INL/CON-23-70988-Revision-2

Language-Theoretic Data Collection to Support ICS Protocol Baselining

May 2023

Gabriel Arthur Weaver, Dan Gunter

Changing the World's Energy Future

INL is a U.S. Department of Energy National Laboratory operated by Battelle Energy Alliance, LLC

INL/CON-23-70988-Revision-2

Language-Theoretic Data Collection to Support ICS Protocol Baselining

Gabriel Arthur Weaver, Dan Gunter

May 2023

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

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

Language-Theoretic Data Analysis to Support ICS Protocol Baselining

Gabriel A. Weaver
D520
Idaho National Laboratory
Idaho Falls, USA
gabriel.weaver@inl.gov

Dan Gunter
Insane Forensics
San Antonio, USA
dan@insaneforensics.com

***Abstract*—Critical infrastructure stakeholders need to baseline their systems to understand expected protocol communications. Baseline behaviors may vary based on operational context. Expected operations during a maintenance window, for example, may be different from normal operations. Furthermore, constructing system baselines for Industrial Control Systems (ICS) is difficult and time-consuming. ICS processes generate artifacts expressed across heterogeneous data sources such as network traffic and device logs. This paper explores the hypothesis that such ICS artifacts form a language in the language-theoretic sense. From a theoretical perspective, the variety of implementations of ICS protocols and constrained environment of OT networks provide a rich application domain for language-theoretic approaches. We present several use cases related to the practical construction of system baselines: grammars for data fusion, language dialects for device fingerprinting, and security automata for system baselining.**

I. Introduction

Processes implemented within *Industrial Control Systems (ICS)* generate artifacts expressed across heterogeneous data sources such as network traffic and device logs. This paper explores the hypothesis that these process artifacts form a language in the language-theoretic sense. The ability to align and analyze operational artifacts relative to a formal language specification would practically benefit asset owners and operators. Research and development of such a capability provides opportunities to adapt and extend language-theoretic approaches to construct system baselines. *System baselines* integrate network traffic, device logs, and other artifacts to define a notion of 'normal' behavior within a specific operational environment. In addition, system baselines support the ability to detect anomalous, potentially adversarial, behaviors.

ICS provides a rich domain to explore applications of principles from *language-theoretic security (LangSec)*. The expected behavior within facility-level systems is more tightly constrained than within *Information Technology (IT)* systems. As a result, we hypothesize that *Operational Technology (OT)* environments—built to support specific processes—are more conducive to constructing system baselines than general-purpose enterprise networks. In addition, many industrial protocols are specified or implemented in terms of automata with states distributed across networked devices and this includes the *Distributed Network Protocol 3 (DNP3)* and *Intenational Electrotechnical Commission (IEC)* 104 protocol. This practice suggests a natural alignment between the representation of the behavior of networked devices and automata as well as the potential to constrain or adapt specified automata for a particular facility.

A language-theoretic approach has the potential to provide several benefits to asset owners and operators that improve upon current practice in system data curation and fusion. A formal language provides a natural way to normalize expressions of the same notional event expressed across different security artifacts. Although the application-layer of the network stack intuitively seems a rich source of high-level semantics, practitioners tend to construct *network baselines*—baselines derived solely from network traffic—from features taken from the lower layers of the network protocol stack (e.g. source IP address, destination IP address, source port, destination port, and protocol). Heuristics expressed as rules constructed using these fields are often integrated within a *Security Information and Event Management (SIEM)* platform. In contrast, constructing a language model (e.g. a grammar) that could recognize operational events would allow one to integrate syntactic features of process artifacts into expressions of high-level events, potentially in a protocol-independent manner. An approach to data fusion that normalizes process artifacts relative to high-level semantics also could help mitigate against an informational single point of failure for visibility into and control over operations.

Furthermore, a language-theoretic approach could be used for device fingerprinting and to construct system baselines as automata and potentially even enforceable security policies [1]. With respect to device fingerprinting, ICS protocols are notorious for being poorly implemented relative to their standard. For example, a recent SANS ICS presentation by Brizinov demonstrated how differences in IEC 61850 protocol communications can identify specific vendor product lines [2]. Moreover, adversaries may implement industrial protocols

Research was sponsored by the *Cybersecurity for the Operational Technology Environment (CyOTE)* program and was accomplished under the direction of the *Department of Energy (DOE) Cybersecurity, Energy Security, and Emergency Response (CESER)*. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

poorly. Gunter et al. demonstrated the ability to detect and profile the Industroyer/CrashOverride malware by gathering state misses for network traffic relative to the state machine within the IEC 104 protocol specification [3]. ICS systems provide an opportunity to study different dialects of a protocol and to evaluate how consistent this behavior is across different platforms and varying operational contexts. Moreover, since ICS systems are deployed within an operational environment, measures of physical events (including side channels) could, in theory, be integrated into a notion of device state to constrain system behavior based on a *specific environment* rather than protocol or device capabilities *in general*.

a) This Paper: As mentioned above, this paper explores the hypothesis and practical benefits of viewing artifacts generated by ICS processes as a language. Section II describes current industry practice for system baselining ICS systems and surveys language-theoretic approaches in the literature to address perceived gaps. Section III provides background on language-theoretic concepts applied to use cases in Section IV. Finally Section V concludes.

II. Related Work

Much of system baselining, from an asset owner perspective, focuses on network traffic analysis. As noted by Hanna [4], much of the attack surface resides in the network and this is consistent with the LangSec view which emphasizes the need to "identify, analyze, and improve" communications boundaries within legacy systems [5]. In this section, we consider the state of the art relative to identified gaps in practice for ICS data fusion, device fingerprinting, and system baselining.

A. Data Fusion

The need to fuse network and device data is an acknowledged industry gap. A recent presentation by researchers from the Salt River Project and Dragos Inc. leveraged OSISoft PI Historian event frames to enrich security data [6]. In the presentation, the researchers discussed potential ways to incorporate operational events with network security events through data normalization and correlation.

Our ongoing research considers languages over security artifacts—defined by grammars—as an approach to normalize, fuse, and analyze disparate data sources [7]. Previous work in the literature aligns with this approach. Eads et al. used grammar-guided feature extraction as a mechanism to integrate domain knowledge for time series classification [8]. McFail recently presented material on the potential to harmonize data fields across the IEC 104 and DNP3 protocols [9]. Finally, Hanna further mentions the need to group actions for a specific behavior or impact into procedures within ICS systems [4]. In Section III, we discuss a way to formalize this alignment via grammars.

Structured formats that have been used previously to align and integrate data (and which may inform the design of such languages) include traces. Traces are sequences of events, ordered in time, that reflect either the execution of software on a device (execution trace) or communications sent by a device on a network (network trace). Traces are an abstraction that bridges the gap between applied disciplines, where they are used in protocol reverse engineering [10], as well as the theoretical. For example, properties of traces (e.g. Lamport's safety/liveness properties [11]) and sets of traces (hyperproperties [12]) have been used to specify and evaluate whether a system is secure.

B. Device Fingerprinting

Traditional security baselining in the IT and industrial space relies on the use of the networking five tuple. The standard components of the five tuple include source IP address, destination IP address, source port, destination port, and protocol. Statistical models are applied to these five values and the time and frequency of 5-tuple occurrences are used to detect anomalies. While this approach works on a fundamental level and summarizes communciations based on the existence of the message, it lacks any context as to what the communication included. Although traditional statistical approaches are widely employed in industry, recent work on cyber baselining by Schulz et al. suggest that low-level observables lack properties upon which traditional statistical tools depend [13].

Our research considers features selected for device fingerprinting as part of a tradeoff between constructing such fingerprints and vulnerabilities resulting from differences among mutually intelligible ICS protocol languages. Sassaman et al. note that parse tree differential analysis may enhance fingerprinting-based attacks [5] and a related approach is recently attested to within the ICS space. Although the Modbus specification includes function code 43 to read device identification information, several vendors choose not to implement this function. Furthermore, in a recent presentation at SANS ICS, Brizinov outlines how minor differences in the IEC 61850 protocol communications can identify specific product lines of Schneider Electric PLCs [2]. This approach mirrors similar software and device fingerprinting from the IT field except it uses ICS-specific protocol values and variables. Previous IT fingerprinting approaches leverages specific time-to-live values in network traffic and known default values in protocols to identify devices.

C. System Baselining

Language-theoretic approaches to system baselining within ICS would provide asset owners the ability to construct baselines for specific processes as automata. This approach builds on previous work by Schneider [1], who identified the ability to represent security policy constraints on execution traces as automata. In fact researchers have noted while discussing the problems with *Intrusion Detection Systems (IDS)* in general, the promise of approaches such as security automata for application-specific security policies [5].

Our research seeks to extend and adapt automata-based security policies to practically construct ICS system baselines of whitelisted behavior from network and device logs. FSM inference may provide a process to practically construct such

automata from observed behavior. Within the literature, there are two types of FSM inference algorithms: active and passive. Active FSM inference algorithms iteratively query a server to exhaustively learn all protocol states. In contrast, passive FSM infer automata from a set of traces. Both approaches have been applied successfully to computer security. For example, active FSM inference has been applied to infer botnet *command and control (c2)* protocols [14], fuzzing test generation for IoT network protocols [15], and compare multiple implementations of the *Datagram Transport Layer Security (DTLS)* protocol [16]. In general, finding a minimized FSM that contains all traces and only those traces makes passive FSM inference NP hard [17]. Nonetheless, several papers have been published to demonstrate the approach's practical utility including protocol security flaw detection [17], honeypot behavior [18], and modeling microservices via Kubernetes netflow data [19].

III. Theory

This section provides background from language-theory. After providing a brief, general introduction to language-theory, we discuss specific aspects of the theory that we apply to Data Fusion, Device Fingerprinting, and System Baselining in Section IV.

From the perspective of language theory, a *language* is a set whose elements are defined over some fixed alphabet, a non-empty set of symbols. *Recognizers* are computational machines that accept or reject a string as being an element of a language. Language theory classifies languages into different classes depending upon the type of computational engine required to recognize the language and this is known as the Chomsky hierarchy.

Although the Chomsky hierarchy defines many classes of languages, we focus on regular and context-free due to their relevance to applications within the ICS domain. A more comprehensive discussion of the Chomsky hierarchy from a language-theoretic security perspective may be found in [5]. The class of *regular languages* consists of languages whose strings can be recognized by some finite automaton, or equivalently, a regular expression.

The next most complex language class in the hierarchy are context-free languages. A *context-free language* is a set of strings that can be recognized by a finite automaton with a stack. In practice, this means that one can write a context-free grammar to recognize a string in a context-free language. These languages possess a recursive or hierarchical structure and are called context free because their elements are generated by substituting strings for variables called *non-terminals* regardless of the context in which they occur [20]. There are more complex language classes still, for example, a grammar is at least context sensitive if the structure of that string is influenced by a value in another part of the string [21].

A. Grammars

As discussed previously, grammars are a formalism that we can employ to align and normalize security data across different artifacts generated by ICS processes. We focus on deterministic context-free grammars or weaker as a tool to analyze such security artifacts and previous work has demonstrated this as well within a non-ICS context [7].

A context-free grammar G consists of four components—terminals, nonterminals, a start symbol, and productions. *Terminals* are the basic symbols from which strings are formed. *Nonterminals* are syntactic variables that denote sets of strings and one of these nonterminals is called the *start symbol*. *Productions* of a grammar specify rewrite rules that transform a nonterminal into a string of nonterminals and terminals. The language of a grammar G is denoted $L(G)$.

Deterministic context-free grammars (or weaker) also allow the theoretical possibility to evaluate both the equivalence of two different dialects of the same language as well as whether a language is a sublanguage of another [5]. *Mutually-intelligible dialects* consist of two grammars G and H that implement the 'same' language.

B. Security Automata

Enforceable security policies introduced by Schneider, define a language over execution traces to enforce constraints aligned with security goals such as memory accesses or file writes [1]. Specifically, he considers policies based on observing steps of an execution and where constraints are *properties*, constraints that can be evaluated for each element of the trace sequence. Automata that recognize languages of traces relative to such properties belong to the class of *Execution Monitoring (EM)* policies. Such policies have Lamport's safety property, in which traces not in the language of whitelisted traces, have a finite set of bad prefixes.

In practice, stakeholders need a process by which to define security automata to enforce such languges. We want to explore whether traces derived from network and device logs can be used to define security automata based on whitelisted communications associated with specific operational events. To reframe an example provided in [5], given a production that describes an ICS protocol feature *in general* ($A ::= B \mid C$), but the only observed branch *in that operational environment* is $A \Longrightarrow C$, then we can construct a security automaton for the observed whitelisted behavior ($A := C$) which we expect, as a subset of the general specification, has the potential to reduce language complexity.

Furthermore, FSM inference algorithms may provide a practical approach to construct security automata for system baselining. Given an input language of network and device traces—normalized by a grammar—it may be possible to learn an automaton using passive or active state machine inference algorithms as described in Section II. If successful, this would provide algorithms to help stakeholders construct security automata as a system baselining tool. Such automata would be relevant within specific operational environments and if no more powerful than context-free deterministic, would support evaluating whether sublanguages of communication protocols used in practice are equivalent. Sublanguages of industry protocols expressed within ICS processes which could not be expressed weakly enough to enable an evaluation of

equivalence would be known by asset owners as potential weak spots for vulnerabilities and candidates for redesign.

IV. Use Cases

A fundamental purpose of ICS networks is to support *Supervisory Control and Data Acquisition (SCADA)* of infrastructure assets across broad geographic regions and diverse types of industrial processes. The operation of such systems, by definition, depends upon secure communication boundaries. Given the diversity of devices and protocol implementations within ICS, being able to evaluate equivalence or differences of mutually-intelligible dialects is important. In this section, we provide several use cases for language-theoretic approaches to address gaps in current practice with respect to data fusion, device fingerprinting, and system baselining.

A. Grammars for Data Fusion

The ability to construct a language to represent the behaviors of individual industrial devices enables additional layers of analysis for industrial asset owners. Grammar productions can be used to represent expressions of higher-level procedures (e.g. 'breaker manipulation') within security artifacts. A simple example from bulk electric power involes normalizing device state changes across different protocols and devices. The process to open a breaker on one device might be implemented using either the DNP3 protocol as a DIRECT OPERATE command or via two consecutive commands, SELECT then OPERATE.

Fusing both expressions of an ICS procedure across commands via a nonterminal symbol may also enable an asset owner to filter on or build composite commands while abstracting away complexities of specific devices. This includes the potential to normalize behavior across multiple ICS protocols by adding productions for a given nonterminal symbolizing a procedure. For example, some industrial devices use sequences of Modbus read and write commands to specific register values to read and manipulate the state of the breaker. Such a language could recognize the sequence of Modbus commands as an expression of the higher-level 'breaker manipulation' concept.

B. Language Dialects for Device Fingerprinting

Industrial device fingerprints derive from both network and device state information. Network protocols upon which industrial devices rely include proprietary and open communication protocols. A variety of different standards organizations (e.g. IEEE, IEC) as well as controlling organizations often publish common specifications to implement within a device. Despite a common specification, vendors and adversaries implement ICS protocols in a variety of different ways and these differences in mutually-intelligible dialects can provide features for device fingerprinting.

Algorithms to find differences among mutually-intelligible dialects, such as parse tree differential analyses, may be viewed as feature selection algorithms to define classifiers for device fingerprinting. For example, although the Modbus specification includes function code 43 to read device identification information, several vendors choose not to implement this function. As mentioned earlier, a recent SANS ICS presentation by Brizinov demonstrate how minor differences in IEC 61850 protocol communications can identify specific product lines of Schneider Electric PLCs [2].

Just as industrial *Original Equipment Manufacturers (OEMs)* implement device protocol states in a variety of ways, malware authors also have different compliance levels with industrial protocol specifications. These differences in implementation can help to identify and profile malware. the Industroyer/CrashOverride IEC 104 malware module violated the state machine of the IEC 104 specification when compared to the OEM implementation. On certain OEMs and devices, the Industroyer/CrashOverride malware returns TCP RSTs due to this violation. Monitoring changes in statistical trends to finite state machines makes it possible to detect different dialects of industrial protocols and these dialects may help to detect and fingerprint malware [3]. Finally, we note that *although a diversity of vendors and devices can help asset owners fingerprint devices and mitigate common-mode vulnerabilities, this diversity may also introduce vulnerabilities due to differences among mutually-intelligible protocol implementations*.

C. Security Automata and System Baselining

Many industrial protocols consist of multiple states distributed across networked devices. At a high level, many industrial devices can be in a *program mode*, where they accept new logic programs; *stop mode*, where they are not executing logic; and *run mode*, where they execute the embedded program. The overarching mode of the device may limit what protocol actions can occur. For example, a device will not allow the industrial communications associated with writing a new program to a device unless that device is in program mode; a device ignores OPERATE commands while in stop mode.

In such a scenario, a high-level enforceable security policy for system baselining could construct a property over network traces that maps a function code for a DNP3 command to either ENABLED or REJECT states given the mode. Such a mapping is a property as it can be checked for each element in the trace sequence and a safety property since no prefix of a recognized sequence of commands can contain a command mapped to the REJECT category. Here we assume that when the security automaton implementing this system baselining policy encounters an ignored command, it alerts to indicate anomalous behavior. Another set of examples are the authentication sequences for DNP3 and IEC 104, which consist of a sequence of commands and values with a strict order of operations. Devices enforce the policy of the state machine by either ignoring packets sent out of state (thereby not enforcing a safety property) or sending a TCP RST back to the offending host to reset the connection (enforcing a safety property).

V. Conclusion

This paper explored the hypothesis that specific operational events in ICS/OT networks form a language whose strings are security artifacts collected across network and device data. Constructing security automata for sublanguages that are used within an ICS system define a system baseline with multiple potential benefits. First, facility or environment-driven baselining could reduce the language complexity of inter-device communications within ICS networks from a protocol *in general* to language constructs which are actually used *in a specific facility or environment*. Second, *system* baselines could help asset owners consciously and systematically choose between the ability to fingerprint devices on a network or mitigating common-mode failures through device diversity and introducing vulnerabilities due to non-equivalent ICS protocol dialects. Finally, given that the same procedure may have multiple implementations within and across protocols, it may be possible to evaluate tradeoffs among different ICS protocols based on the degree to which they reduce language complexity, thereby realizing the principle of least privilege, an important traditional security concept for modern-day, zero trust architectures. In practice, such automata could enforce baselined communications policies for micro-services segmented by *Software Defined Networking (SDN)* technologies.

ICS and OT networks provide a rich application domain for language-theoretic security. These systems are vital to our critical infrastructure systems and civilization as a whole. The intent of our research, if successful, is for asset owners to systematically obtain deeper understanding of their systems, and to discover unexpected behaviors indicative of potentially-adversarial behavior.

References

[1] F. B. Schneider, "Enforceable security policies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 1, pp. 30–50, 2000.

[2] S. Brizinov, "Hunting EtherNet/IP Protocol Stacks. (July 19, 2022). SANS ICS Security Summit," Accessed June 6, 2023. [Online video]. Available: https://www.youtube.com/watch?v=0jftEYDo0ao.

[3] D. Gunter and D. Michaud-Soucy, "Stateful Protocol Hunting. (June 11, 2019). 5th Stockholm International Summit on Cyber Security in SCADA," Accessed June 9, 2023. [Online video]. Available: https://www.youtube.com/watch?v=KTczBtb2ReU.

[4] J. Hanna, "Graphing OT Protocol Relationships to Define Network TTPs," MITRE, 2021.

[5] L. Sassaman, M. L. Patterson, S. Bratus, and M. E. Locasto, "Security applications of formal language theory," *IEEE Systems Journal*, vol. 7, no. 3, pp. 489–500, 2013.

[6] M. Johnson-Barbier and D. Gunter, "Utilizing Operations Data for Enhanced Cyber Threat Detection. (April 18, 2019). PI World," Accessed June 9, 2023. [Online video]. Available: https://www.youtube.com/watch?v=Inn6FPPPaXN1w.

[7] G. A. Weaver and S. W. Smith, "XUTools: Unix Commands for Processing Next-Generation Structured Text," in *Proceedings of the 26th Large Installation System Administration Conference (LISA '12)*. USENIX, 2012, pp. 83–99.

[8] D. Eads, K. Glocer, S. Perkins, and J. Theiler, "Grammar-Guided Feature Extraction for Time Series Classification," in *Proceedings of the 9th Annual Conference on Neural Information Processing Systems (NIPS'05)*, 2005.

[9] M. McFail, "Applying Detection Engineering Methods to ICS (June 14, 2022). IoB Working Group," Accessed June 14, 2023. [Online]. Available: https://us-isr-energycenter.org/energy_cyber/materials/IoBWG%20ICS%20Detection%20Engineering.pdf.

[10] Y. Huang, H. Shu, F. Kang, and Y. Guang, "Protocol Reverse-Engineering Methods and Tools: A Survey," *Computer Communications*, vol. 182, pp. 238–254, 2022.

[11] L. Lamport, "Proving the Correctness of Multiprocess Programs," *IEEE Transactions on Software Engineering*, no. 2, pp. 125–143, 1977.

[12] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *Journal of Computer Security*, vol. 18, no. 6, pp. 1157–1210, 2010.

[13] A. Schulz, E. Aubin, P. Trepagnier, and A. Wollaber, "Cyber baselining: Statistical properties of cyber time series and the search for stability," in *Proceedings of the 2019 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2019, pp. 1–7.

[14] C. Y. Cho, D. Babić, E. C. R. Shin, and D. Song, "Inference and Analysis of Formal Models of Botnet Command and Control Protocols," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 426–439.

[15] Z. Shu and G. Yan, "IoTInfer: Automated Blackbox Fuzz Testing of IoT Network Protocols Guided by Finite State Machine Inference," *IEEE Internet of Things Journal*, vol. 9, no. 22, pp. 22 737–22 751, 2022.

[16] P. Fiterau-Brostean, B. Jonsson, R. Merget, J. De Ruiter, K. Sagonas, and J. Somorovsky, "Analysis of DTLS Implementations Using Protocol State Fuzzing," in *29th USENIX Security Symposium (USENIX Security '20)*. USENIX, 2020, pp. 2523–2540.

[17] Y. Hsu, G. Shu, and D. Lee, "A Model-Based Approach to Security Flaw Detection of Network Protocol Implementations," in *Proceedings of the 2008 IEEE International Conference on Network Protocols*. IEEE, 2008, pp. 114–123.

[18] T. Krueger, H. Gascon, N. Krämer, and K. Rieck, "Learning Stateful Models for Network Honeypots," in *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*. ACM, 2012, pp. 37–48.

[19] C. Cao, A. Blaise, S. Verwer, and F. Rebecchi, "Learning State Machines to Monitor and Detect Anomalies on a Kubernetes Cluster," in *Proceedings of the 17th International Conference on Availability, Reliability, and Security*, 2022, pp. 1–9.

[20] G. A. Weaver, "Security-Policy Analysis with eXtended Unix Tools," Ph.D. dissertation, Department of Computer Science, Dartmouth College, 2013, [Online]. Available: https://digitalcommons.dartmouth.edu/dissertations/38/.

[21] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: Automatic Protocol Reverse Engineering from Network Traces," in *16th USENIX Security Symposium (USENIX Security '07)*. USENIX, 2007, pp. 1–14.