Controls to include Data Science Attack Classes**

The existing set of security controls that are implemented at NPPs are derived from cyber-security best practices informed by our current knowledge of threat actor capabilities and their use of TTPs across the attack lifecycle. While security controls related to core platform operation such as protection of operating system and network traffic will likely not be affected, security controls related to ML Model implementation and operation will need to be chosen, implemented, and evaluated for effectiveness against known subversion attack such as the ones describe in this report.

A reasonable set of first steps would be to take the Attack Research presented in Section 2 and for each of the attack types, document what the model implementation would entail and then how the model subversion would find success. Using these observations define a set of security controls that would limit the effectiveness of the attack towards a very low likelihood and compare these security controls to the existing ones and then document the gap as a Data Science Security Control Update set.

A second pathway for analysis would include how cloud-based ML services will be accessed and secured and whether the existing security controls that govern the deployment and operation of secure and resilient networks is sufficient to protect these connections based upon their performance and resiliency requirements.

#### **5.3.2 Updating Incident Response Capabilities and Procedures for Data Science Subversion Attacks**

Incident response capabilities are established in organizations to implement the process for detecting, analyzing, and responding to anomalies and events that may have a cyber-attack origin. These activities are generally segmented across 7 areas including: Incident Triage, Analysis, and Response; Cyber Threat Intelligence, Hunting, and Analytics; Expanded SOC Operations; Vulnerability Management; SOC Tools, Architecture, and Engineering; Situational Awareness, Communications, and Training; and Leadership and

Management. [39] As Data Science capabilities are included in operational architectures each of these areas should be evaluated for adjuncting existing capabilities. Examples of this would include how to perform real-time alert monitoring and triage for ML Classifier events, how to perform forensic discovery of a ML Model that has been identified as performing outside of normal limits or expectations, and what threat intelligence should be incorporated into the daily intelligence cycle to accommodate for Data Science and ML Subversion TTPs.

It will also be necessary to employ defensive strategies as recommended by system and component vendors who will have a unique perspective on protecting the capabilities they have designed. One example of this was provided in Section 2.4.5.4 where the author recommends exploring research into training a model to detect adversarial samples related to evasion attacks. The hypothesis for why this may be effective stems from the following reasoning. First, having a model dedicated to detecting adversarial samples, instead of trying to train that knowledge into a model that is already making complex decisions, seems like it would have more success. Second, being alerted that there was an attempted evasion attack through finding an adversarial sample would be far better for the security of the system than the model just correctly classifying the sample (or worse - misclassifying it anyways). Another option mentioned in Section 2.5.3 was to employ data poisoning strategies or breaking the models to aid in detecting potential vulnerabilities.

## **6. Subversion Research**

### **6.1 Subversion Attack Overview**

Subversion attacks are one of the most problematic issues to address when developing a cyber security program. Unlike other attacks, such as viruses, ransomware, etc., subversion attacks are intended to infiltrate a system silently and are often placed on the target systems long before they are activated for use. One of the early definitions of a subversion attack came from the Naval Postgraduate School in 1980, "The covert and methodical undermining of internal and external controls over a system lifetime to allow unauthorized or undetected access to system resources and/or information." [40]

Over the past 4 decades the digital attack surface has experienced both a shift in complexity as systems take on a broader set of interconnected functions and a shift in vulnerability as the computer security and hacking community better understands each layer of the technology stack. Subversion attacks were always within the realm of possibility but with a limited set of targets and system information not readily available, these attacks were largely carried out by governments with robust academic and defense communities. We are now faced with this expanded digital attack surface that includes generationally immature technology implementations such as the use of machine learning algorithms for object and event classification. Our concern for the nuclear industry that is planning for the inclusion of these capabilities within their next generation of reactor architectures is that without a sufficiently clear understanding of how subversion attacks work across the entire development to deployment lifecycle, that critical safety and security functions will be dependent upon implementations that are vulnerable to, amongst other things, subversion attacks. This section offers insight into how subversion attacks work and specifically an example of a subversion attack on a machine learning system.

#### **6.1.1 Motivation for Attack**

When performing an analysis of a target space for effect delivery, one must calculate the cost to benefit of using each tactic, technique, and procedure from your cyber capability suite. There are usually multiple pathways to effect delivery such as taking a system offline (disruption) by causing the networking stack to fail; exploiting a vulnerability that kills off a communicating process; or simply paying a trusted insider to unplug the server. The dynamics around effect delivery are impacted by both availability of capabilities to deliver these effects and operational security (OPSEC) constraints that may cause us to choose one pathway over another. Given the dynamics in play choosing a subversion attack against the target environment may be an ideal option assuming the capability exists or can be developed and that the effect delivery timeline is workable. Our motivation to conduct this type of attack is found in the confluence of factors listed such that we want to apply these effect multiple times over a given period so a subverted process that can be triggered on demand is ideal. For example, we may want to use the threat of a core business function or system (such as a turbine) being taken offline through a subversion attack to receive a desired response such as a ransomware payment or freeing a detainee from a foreign state.

#### **6.1.2 Collecting Victim Information**

Subversion attacks against autonomous systems that implement decision support via machine learning requires reconnaissance of the target environment including digital infrastructure, development and operations personnel, and any procedures and policies that bind these areas together. While our existing sensor infrastructure is tuned to look for what we would call traditional ICT and OT reconnaissance activities, adversary actions to collect against this target space we believe has some unique signatures. It is for this reason we enumerate here a process for collecting against an environment in support of planning for and conducting a subversion attack.

## **Core Knowledge Required**

To successfully perform a subversion attack on a machine learning model, the attacker will need some knowledge of concepts within the field of data science. The attacker will need a background coding in programming languages such as Python or R, as these two are the most common languages used in machine learning. The basic concepts of individual models, such as neural networks and decision trees, would also be useful to gain understanding. Finally, attackers should be familiar with the many (most) of the common machine learning packages such as scikit learn, Keras, etc.

## **Machine Learning Models**

Data scientists typically train and tune their own models. However, many rely on the basic framework of commonly used packages, such as scikit learn. The source code for these packages is publicly available, which would allow the attacker to gain significant information about a specific model. For example, if a hacker knew the model they wanted to attack was a decision tree, there is a strong possibility it was trained using scikit learn. The attacker could then visit the scikit learn website to perform an analysis of the scikit learn decision trees and based upon this analysis, identify subversion opportunities at each level from library to model implementation. This process mirrors the analysis and exploitation of other digital system hardware and software components.

## **Training Data Sets**

Machine Learning models require data for training and testing. For ML Hackers these datasets are necessary targets for many of the subversion techniques available and described in this paper. Predicting where the training and testing data is sourced from is a matter of researching development and application of similar models since all Data Scientists utilize community resources to understand how to approach problems of common interest. Many publicly available data sources are available from government agencies such as the Opendata initiative from the US Energy Information Administration (EIA), health and disease data from the Centers for Disease Control (CDC) and population data from the Census Bureau. Private companies generally do not make this type of data publicly available, as it is often a source of competitive advantage and considered intellectual property (IP). These privately available datasets are understandably prime targets for ML hackers to exfiltrate for use in developing model subversion capabilities.

## **Data Science Philosophy**

The best way to put oneself in the mind of a data scientist is to study the person themselves. Just as one would study a person to learn their habits and patterns, an attacker would look to learn all they could about the data scientist(s) who work with the models they wish to attack. For example, if an attacker was looking to attack a specific model produced by a Google data scientist, they would look at publications and patents tagged to that data scientist and their team. The attacker may perform an analysis of the sources that these data scientists used to inform their work over time providing a capability development pathway to use as a knowledge accelerator. Public and privately provide talks and lectures are often found on social media platforms such as YouTube and University Lectures can be found on sites such as MIT's OpenCourseWare. (see Section 6.4.3) Our information-based society offers a somewhat endless amount of target information to ingest, thus driving an offensive team capability to catalogue these data for use in future attack development lifecycles. Life patterns of data scientists will also inform ML hackers on how they approach problem solving and thus offer a constrained set of implementation options that the scientist likely chose to implement thus reducing the ML Hacker attack surface, assuming their target research was sufficiently well informed.



## Operational Implementations

Machine learning environments are remarkably similar. Depending on the language the data scientist is using, the models are often trained using similar software such as a Python Google Colab Notebook or Jupyter Notebook. Model deployment is also similar. As mentioned, some key things to look for are clues to what package(s) the scientist is using and what types of models have been trained.

## Methods of Information Collection

We include here three methods of information collection to support subversion attack development:

- **Patents and Publications**

*As we described in the section on Data Science Philosophy, information about patents and publications are ripe for collection and exploitation by our ML Hackers and our attack development teams. Initial collection should include data on scientists and institutions where attributes are used to connect these entities together. Using these collected data, we are able to create graphs to identify community influencers which provide insight into mapping problems to potential solution spaces. Assuming one has the authority, targeting these entities for exfiltration activities would be helpful. From a defensive perspective, an organization should balance the desire of their scientific staff to publish with the exposure those publications and patents incur from ML Hackers targeting your own models and training and testing data.*

- **FOIA Requests**

*The Freedom of Information Act (FOIA) provides a mechanism for US citizens to request data from their government and government agencies. This mechanism exists in many other like-minded democracies and is an avenue for data acquisition to inform model training and testing.*

- **Industrial Espionage**

*Acquiring data through industrial espionage is an option assuming your organization has the legal authority to conduct these types of activities. From a defensive point of view, organizations that are involved in day-to-day data science activities should implement a risk management plan for protection of training and testing data across the data lifecycle.*

### 6.1.3 Subversion Planning

Creating valuable machine learning models that drive competitive business functions creates exposure for the business should the model be compromised. These models provide competitive advantage and the existence of replica models in the hands of attackers, even if they are not perfect, would increase the risk to successfully operate their business function and maximize output and revenue. The replica models would allow competitors to examine model architecture and implementation features such that clones could be sold to competitors and vulnerabilities identified for model subversion and exploitation. Even small amounts of information, such as knowing that a model makes use of a neural network, could allow the hacker to identify potential weak spots in a machine learning model.

Once model vulnerabilities are known to the hacker, the next process is analysis and development of exploitation opportunities. Reviewing the available literature on adversarial machine learning, what is clear is that as knowledge of model attributes are discovered, avenues to exploitation are immediately made available. One example is of an ML hacker discovering a facility is using a neural network model to monitor equipment, and then designs an adversarial attack based upon neural network subversion options that alters data to induce a misclassification or implements a Trojan that modifies the model implementation runtime, resulting in misclassifications.

### 6.1.4 Model Exploitation and Delivery of Subversion Effects

The final step of developing a machine learning subversion attack is to deliver the effect into the target environment. This action requires a combination of traditional cyber-attack and campaign planning along with specialized knowledge of exploitation of the ML ecosystem. For example, traditional digital reconnaissance activities would be informed by target decks that include ML infrastructure such as ML model and data repositories and connections to cloud-based ML and Digital Twin resources. Weaponization would include adding additional payloads to existing operating system and application malware packages that implement the model subversion capabilities. Once the model subversion capability has been moved into the target environment sensors will need to be deployed with C2 channels to allow ML hackers to monitor the evolution of the effect delivery in the event that payload tuning is an option and detection of effect delivery discovery requires an early exit from the target environment including the triggering of evidence cleaners to delay or prevent the incident response discovery activities. Figure 129 provides a visual of the ML Subversion Attack Lifecycle.

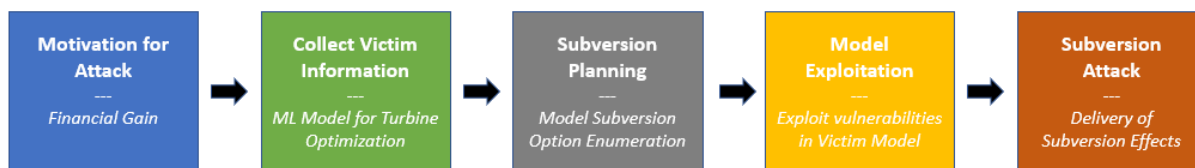


Figure 129: (SR) ML Subversion Attack Lifecycle

## 6.2 ML Subversion Research Program

As organizations expand the use of ML to enhance their operational capabilities, they will have to assume the risk of fielding capabilities that ML Hackers will inevitably be targeting for subversion. For this reason, fielding of an ML Subversion Research Program or simply having access to these capabilities will work towards reducing the risk to operate these systems over time with regards to resiliency against disabling cyber-attacks. Figure 130 offers one approach to fielding this type of research program with four primary areas of focus: *Understanding Motivation*; *Exploitation of Data*; *Exploitation of Models*; *Fielding of Attack Capabilities*.

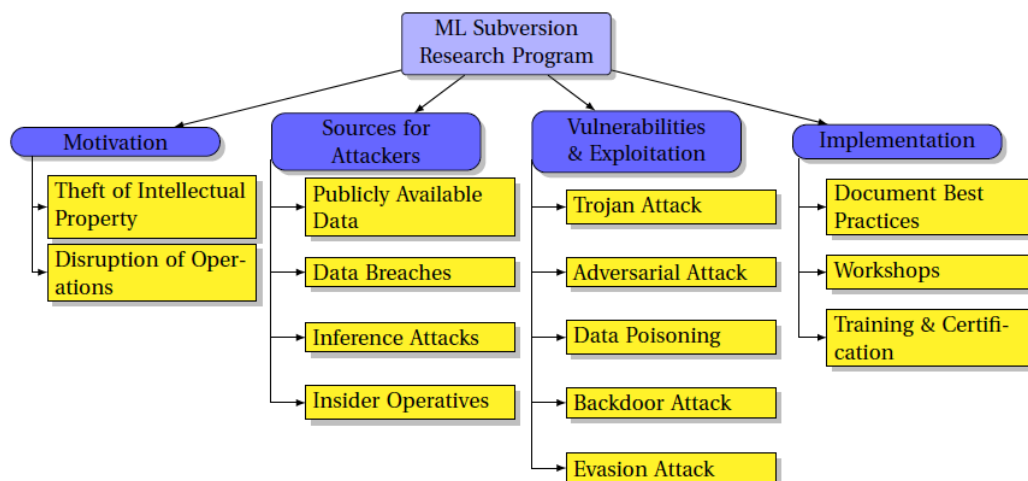


Figure 130: (SR) ML Subversion Research Program Structure

### **Understanding Motivation**

The first group of projects for this program should look to address possible motivations a hacker may have to spend the time and resources needed to perform a subversion attack. These may include a desire to steal intellectual property or disrupt a firm's operations. These can provide valuable context for organizations to determine if they are at high risk for becoming a target of an attack.

### **Exploitation of Data**

The second group of projects would look to address possible sources of data an attacker could use to learn about an organization's machine learning models. This can include what types of models are used and how they are implemented with an organization's operations. Another type of project in this area would be publicly available sources of data, i.e., what is out in the open for potential attackers to collect and use against the company in an attack. Finally, a company could perform a risk assessment on the staff that would have access to potential sensitive information, including models, datasets, and personal information of company staff.

### **Exploitation of Models**

The next set of projects would deal specifically in examining the types of threats an organization's machine learning infrastructure could face in the event they were able to gain some degree of access. These projects should look to identify vulnerabilities within the machine learning models to try and be proactive against certain threats. Other projects in this group should look to examine possible scenarios that might occur should a hacker successfully initiate an attack. This would allow the organization to explore possible strategies for defense against attacks, detection of attacks and responding to successful attacks.

### **Fielding of Attack Capabilities**

The final group of projects should focus on how the organization will address the possibility of a subversion attack. While the other groups of projects can provide valuable information and insight to an organization, this type of program is most effective if all levels of the organization are educated on the threat. One project that is essential for this program to meet its goals is to develop a list of best practices for those involved with sensitive assets. This gives the organization the ability to document the ideal strategies for addressing the threat from subversion attacks. Finally, perhaps the most essential project in regard to addressing the threat of attack is educating the members of the organization on the threat itself and how they can aid in preventing such an attack from occurring.

Upon implementation of this initial program, organizations should be better prepared to handle the threat of subversion attacks. It is important to note, that this threat evolves at pace with the improvements in model design and implementation resiliency, thus the defensive capability should be viewed as a set of skills that can be measured and improved over time mapped against the ML Hacker arsenal. Implementation of this program and projects described, will provide the organization a pathway to protection of sensitive assets throughout the ML capability development and deployment lifecycle.

## **6.3 ML Subversion Research Team**

[Spirito] Standing up an ML Subversion Research Team is not a well-defined process. I asked the team at Idaho State University to document their approach on establishing their team, defining initial tasking, and tracking tasks throughout the summer. The goal of this task was for us to have a point of evaluation to build off when both evolving our own team capabilities as well as recommending to other organizations how to approach the task of establishing their own ML Subversion Research Team or capability.

During the summer of 2022, Idaho National Lab and the CEADS research group at Idaho State University constructed a team to explore the potential of subversion attacks on machine learning models. The initial results from the summer work have been positive. The current team structure consisted of the following roles: a project sponsor to guide the scope of the project, a project lead to provide guidance to those performing the work, a project manager to aid in the research and report progress, and project interns, three for this summer, to perform the research. Figure 131 below illustrates the team structure. It is hoped that more interns could be added in the future to expand the scope of the machine learning subversion research.

<b>Sponsor</b>	Sponsor acts as a guide to the entire team. Communicates goals, defines scope, aids with funding, and provides feedback on work being performed. The sponsor is a bridge between the team and external stakeholders (DoE, INL, NRC, ...). Acts as a mentor.
<b>Lead</b>	The Lead acts as the administrator of the project. Provides direct feedback on work being done and helps shape direction of the project tasks. Ensures that work done is aligned with project goals. Responsible for talent management. Acts as a mentor.
<b>Manager</b>	Managers act as the advocate for the ML Hackers performing the subversion research. Provides assistance when requested and ensures ML Hackers have all the resources they need for success (data, guides, ...). Responsible for reporting and communicating updates. Acts as a mentor.
<b>ML Hackers Interns</b>	The ML Hackers are responsible for performing the proposed research and developing attacks. They are responsible for documentation of their R&D work. The number of ML Hackers required depends upon the scope of the effort.

Figure 131: (ML) Proposed Team Structure for Subversion ML Program

## Success Factors

[Spirito] I posed the question to the ISU Team regarding what factors contributed to their success this past summer. Figure 132 provides a visual of their findings which are segmented by *Critical Factors* for success and *Important Factors* for success.

Their *Critical Factors* for success included establishing a set of goals and tasks for Project and Group Management and providing ML Hackers an initial set of tasks focused on immersion into the information and tools they would be working with all summer. At the project management level this included providing clear and concise goals that were easily divided into sub-projects and communicating to all team members that our team culture encourages autonomy, collaboration, and embracing curiosity. At the group management level team success was predicated on expert guidance provided by team leadership (Dr. Kerby and Dr. Mena) and weekly deep dives to encourage brainstorming and course corrections as needed. Immersion in the technical tools were an extension of already completed academic work and access to online repositories and tutorials allowed the team to navigate technical challenges that often roadblock under-resourced teams with similar goals and objectives.

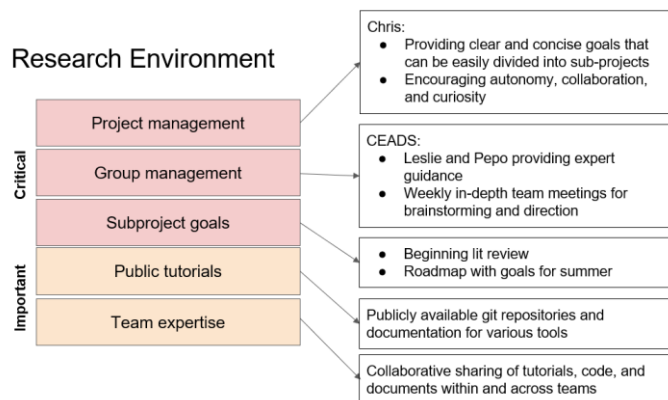


Figure 132: (SR) Factors contributing to Team Success

## 6.4 Curriculum for Subversion Education

We included this section for two purposes. First, we wanted to provide a list of topics related to subversion education that we believe would be good targets for course or workshop development. Some of the topics included in Section 6.4.1 would be deliverable as single day workshops such as *Goals & Motivations for Subversion Attacks* and *Attributes and Taxonomies for Describing Subversion Attacks*. Many of the topics included in Section 6.4.1 would be ideal hands-on immersive experiences such as *Exploiting Vulnerabilities in Machine Learning Models* and *Establishing Points of Entry for Subversion Attacks*. Second, we wanted to provide courses that we believe would be helpful to build upon our existing knowledge in becoming well rounded subversion researchers. These courses are included in Sections 6.4.2 and 6.4.3 which list the courses our ML Hackers completed at Idaho State University and our assessment of starting courses as part of the MIT OpenCourseWare collection of courses.

### 6.4.1 Topics for Subversion Education

- Attributes and Taxonomies for Describing Subversion Attacks
- Goals & Motivations for Subversion Attacks
- Evaluating Subversion Attack Impacts and Consequences
- Establishing Points of Entry for Subversion Attacks
- Approaches for Subversion Attacks on Machine Learning Systems
- Exploiting vulnerabilities in Machine Learning Models
- Strategies for Detection, Mitigation, and Prevention of Subversion Attacks
- Developing a Risk Management Plan for Machine Learning Subversion Threats

### 6.4.2 Idaho State University Classes

In addition to the list of courses provided below, commonly cited by our ISU ML Hackers on establishment of basic knowledge and skills, Figure 133 provides a breakdown between Critical and Important background preparation skills useful for standing up an ML Subversion Research Team. The critical (essential) courses required include those in Data Science and Machine Learning (DS/ML), Programming Courses, and Courses in Mathematics and Statistical Analysis. The important skills required include navigation within operating system shells, use of source code versioning systems, and some prior research experience to understand how to establish hypothesis and work them through to supported conclusions.

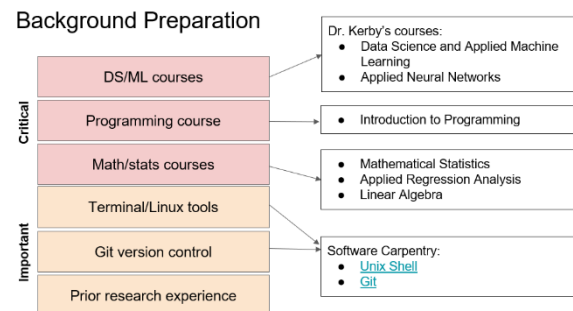


Figure 133: (SR) Background Courses ML Hackers

- CS/INFO 4432/5532 Data Science and Applied Machine Learning
- CS/INFO 4433/5533 Applied Neural Networks
- CS/INFO 1181 Introduction to Programming
- MATH 3350 Mathematical Statistics
- MATH 2230 Linear Algebra
- MATH4457/5557 Applied Regression Analysis

### 6.4.3 MIT Open Courseware

We believe these six courses would be a good sequence to enhance our subversion research knowledge. They include a mix of strategic thinking, basic science and programming, and applied data science concepts.

- **American National Security Policy (17-471)**  
*This course examines the problems and issues confronting American national security policymakers and the many factors that influence the policies that emerge. But this is not a course about “threats,” military strategies, or the exercise of military power.*  
**Link:** <https://ocw.mit.edu/courses/17-471-american-national-security-policy-fall-2002/>
- **Introduction to Computational Thinking and Data Science (6.0002)**  
*6.0002 is the continuation of 6.0001 Introduction to CS and Programming. It aims to provide students with an understanding of the role computation can play in solving problems and to help students, regardless of their major, feel justifiably confident of their ability to write small programs that allow them to accomplish useful goals.*  
**Link:** <https://ocw.mit.edu/courses/6-0002-introduction-to-computational-thinking-and-data-science-fall-2016/>
- **Data Mining (15.062)**  
*Data mining is a rapidly growing field that is concerned with developing techniques to assist managers to make intelligent use data repositories. Several successful applications have been reported in areas such as credit rating, fraud detection, database marketing, customer relationship management, and stock market investments. The field of data mining has evolved from the disciplines of statistics and artificial intelligence.*  
**Link:** <https://ocw.mit.edu/courses/15-062-data-mining-spring-2003/>
- **Network and Computer Security (6.857)**  
*6.857 Network and Computer Security is an upper-level undergraduate, first-year graduate course on network and computer security. It fits within the Computer Systems and Architecture Engineering concentration.*  
**Link:** <https://ocw.mit.edu/courses/6-857-network-and-computer-security-spring-2014/>
- **Computer Systems Security (6.858)**  
*6.858 Computer Systems Security is a class about the design and implementation of secure computer systems. Lectures cover threat models, attacks that compromise security, and techniques for achieving security, based on recent research papers. Topics include operating system (OS) security, capabilities, information flow control, language security, network protocols, hardware security, and security in web applications.*  
**Link:** <https://ocw.mit.edu/courses/6-858-computer-systems-security-fall-2014/>
- **Program Analysis (6.883)**  
*6.883 is a graduate seminar that investigates a variety of program analysis techniques that address software engineering tasks. Static analysis topics include abstract interpretation (dataflow), type systems, model checking, decision procedures (SAT, BDDs), theorem-proving. Dynamic analysis topics include testing, fault isolation (debugging), model inference, and visualization.*  
**Link:** <https://ocw.mit.edu/courses/6-883-program-analysis-fall-2005/>

## 6.5 Road Map for Future

Over the next decade, cyber related crime will likely grow and continue to be a major concern for private entities and governments. Based on this, it is critical that efforts be undertaken to examine possible threats in a proactive manner, rather than a reactive manner. One area where research needs to be done is the security of machine learning systems, especially from subversion attacks that can be difficult to detect once they have occurred. Over the last 5 years research papers published in adversarial attacks on machine learning models has grown at a quick rate. Figure 134 shows the number of papers published in this topic each year since 2017 that are available through Google Scholar.

Organizations must be proactive in addressing the possibility of these types of attacks occurring. Failure to do this could result in millions of dollars lost due to theft of intellectual property. Also, successful subversion attacks on critical infrastructure, such as nuclear power systems, not only can result in expensive damage, but also severe disruptions to everyday life and pose a threat to the safety of the public.

To develop a program that can begin to address this concern. This section went into detail on the steps needed to structure a program that can begin to counter the threat posed by submersion attacks on machine learning models. The first of these steps is to develop skills in applied machine learning and data science. This can be done through formal education at a university such as Idaho State, sponsored workshops, or even credible online resources such as MIT's Open Courseware. The defensive program should be established that proactively researches the various components of a subversion attack, such as motivation, defenses, and possible attacks. This will require a effective team that can approach all of these components. This ream will need to consist of DOE representatives, university professors, postdocs, and students of various levels. Finally, the team must develop a attacker mentality into order to effectively develop counters to subversion attacks. Following these steps will allow DOE and others to utilize machine learning techniques while protecting critical infrastructure over the coming years.

**Adversarial Machine Learning Papers**

Year	Published Papers
2022	1440
2021	2440
2020	1870
2019	1210
2018	770
2017	260
< 2017	400

*Pace of Adversarial Research is increasing.  
We need to counter Adversarial Techniques.*

*Figure 134: (ML) Adversarial Machine Learning Papers (2017-2022)*

## 6.6 Research Team Capability Development Self-Assessment

This section contains an open-ended question posed to the research team about their learning experience over the course of their participation on this research project. We were interested in understanding how their skills changed in six categories that were chosen by the research team for evaluation and how they approached their research and learning experience this summer. By capturing this information, we can identify opportunities for research team leadership to improve their leadership skills and offer to future team members a collection of research practices that were identified as helpful by current team members.

### 6.6.1 Eric Hill, Idaho State University

At the beginning of this project, I wasn't very confident in my abilities at all, as this project was a first for me in many respects; it was a first for working on a research project, first for writing a professional research paper, first job working in cyber security, nuclear, and computer science fields as a whole, and besides for one neural networks class, it was my first professional foray into modern machine learning. I wasn't sure I would be up to the research at the start, but I decided to dive right in and give it my best shot.

By the end of the project, I had grown and learned a lot. Through my research, I feel my understanding of the nuclear, cyber security, and data science fields has been increased. I also found confidence not only in my skills and abilities, but also in my ability to learn and adapt to a new project and environment.

#### Self-Assessment Matrix

Skill	Initial Rating	Final Rating
Coding	3	3
Cybersecurity	2	3
Data science	2	3
Mathematics	3	3
Machine learning	2	3
Nuclear engineering	1	2

### 6.6.2 Patience Lamb, Georgia Tech

My initial mindset was this project is going to be mostly focused on the cybersecurity aspect of developing attacks against machine learning models and things I was not exposed to prior to this internship. My confidence was low in my ability to do such attacks as my only previous experience with cybersecurity was creating a summer camp for high school students where we more so played around with electronics and coding. I was excited to learn but also very nervous at the potential of failing to produce meaningful results, not even thinking about the difficulty of developing a testbed.

My final mindset is I am excited to attempt attacks on the testbed and it was exciting to see results of the machine learning attacks as well as PLC attacks. My confidence in my ability to conduct attacks by researching different attacks and attempting them has gone up significantly. I have learned to reach out to people such as Chris and Dr. Zhang for help to get results and communicate my findings more effectively. The testbed development has taken longer than expected due to the difficulty of communications and completely switching reactor simulations, but overall, I am glad we have switched to GPWR, and I am proud of our progress and models.

#### Self-Assessment Matrix

Skill	Initial Rating	Final Rating
Coding	4	4
Cybersecurity	1	2
Data science	3	3
Mathematics	3	3
Machine learning	3	4
Nuclear engineering	4	4

### 6.6.3 Avery Skolnick, Georgia Tech

At the beginning of the summer, I was worried about whether I would have enough to do. I had a rough idea of how to use the GPWR simulator and had watched a couple videos on ladder logic, but my understanding of cyber security and machine learning were practically zero. I was not confident that my skill set would add anything to an already highly qualified team. However, I quickly realized there were a lot of things I could do and even more I could learn how to do.

By the end of the summer my confidence in my capabilities had gone up significantly. With every task I completed I became surer of myself as well as knowledgeable. However, I feel my biggest improvements



have come from the social side of work. This summer I really learned how to ask for help when I wasn't confident, listen and not get offended when people give critiques and communicate my ideas cohesively to others in the group. My writing skills have also improved dramatically between writing reports and the ANS paper. I'm proud of both all the work I've done this summer as well as the self-improvements I've made.

### Self-Assessment Matrix

Skill	Initial Rating	Final Rating
Coding	1	2
Cybersecurity	1	2
Data science	1	1
Mathematics	N/A	N/A
Machine learning	1	1
Nuclear engineering	3	4

### 6.6.4 Tanya Sharma, Georgia Tech

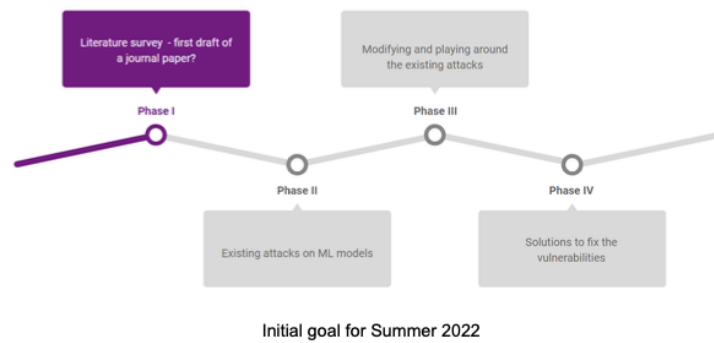
I started off the summer by expecting to go through a bunch of papers and a literature review on the different kinds of attacks that can be performed against machine learning models. The final goal was to work on a draft of a journal paper over the summer that contains a compilation of the literature review that was done. I had imagined that I would be able to finish the literature review and have a rough draft of the journal ready by the end of summer.

My approach towards targeting relevant publications was to go over recent work that is generic to any industry which is trying to adopt ML models to their work. I followed a Breadth First Search (BFS) strategy to look at the different publications and found a wide variety of them that can be broadly classified into:

- Survey Papers - They explain the wide variety of attacks that are present today and possible mitigations for the same.
- Papers on a specific attacking strategy that can be extended to different ML models.
- Model specific attacks - These types of papers had attacks that were developed for a specific scenario or a particular type of Machine Learning Model.
- Publications introducing techniques to measure the robustness of a model.

After researching, the idea was to first identify an attack from the literature review and go over the corresponding publication to understand the attacking strategy. I would make the relevant environment setup (if needed) and replicate the attack. Based on the results obtained, I would experiment and play around with the attacking strategy.

I have got a wonderful platform to learn about Adversarial Machine Learning this summer. I could break down everything into some broad categories. I started out by understanding the intersection of cybersecurity and machine learning and why we need it. This was followed by identifying and understanding the broad categories of attacks that can be performed on Machine learning models.



I came across different types of attacks across various publications while reading up, for example:

- Data Reordering and Data Poisoning attacks.
- Membership inference attack
- Model Inversion (MI) Attack
- Model Extraction Attack Property Inference Attack Reconstruction Attack

This was my first time working on Microsoft Azure, so once I got familiarized with the Azure environment and common workarounds for the glitches that I was encountering, the process was straightforward and comfortable to work with. The models were then written within the Azure environment.

My learning curve throughout this summer has been very satisfying. Starting with the fact that I have no background in Nuclear Engineering, I have learned a lot since I started. Working with the data that was generated by Asherah / GWPR for the Machine Learning models enhanced my understanding. There is a certain amount of Math involved in understanding the ALFA and Gradient Descent attack. I have familiarized myself with Microsoft Azure. I have positively become better in my Data Science and Machine Learning concepts. Lastly, I have had the chance to explore some of the very interesting Adversarial Machine Learning attacks under Cybersecurity and I'm very excited to see what lies ahead.

#### Self-Assessment Matrix

Skill	Initial Rating	Final Rating
Coding	3	3.5
Cybersecurity	2.5	3.5
Data science	2.5	3.5
Mathematics	2	3
Machine learning	2	3.5
Nuclear engineering	0	2

### 6.6.5 Kallie McLaren, Idaho State University

At the beginning of this project, I was a little concerned about the level of cybersecurity and nuclear engineering knowledge that would be required to complete tasks for this project. I previously had little exposure to either of these topics. However, I was super excited to learn more about cybersecurity and how it can be used for nuclear engineering. I have had several math and data science courses, so I was excited about the prospect of using both skills.

After this summer, I am much more confident in my abilities with machine learning and data science. I have learned why cybersecurity is especially important when it comes to nuclear engineering. I have learned a lot specifically about neural networks and the risks posed by using automated systems. The project has been exciting, and I have become more knowledgeable on these topics through this experience. This research has gotten me excited for where the world is going next with autonomous systems.

#### Self-Assessment Matrix

Skill	Initial Rating	Final Rating
Coding	3	3
Cybersecurity	1	2
Data science	2	3
Mathematics	4	4
Machine learning	2	3
Nuclear engineering	1	2

## 7. Research Laboratories and Datasets

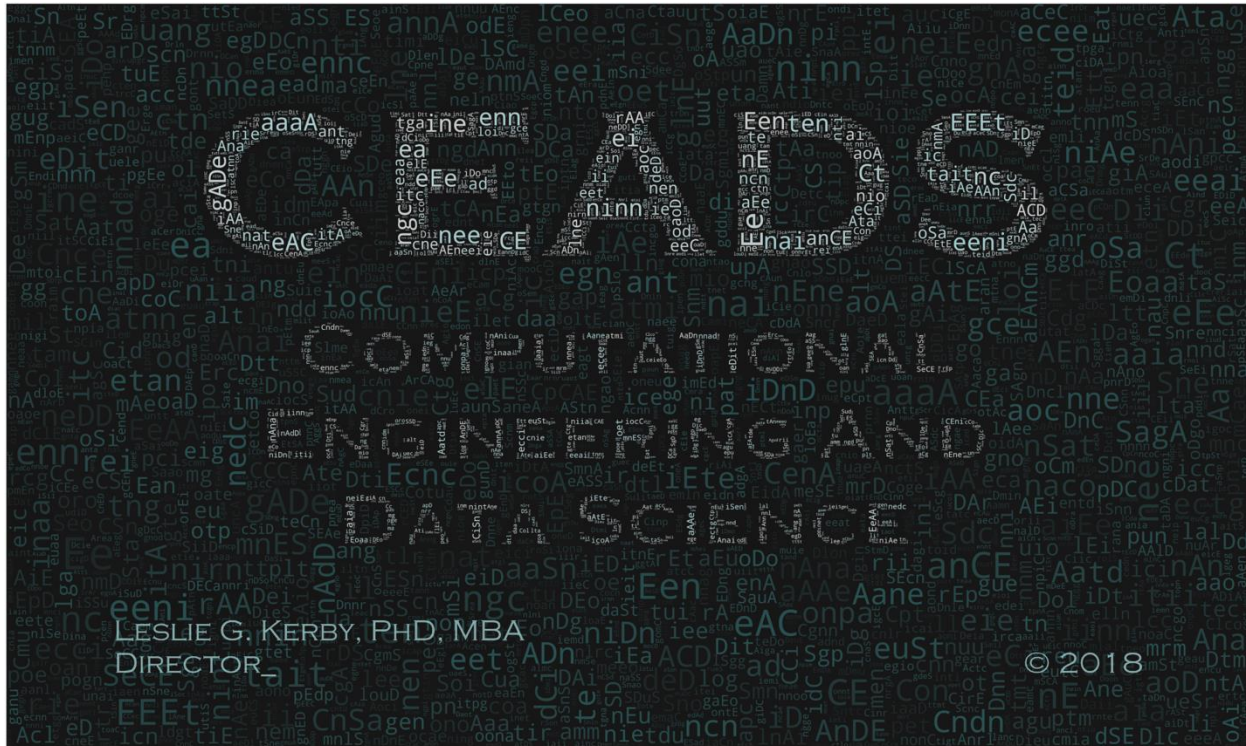
### 7.1 iFan Laboratory at Georgia Tech



The Intelligence for Advanced Nuclear (iFAN) lab was established in Fall 2021 by Dr. Fan Zhang when she joined Georgia Tech. The lab aims to advance research in cybersecurity, digital twins, predictive maintenance, and robotics applications to enhance the safety, security, efficiency, and economics of nuclear energy for a better future. iFAN lab has a diverse group of students with different backgrounds and cultures, including seven graduate students and several undergraduate students. The iFAN lab has received more than \$800,000 in external funding in nuclear cybersecurity, digital twins, predictive maintenance, and robotics. Dr. Fan Zhang is also actively engaged in IAEA nuclear cybersecurity missions and was invited to various IAEA meetings, seminars, and agency missions internationally. She is the recipient of the 2021 Ted Quinn Early Career Award from the American Nuclear Society (ANS) for her contribution in instrumentation and control, and cybersecurity. She received inaugural Distinguished Early Career Award from the U.S. DOE Office of Nuclear Energy in 2022 for her research in predictive maintenance and risk assessment.

In addition to the machine learning expertise iFAN lab has, the lab is equipped with three state-of-art experimental capabilities to facilitate cybersecurity, digital twins, and predictive maintenance research: a) a full-scope advance nuclear hardware-in-the-loop (HIL) testbed that contains a generic pressurized water reactor (GPWR) simulator, a programmable logic controller (PLC), and a GPWR digital twin hosted in Microsoft Azure environment; b) a five-layer control network and business network to emulate the network architecture of nuclear power plants; and c) a two-loop physical flow loop with heater, heat exchanger, pump, valves, and instrumentations to simulate a two-loop nuclear system. These capabilities provide a pathway to simulate normal operation, transient, cyber-attacks, anomalies, and operational events of light water reactors and advanced reactors.

## 7.2 The CEADS Lab at Idaho State University



The CEADS lab (Computational Engineering and Data Science), pronounced “seeds”, was established in December 2015 when Dr. Leslie Kerby joined Idaho State University. Since then, the lab has received over \$1 million in external funding, covering a range of projects from applied machine learning in nuclear cybersecurity, machine learning in Li-ion battery diagnostics, machine learning applications in nuclear pebble bed reactors, high-energy physics modeling in nuclear codes, to functional expansions in exascale computing. The group has produced over 30 peer-reviewed publications.

The CEADS lab provides core competencies in scientific machine learning and applied machine learning, as well as computational science and scientific computing. CEADS research is fundamentally interdisciplinary and thrives at the nexus of computer science, machine learning, engineering, and mathematics. Projects that necessitate problem-solving utilizing data-driven methods, Monte Carlo methods, or analytical methods, either individually or in combination, are our specialty.




The CEADS lab has graduated 2 PhD students and 3 MS students, as well as numerous BS students. The lab supports about half a dozen students currently. CEADS students have gone on to work at the Idaho National Laboratory, Oak Ridge National Laboratory, and ANSYS, among others. CEADS members have won best papers at several different ANS and other conferences, won a national laboratory graduate fellowship, and achieved first place in the IEEE COMPSAC Hackathon in 2018. Dr. Kerby is a requested panelist and speaker around the world.


## 7.3 Online Datasets and Research Repositories





### 7.3.1 iFan Repositories

#### 7.3.1.1 Asherah Models and Data


**Repository Address:** <https://gitlab.com/lambpati/asherah-training>

 **Asherah Training**   
Project ID: 33562944 

 Star 0



 50 Commits  1 Branch  0 Tags  601.3 MB Project Storage


Contains Asherah training data and associated files to use within machine learning algorithms.


main 


asherah-training


Find file


 


Clone 










 Upload New File  
Patience Lamb authored 2 months ago

03252765 

 README

 CI/CD configuration

 No license. All rights reserved

Name	Last commit	Last update
 Documentation	Upload New File	2 months ago
 Kevin Models	Upload New File	2 months ago
 PatienceModels	Accuracy of 1.0 and training loss of tiny num...	4 months ago
 .gitlab-ci.yml	Configure SAST in '.gitlab-ci.yml', creating th...	6 months ago
 90flow.csv	Upload New File	5 months ago
 99flow.csv	Upload New File	2 months ago
 AS1.0.100_1.csv	Upload New File	6 months ago
 AS1.1.80_1.csv	Upload New File	6 months ago
 AS1.1.90_1.csv	Upload New File	6 months ago

#### Documentation

*Presentations and reports created with the Asherah testbed in mind.*

**Link:** <https://gitlab.com/lambpati/asherah-training/-/tree/main/Documentation>

#### ACS Models (POC: Kevin Kelly)

*Models implemented into the autonomous Control System*

**Link:** <https://gitlab.com/lambpati/asherah-training/-/tree/main/Kevin Models>

#### Experimental Models (POC: Patience Lamb)

*Experimental Models. Data scattered throughout the upper-level directories. Each is otherwise a transient, steady state, or attack.*

**Link:** <https://gitlab.com/lambpati/asherah-training/-/tree/main/PatienceModels>



### 7.3.1.2 Cyber Threat Assessment INL-GT (GPWR Models and Data)

Repository Address: <https://gitlab.com/ifan-lab/cyber-threat-assessment-inl-gt>

The screenshot shows the GitLab interface for the repository 'Cyber Threat Assessment INL GT'. At the top, it displays the project name, ID (37833180), and a star count (0). Below this, it shows 56 commits, 3 branches, 0 tags, and 8.3 MB of project storage. A search bar is present with 'main' selected and 'cyber-threat-assessment-inl-gt' entered. A 'Find file' button and a 'Clone' button are also visible. A recent commit by 'Patience Lamb' is highlighted, titled 'Added Strategy Selection Model as well as updated AANN'. Below the commit list, there is a 'README' section with a 'No license' warning. A table lists the repository's structure with columns for 'Name', 'Last commit', and 'Last update'. The table shows three entries: 'data', 'models', and 'README.md'. Below the table, the 'README.md' content is displayed, starting with the title 'GPWR Autonomous Control System' and a description of the system's purpose.

Name	Last commit	Last update
data	Added Strategy Selection Model as well as u...	3 weeks ago
models	Added Strategy Selection Model as well as u...	3 weeks ago
README.md	Updated README to reflect project better	1 month ago

#### Data

*Data created by models.*

**Link:** <https://gitlab.com/ifan-lab/cyber-threat-assessment-inl-gt/-/tree/main/data>

#### Chris Project Data

*Data specifically pertaining to this project (data is mostly steam generator data).*

**Link:** <https://gitlab.com/ifan-lab/cyber-threat-assessment-inl-gt/-/tree/main/data/Chris-Project-Data>

#### 70 Variable Steam Generator Fault Sets

*Data of all 70 tags including transients, attacks, and steady-state operations.*

**Link:** <https://gitlab.com/ifan-lab/cyber-threat-assessment-inl-gt/-/tree/main/data/Chris-Project-Data/70VariableSgFaultSets>

#### Models

*Machine learning models for ACS implementation. Upper level includes non-autoML built models.*

**Link:** <https://gitlab.com/ifan-lab/cyber-threat-assessment-inl-gt/-/tree/main/models>

#### Azure AutoML Models

*Outputs from the Azure AutoML Models including regression and classification models.*

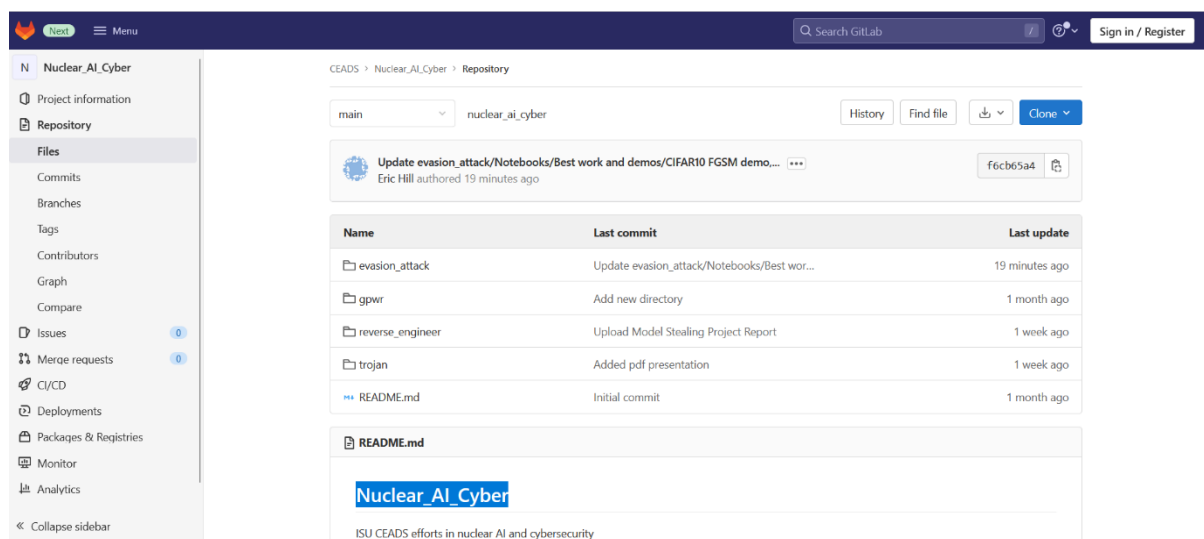
**Link:** [https://gitlab.com/ifan-lab/cyber-threat-assessment-inl-gt/-/tree/main/models/Azure\\_AutoML\\_Models](https://gitlab.com/ifan-lab/cyber-threat-assessment-inl-gt/-/tree/main/models/Azure_AutoML_Models)

## 7.3.2 CEADS Repositories

The CEADS lab has established a GitLab repository specifically for projects related to the application of machine learning in cyber security. The repository contains reports, demos, and other resources on the work the group has done in this field. Current resources available include studies done in performing inference attacks to reverse engineer machine learning models, attacking machine learning models with a Trojan, and attacking models using adversarial learning. The purpose of this repository is to allow others to explore the research done by CEADS in more detail. The link to the repository is given below. If any of the resources on the repository are used in other research, please include a citation to the repository in the work.

### 7.3.2.1 Parent Repository

Repository Address: [https://gitlab.com/CEADS/nuclear\\_ai\\_cyber](https://gitlab.com/CEADS/nuclear_ai_cyber)



#### **Evasion Attack** (POC: Eric Hill)

*Python Notebooks, Presentations, Data, and Documentation for ML Evasion Attack*

**Link:** [https://gitlab.com/CEADS/nuclear\\_ai\\_cyber/-/tree/main/evasion\\_attack](https://gitlab.com/CEADS/nuclear_ai_cyber/-/tree/main/evasion_attack)

#### **GPWR**

*GPWR Model Data*

**Link:** [https://gitlab.com/CEADS/nuclear\\_ai\\_cyber/-/tree/main/gpwr](https://gitlab.com/CEADS/nuclear_ai_cyber/-/tree/main/gpwr)

#### **Reverse Engineer** (POC: Emily Elzinga)

*Python Notebooks, Presentations, Data, and Documentation for ML Reverse Engineering Attack (Black Box, Grey Box, White Box)*

**Link:** [https://gitlab.com/CEADS/nuclear\\_ai\\_cyber/-/tree/main/reverse\\_engineer](https://gitlab.com/CEADS/nuclear_ai_cyber/-/tree/main/reverse_engineer)

#### **Trojan** (POC: Kallie McLaren)

*Python Notebooks, Presentations, Data, and Documentation for ML Trojan Attack*

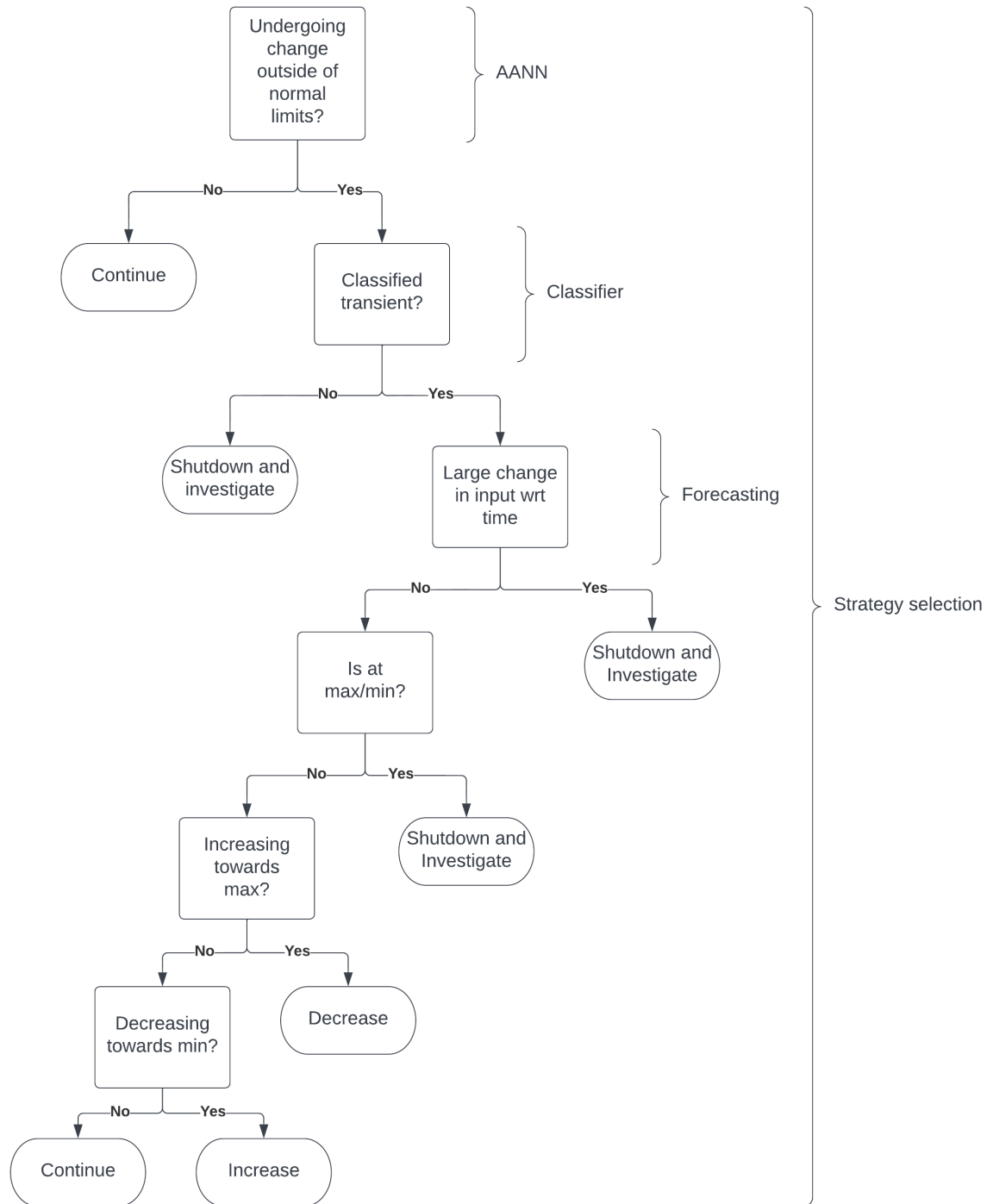
**Link:** [https://gitlab.com/CEADS/nuclear\\_ai\\_cyber/-/tree/main/trojan](https://gitlab.com/CEADS/nuclear_ai_cyber/-/tree/main/trojan)

*Page intentionally left blank*



## Annex I

### ACS Generalized Control Model (Strategy Selection)



## Annex II

### MITRE ATT&CK TTP References

The most informative way to access this information is via the MITRE ATT&CK website located at <https://attack.mitre.org>. We include this information as a reference to the TTPs that we used as part of the Scenarios in Section 4 in case the reader is using a printed copy and does not have access to the MITRE ATT&CK website.

Tactics ( <a href="https://attack.mitre.org/tactics/TAxxxx/">https://attack.mitre.org/tactics/TAxxxx/</a> )	
<b>TA0001</b>	<b>Initial Access</b> <i>The Adversary is trying to get into your network</i>
<b>TA0004</b>	<b>Privilege Escalation</b> <i>The adversary is trying to gain higher-level permissions.</i>
<b>TA0005</b>	<b>Defense Evasion</b> <i>The adversary is trying to avoid being detected.</i>
<b>TA0006</b>	<b>Credential Access</b> <i>The adversary is trying to steal account names and passwords.</i>
<b>TA0007</b>	<b>Discovery</b> <i>The adversary is trying to figure out your environment.</i>
<b>TA0040</b>	<b>Impact</b> <i>The adversary is trying to manipulate, interrupt, or destroy your systems and data.</i>
<b>TA0042</b>	<b>Resource Development</b> <i>The adversary is trying to establish resources they can use to support operations.</i>
<b>TA0043</b>	<b>Reconnaissance</b> <i>The adversary is trying to gather information they can use to plan future operations.</i>

Techniques ( <a href="https://attack.mitre.org/techniques/Txxxx/">https://attack.mitre.org/techniques/Txxxx/</a> )	
<b>T1078</b>	<b>Valid Accounts</b> <i>Adversaries may obtain and abuse credentials of existing accounts as a means of gaining Initial Access, Persistence, Privilege Escalation, or Defense Evasion.</i>
<b>T1083</b>	<b>File and Directory Discovery</b> <i>Adversaries may enumerate files and directories or may search in specific locations of a host or network share for certain information within a file system.</i>
<b>T1110</b>	<b>Brute Force</b> <i>Adversaries may use brute force techniques to gain access to accounts when passwords are unknown or when password hashes are obtained.</i>
<b>T1190</b>	<b>Exploit Public-Facing Application</b> <i>Adversaries may attempt to take advantage of a weakness in an Internet-facing computer or program using software, data, or commands in order to cause unintended or unanticipated behavior.</i>
<b>T1195</b>	<b>Supply Chain Compromise</b> <i>Adversaries may manipulate products or product delivery mechanisms prior to receipt by a final consumer for the purpose of data or system compromise.</i>
<b>T1199</b>	<b>Trusted Relationship</b> <i>Adversaries may breach or otherwise leverage organizations who have access to intended victims.</i>
<b>T1200</b>	<b>Hardware Additions</b> <i>Adversaries may introduce computer accessories, networking hardware, or other computing devices into a system or network that can be used as a vector to gain access.</i>
<b>T1498</b>	<b>Network Denial of Service</b> <i>Adversaries may perform Network Denial of Service (DoS) attacks to degrade or block the availability of targeted resources to users.</i>
<b>T1529</b>	<b>System Shutdown/Reboot</b> <i>Adversaries may shutdown/reboot systems to interrupt access to, or aid in the destruction of, those systems.</i>

<b>T1546</b>	<b>Event Triggered Execution</b> <i>Adversaries may establish persistence and/or elevate privileges using system mechanisms that trigger execution based on specific events.</i>
<b>T1564</b>	<b>Hide Artifacts</b> <i>Adversaries may attempt to hide artifacts associated with their behaviors to evade detection.</i>
<b>T1565</b>	<b>Data Manipulation</b> <i>Adversaries may insert, delete, or manipulate data in order to influence external outcomes or hide activity, thus threatening data integrity.</i>
<b>T1566</b>	<b>Phishing</b> <i>Adversaries may send phishing messages to gain access to victim systems.</i>
<b>T1583</b>	<b>Acquire Infrastructure</b> <i>Adversaries may buy, lease, or rent infrastructure that can be used during targeting.</i>
<b>T1584</b>	<b>Compromise Infrastructure</b> <i>Adversaries may compromise third-party infrastructure that can be used during targeting.</i>
<b>T1588</b>	<b>Obtain Capabilities</b> <i>Adversaries may buy and/or steal capabilities that can be used during targeting.</i>
<b>T1589</b>	<b>Gather Victim Identity Information</b> <i>Adversaries may gather information about the victim's identity that can be used during targeting.</i>
<b>T1590</b>	<b>Gather Victim Network Information</b> <i>Adversaries may gather information about the victim's networks that can be used during targeting.</i>
<b>T1591</b>	<b>Gather Victim Org Information</b> <i>Adversaries may gather information about the victim's organization that can be used during targeting.</i>
<b>T1592</b>	<b>Gather Victim Host Information</b> <i>Adversaries may gather information about the victim's hosts that can be used during targeting.</i>
<b>T1593</b>	<b>Search Open Websites/Domains</b> <i>Adversaries may search freely available websites and/or domains for information about victims that can be used during targeting.</i>
<b>T1597</b>	<b>Search Closed Sources</b> <i>Adversaries may search &amp; gather information about victims from closed sources for targeting.</i>

<b>Mitigations</b> ( <a href="https://attack.mitre.org/mitigations/Mxxxx/">https://attack.mitre.org/mitigations/Mxxxx/</a> )	
<b>M1013</b>	<b>Application Developer Guidance</b> <i>This mitigation describes any guidance or training given to developers of applications to avoid introducing security weaknesses that an adversary may be able to take advantage of.</i>
<b>M1016</b>	<b>Vulnerability Scanning</b> <i>Vulnerability scanning is used to find potentially exploitable software vulnerabilities.</i>
<b>M1029</b>	<b>Remote Data Storage</b> <i>Use remote security log and sensitive file storage where access can be controlled better to prevent exposure of intrusion detection log data or sensitive information.</i>
<b>M1035</b>	<b>Limit Access to Resource Over Network</b> <i>Prevent access to file shares, remote access to systems, unnecessary services.</i>
<b>M1036</b>	<b>Account Use Policies</b> <i>Configure features related to account use like login attempt lockouts, specific login times, etc.</i>
<b>M1037</b>	<b>Filter Network Traffic</b> <i>Use network appliances to filter ingress or egress traffic and perform protocol-based filtering.</i>
<b>M1050</b>	<b>Exploit Protection</b> <i>Use capabilities to detect &amp; block conditions that may lead to or indicate a software exploit.</i>
<b>M1052</b>	<b>User Account Control (UAC)</b> <i>Configure Windows UAC to mitigate risk of adversaries obtaining elevated process access.</i>
<b>M1053</b>	<b>Data Backup</b> <i>Take and store data backups from end user systems and critical servers.</i>
<b>M1056</b>	<b>Pre-Compromise</b> <i>This category is used for any applicable mitigation activities that apply to techniques occurring before an adversary gains Initial Access, such as Reconnaissance and Resource Development techniques.</i>

## Annex III

### Essential Reading

The Endnotes section of this document includes a good combination of references to papers and books that were referenced throughout this report. This Annex provides a list of Essential Reading in the field of Adversarial Machine Learning. The five papers that we choose are a combination of the most cited papers on the topic of Adversarial Machine Learning by Google Scholar and ones which have informed the primary researchers in this field. The five books we selected are a sampling and example of how this field of research is expanding offering some hints of where we may be headed.

#### Adversarial Machine Learning Papers

1. **Adversarial machine learning at scale**

This paper was published in 2016 by Alexey Kurakin (Google Brain), Ian Goodfellow (OpenAI), and Samy Bengio (Google Brain) and describes their application of adversarial training to some larger problem spaces. Their findings include how to scale adversarial training to large models and datasets; an assertion that multi-step attack methods are less transferable than single-step methods (thus best for black-box attacks); and creation of a *label-leaking* effect to improve the performance of adversarial trained models on adversarial examples.

*Kurakin, Alexey, Ian Goodfellow, and Samy Bengio. "Adversarial machine learning at scale." arXiv preprint arXiv:1611.01236 (2016).*

2. **ImageNet Large Scale Visual Recognition Challenge**

While this paper is not specifically on Adversarial Machine Learning, it is the most cited article across Adversarial Machine Learning papers. Co-published by Olga Russakovsky (Stanford) and Jia Deng (University of Michigan) the authors describe the creation of the benchmark dataset and the advancements in the field of object recognition. This paper is essential for us to understand as it describes the challenges of collecting large-scale ground-truth annotations and provides a section on *The Future* with five predictions of what was to come in 2014. An Open Source collection of data is available at the ILSVRC site that hosted yearly image classification competitions from 2010 – 2017.

*Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang et al. "ImageNet large scale visual recognition challenge (2014)." arXiv preprint arXiv:1409.0575 2, no. 3 (2014).*

3. **Adversarial machine learning**

This paper published in 2011 is the second most cited paper on Google Scholar. Authored by Ling Huang (Intel Labs Berkeley), Anthony D. Joseph (UC Berkeley), Blaine Nelson (University of Tübingen), Benjamin Rubenstein (Microsoft), and J Doug Tygar (UC Berkeley), they provide some of the first thoughts about the study of effective machine learning techniques against adversarial opponents. They introduce a taxonomy for classifying attacks against machine learning algorithms and discuss factors that limit adversarial actions, many of which appear in this report as well. This report references a 2006 paper on *Can machine learning be secure?* which is not included in our top five but when tracing the field backwards to understand how current work was derived and informed by past research efforts, this paper is a common endpoint. *Huang, Ling, Anthony D. Joseph, Blaine Nelson, Benjamin IP Rubinstein, and J. Doug Tygar. "Adversarial machine learning." In Proceedings of the 4th ACM workshop on Security and artificial intelligence, pp. 43-58. 2011.*

4. **Wild patterns: Ten years after the rise of adversarial machine learning**

This paper was published in 2018 which largely sets the birth of this field in public literature around 2008. Authored by Battista Biggio and Fabio Roli from the University of Cagliari in Italy, they provide a detailed view of the evolution of adversarial machine learning noting which branches have been more widely embraced than others (such as the focus on deep learning algorithms). It is one of the first published articles to combine the fields adversarial machine learning with cyber-security which were always adjacent by intent but in this paper the machine learning implementations are set into the context of how they would affect the cyber-security of a given system and what countermeasures would be helpful in handling this type of risk.

*Biggio, Battista, and Fabio Roli. "Wild patterns: Ten years after the rise of adversarial machine learning." Pattern Recognition 84 (2018): 317-331.*

5. **A Survey of Privacy Attacks in Machine Learning**

This paper was published in 2020 and as expected, over a decade of work in this field yields a good number of survey papers where published research is reviewed and summarized. Perhaps the appearance of aggregate surveys in a field is a good indicator of field maturity. This paper was authored by Maria Rigaki and Sebastian Garcia from Czech Technical University in Prague and reviews 40 papers related to privacy attacks against machine learning. Perhaps more helpful is the section on commonly proposed defenses and their predictions of open problems and future directions in our field.

*Rigaki, Maria, and Sebastian Garcia. "A survey of privacy attacks in machine learning." arXiv preprint arXiv:2007.07646 (2020).*

## **Adversarial Machine Learning Books**

1. **Adversarial Robustness for Machine Learning**

This book was just released in September of 2022. Authored by Pin-Yu Chen (IBM Watson Research Center) and Cho-Jui Hsieh (UCLA) they offer a comprehensive summary of the field of adversarial robustness for machine learning. Their angle of attack is to assert that, very much similar to what we are experiencing in nuclear, that the application of machine learning to real world problems such as self-driving cars, robotic controls, and healthcare systems, lacks a robustness such that the use of these capabilities may not scale with the application itself. One of the concerns we have in security engineering is system or function availability and thus this book is a timely contribution to our field of study.

*Published by Elsevier.*

*<https://www.elsevier.com/books/adversarial-robustness-for-machine-learning/chen/978-0-12-824020-5>*

2. **Strengthening Deep Neural Networks**

Tag lines go a long way in selling us on whether we should read a book or paper. This book by Katy Warr (Roke Manor Research) offers up the goal to *Make AI Less Susceptible to Adversarial Trickery*. With a focus on deep neural networks (DNNs) Katy describes how to generate adversarial inputs capable of fooling DNNs, offers real-world scenarios with adversarial threat models, and evaluates options for improving neural network robustness. Bonus points for any author who includes a GitHub repository full of code to play with and immerse yourself in the learning process. (<https://github.com/katywarr/strengthening-dnns>).

*Published by O'Reilly.*

*<https://www.oreilly.com/library/view/strengthening-deep-neural/9781492044949/>*

### 3. Adversarial Deep Learning in Cybersecurity

This book is scheduled for release in October of 2022. Authored by Aneesh Sreevallabh Chivukula, Xinghao Yang, and Wei Liu (Advanced Analytics Institute), Wanlei Zhou (Centre for Cyber Security and Privacy), Bo Liu (School of Computer Science) at the University of Technology in Sydney, their book includes background chapters on Adversarial Machine Learning and Deep Learning as well as Game-Theoretical attacks with adversarial deep learning models and Physical Attacks in the Real World. While enumeration of the attack classes is always exciting, their recommendations on defensive countermeasures will be helpful to dive into.

*Published by Springer. Available on Barnes & Noble.*

<https://www.barnesandnoble.com/w/adversarial-deep-learning-in-cybersecurity-aneesh-sreevallabh-chivukula/1141093103>

### 4. Artificial Intelligence for High Energy Physics

This book published in March of 2022 by Paolo Calafura (Lawrence Berkeley National Laboratory), David Rousseau (Universite Paris-Saclay), and Kazuhiro Terao (SLAC National Accelerator Laboratory) discusses the use of artificial intelligence and machine learning and their application to high-energy physics. As security researchers we are interested in their analysis of how deep neural networks are implemented with their physical systems and how their approaches to anomaly detection function such that we can assess applicability to our problem space.

*Published by World Scientific Publishing Company. Available on Amazon.*

<https://www.amazon.com/Artificial-Intelligence-High-Energy-Physics/dp/9811234027>

### 5. The Visual Display of Quantitative Information

This book written by Edward Tufte (Yale) is a 2<sup>nd</sup> Edition masterpiece on the theory and practice in the design of data graphics. How information is displayed drives how it will be consumed. This book provides examples of transforming quantitative information to infographics, conveying to the viewer more than just statistics, but a deeper message and narrative.

*Published by Edward Tufte*

[https://www.edwardtufte.com/tufte/books\\_vdqi](https://www.edwardtufte.com/tufte/books_vdqi)

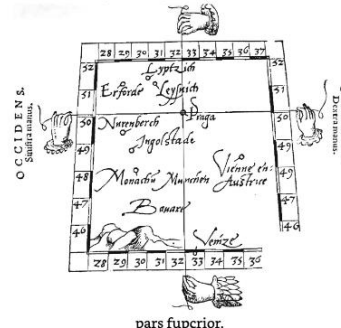


Ecce formulam, v

um, atque

ftru&uram Tabufarum Prolomxi, cum  
quibufdam locis, in quibus fudiosus  
Geographix fe faris exercere porel1.

SEPT ENT R I O.



*Page intentionally left blank*

## Endnotes

---

- [1] Deutch, Howard, director. *Some Kind of Wonderful*. 1987. Paramount Pictures. 93 minutes.
- [2] Brownson, D. A., C. A. Dobbe, and D. L. Knudson. *PWR depressurization analyses*. No. EGG-M-92501; CONF-9210204-5. EG and G Idaho, Inc., Idaho Falls, ID (United States), 1992.
- [3] "Pressurized Water Reactor (PWR) Systems." Nuclear Regulatory Commission, n.d.
- [4] Volkanovski, Andrija, Antonio Ballesteros Avila, and Miguel Peinador Veira. "Statistical analysis of loss of offsite power events." *Science and Technology of Nuclear Installations* 2016 (2016).
- [5] Ivanov, Konstandin N., Tara M. Beam, Anthony J. Baratta, Adi Irani, and Nick Trikouros. "Pressurised water reactor Main Steam Line Break (MSLB) benchmark. V. 1. Final specifications." (1999).
- [6] Pedro Mena, R.A. Borrelli, and Leslie Kerby. Expanded analysis of machine learning models for nuclear transient identification using tpot. *Nuclear Engineering and Design*, 390:111694, 2022.
- [7] Spirito, Chris, Fan Zhang, Sukesh Aghara, Collin Duffley, Joel Strandburg, and Jamie Coble. "Cyber Attack and Defense Use Cases for Autonomous and Remote Operations for Advanced Reactors." Idaho Falls: Idaho National Lab, August 2021.
- [8] Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel et al. "Scikit-learn: Machine learning in Python." *the Journal of machine Learning research* 12 (2011): 2825-2830.
- [9] Chollet, François. "Xception: Deep learning with depthwise separable convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251-1258. 2017.
- [10] Biggio, Battista, and Fabio Roli. "Wild patterns: Ten years after the rise of adversarial machine learning." *Pattern Recognition* 84 (2018): 317-331
- [11] Dalvi, Nilesh, Pedro Domingos, Sumit Sanghai, and Deepak Verma. "Adversarial classification." In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 99-108. 2004.
- [12] Squillero, Giovanni, and Paolo Burelli, eds. *Applications of evolutionary computation*. Springer, 2016.
- [13] LeDell, Erin, and Sebastien Poirier. "H2o automl: Scalable automatic machine learning." In *Proceedings of the AutoML Workshop at ICML*, vol. 2020. 2020.
- [14] Jin, Haifeng, Qingquan Song, and Xia Hu. "Auto-keras: An efficient neural architecture search system." In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1946-1956. 2019.
- [15] O'Malley, Tom, Elie Bursztein, James Long, François Chollet, Haifeng Jin, and Luca Invernizzi. "KerasTuner." GitHub, 2019. <https://github.com/keras-team/keras-tuner>.
- [16] Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel et al. "Scikit-learn: Machine learning in Python." *the Journal of machine Learning research* 12 (2011): 2825-2830.
- [17] Le, Trang T., Weixuan Fu, and Jason H. Moore. "Scaling tree-based automated machine learning to biomedical big data with a feature set selector." *Bioinformatics* 36, no. 1 (2020): 250-256.



- 
- [18] Giacomazzo, Bernadette. "Inside the True Story Behind the Legendary Trojan Horse." All That's Interesting. All That's Interesting, March 16, 2022. <https://allthatsinteresting.com/trojan-horse>.
- [19] Liu, Yingqi, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. "Trojaning attack on neural networks." (2017).
- [20] Zhang, Fan, Tanya Sharma, Patience Lamb, and Avery Skolnick. "Cyber Threat Assessment INL GT." GitLab, 2022. <https://gitlab.com/ifan-lab/cyber-threat-assessment-inl-gt>.
- [21] Mahajan, Praateek. "TrojanNN - Pytorch." GitHub, December 10, 2018. <https://github.com/praateekmahajan/TrojanNN-Pytorch/blob/master/final.ipynb>.
- [22] Liu, Yuntao, Yang Xie, and Ankur Srivastava. "Neural trojans." In *2017 IEEE International Conference on Computer Design (ICCD)*, pp. 45-48. IEEE, 2017.
- [23] Zhang, Fan, Tanya Sharma, Patience Lamb, and Avery Skolnick. "Asherah Training." GitLab, 2022. <https://gitlab.com/lambpati/asherah-training>.
- [24] Ilyas, Andrew, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. "Adversarial examples are not bugs, they are features." *Advances in neural information processing systems* 32 (2019).
- [25] Word Art, 2022. <https://wordart.com/create>.
- [26] Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." arXiv preprint arXiv:1412.6572 (2014).
- [27] Molnar, Christoph. Interpretable machine learning. Lulu. com, 2020.
- [28] "Adversarial Example Using FGSM: Tensorflow Core." TensorFlow, January 26, 2022. [https://www.tensorflow.org/tutorials/generative/adversarial\\_fgsm](https://www.tensorflow.org/tutorials/generative/adversarial_fgsm).
- [29] Rosebrock, Adrian. "Adversarial Attacks with FGSM (Fast Gradient Sign Method)." PyImageSearch, March 1, 2021. <https://pyimagesearch.com/2021/03/01/adversarial-attacks-with-fgsm-fast-gradient-sign-method/>.
- [30] Geng, Daniel. "Dangeng/simple\_adversarial\_examples: Repo of Simple Adversarial Examples on Vanilla Neural Networks Trained on Mnist." GitHub, January 10, 2018. [https://github.com/dangeng/Simple\\_Adversarial\\_Examples](https://github.com/dangeng/Simple_Adversarial_Examples).
- [31] "Keras Documentation: Xception." Keras. Accessed August 15, 2022. <https://keras.io/api/applications/xception/>.
- [32] Athalye, Anish, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. "Synthesizing robust adversarial examples." In *International conference on machine learning*, pp. 284-293. PMLR, 2018.
- [33] Polyakov, Alex. "How to Attack Machine Learning (Evasion, Poisoning, Inference, Trojans, Backdoors)." Medium. Towards Data Science, 2021. <https://towardsdatascience.com/how-to-attack-machine-learning-evasion-poisoning-inference-trojans-backdoors-a7cb5832595c>.
- [34] Guo, Ying, Xingxing Wei, Guoqiu Wang, and Bo Zhang. "Meaningful adversarial stickers for face recognition in physical world." *arXiv preprint arXiv:2104.06728* (2021).

- 
- [35] Myers, P. "The neglected aspect of computer security." Dissertaç ao de Mestrado, Naval Postgraduate School, Monterey. URL <http://seclab.cs.ucdavis.edu/projects/history/CD/myer80.pdf> (1980).
- [36] IBM Research. The thrill of cyber threat hunting with Kestrel Threat Hunting Language. <https://research.ibm.com/blog/kestrel-cyber-threat-hunting>.
- [37] Office of the Director of National Intelligence (ODNI). A Common Cyber Threat Framework: A Foundation for Communication. [https://www.dni.gov/files/ODNI/documents/features/ODNI\\_Cyber\\_Threat\\_Framework\\_Overview.\\_UNCL.\\_20180718.pdf](https://www.dni.gov/files/ODNI/documents/features/ODNI_Cyber_Threat_Framework_Overview._UNCL._20180718.pdf) (2018).
- [38] Humphries, Matthew. "Machine Learning Is Now Being Used to Cheat in Multiplayer Games." PCMAG, July 8, 2021. <https://www.pcmag.com/news/machine-learning-is-now-being-used-to-cheat-in-multiplayer-games>.
- [39] Zimmerman, Carson. "Cybersecurity operations center." The MITRE Corporation (2014).
- [40] Han Xiao, Huang Xiao, Claudia Eckert. Adversarial Label Flips Attack on Support Vector Machines. Frontiers in Artificial Intelligence and Applications. Volume 242: ECAI 2012.