INL/CON-24-81419-Revision-0

SCA Tools - SCRM Value Add or Lossy Noise Machines

October 2024

Robert J Erbes, Micaela Gallegos

Changing the World's Energy Future

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

INL/CON-24-81419-Revision-0

SCA Tools - SCRM Value Add or Lossy Noise Machines

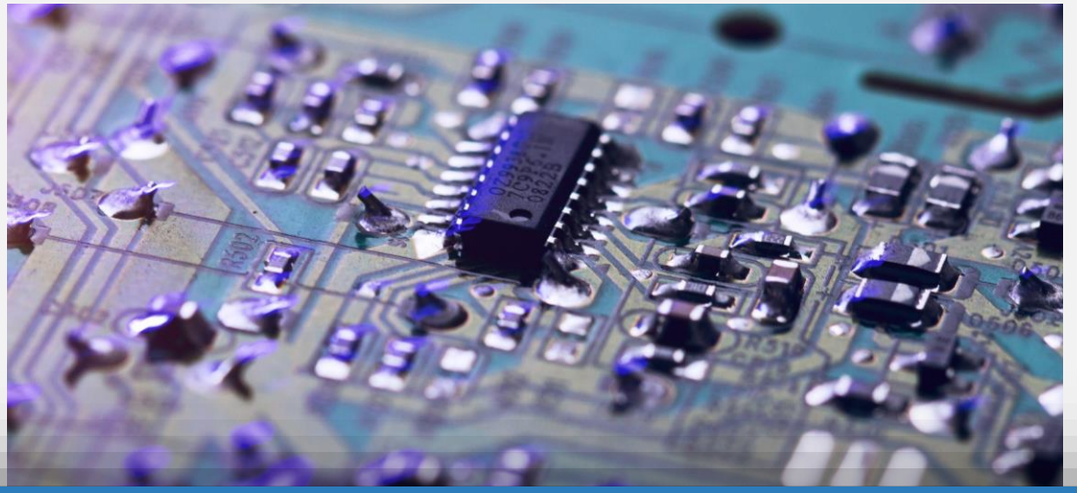
Robert J Erbes, Micaela Gallegos

October 2024

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



U.S. DEPARTMENT OF ENERGY | OFFICE OF Cybersecurity, Energy Security, and Emergency Response

CyTRICS™ Cyber Testing for Resilient Industrial Control Systems

Software Composition Analysis Tools:
SCRM Value Add or Lossy Noise Machines?

Robert Erbes, INL

Micaela Gallegos, LLNL

Idaho National Laboratory

INL/CON-24-81419

CyTRICS

SCA & SBOM Tool Analysis

Analysis Results

Edge Cases

Edge Case Results

Edge Case Results Summary

Takeaways

Future Work

CyTRICS Intro

- Cyber Testing for Resilient Industrial Control Systems, née Supply Chain Test Bed
- Testing arm of DOE CESER's Energy Cyber Sense
- Cyber Supply Chain Security

CyTRICS

SCA & SBOM Tool Analysis

- Analysis Results
- Edge Cases
- Edge Case Results
- Edge Case Results Summary
- Takeaways
- Future Work

Research Design

- **Goal:** To identify commercial SCA/SBOM tools that can quickly enumerate components in firmware binaries and produce an SBOM equivalent to manual analysis in less time.
- Steps:
  - Identify SBOM tools and assess suitability
  - Procure & deploy commercial tools
  - Test tools' capabilities & evaluate results

Tool Selection

- Surveyed and gathered information about the general capabilities of 33 open source & commercial SBOM tools
- Requested demos and more detailed information from 12 commercial vendors
  - Lessons learned:
    - A lot of tools in the SBOM space are focused on SCA for **source code only**
    - A niche area of tools specialize in Binary Composition Analysis
- Identified & procured 3 tools for testing

Tool Procurement & Evaluation

- **Procured 3 tools**
  - All had Binary Composition Analysis (BCA) Capabilities
  - Tool A (hybrid on-premise)
  - Tool B (SaaS)
  - Tool C (SaaS)
- **Ran two types of tests**
  - Submitted 5 firmware samples to each tool
  - Submitted custom edge cases developed with INL to each* tool
- **Developed a set of evaluation criteria**

*one tool was unavailable for ½ of the tests

Tool Evaluation Criteria

Categories
Accuracy and Completeness
Ease of Use
Integration and Compatibility
Customization and Configurability
Scalability
Security Features
Comprehensive Reporting
Licensing Compliance
Support and Documentation
Cost and Licensing Model
Community and User Feedback
Regulatory Compliance

Value	Description
1	Does not meet expectations
2	Slightly below expectations
3	Meets expectations
4	Exceeds expectations
5	Greatly exceeds expectations

- Developed 12 categories for scoring
- Tools were scored on each category using a Likert scale, ranging from 1 to 5.
- Ex:
  - Ability to accurately identify and list all the components within the software, identify dependencies and their versions
  - Intuitiveness of the user interface and navigation and efficient in generating and managing SBOMs

Firmware Sample Testing

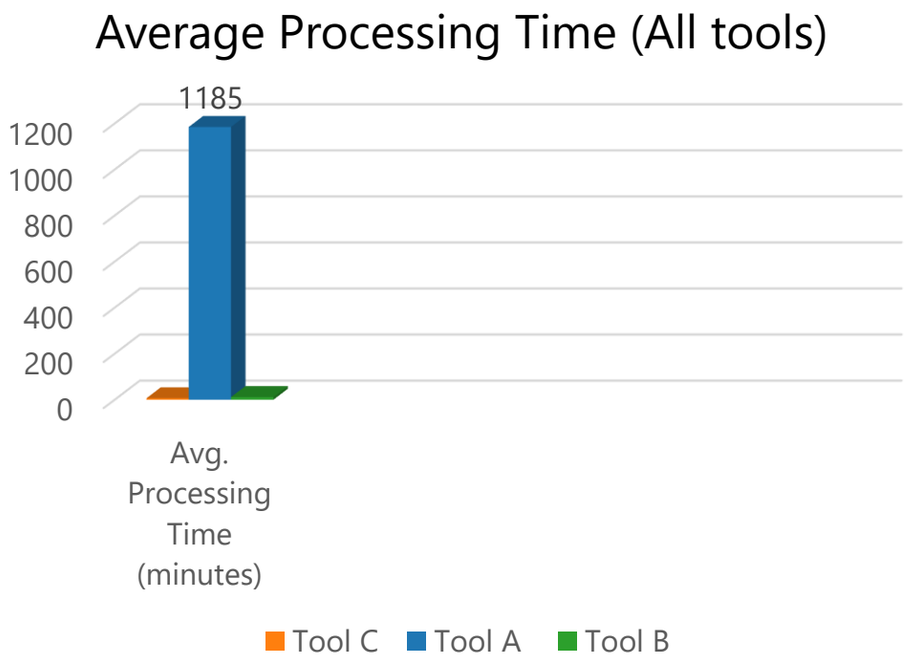
- Submitted 5 firmware samples to each tool
  - varied in vendor/make/model, compression format, size, and components
- Each sample had previously gone through the CyTRICS Testing process and had an SBOM associated with it
  - 4 SBOMs were the result of manual analysis, at times supplemented by internal tools
  - 1 SBOM was provided by the vendor
- These SBOMS served as our ground truth

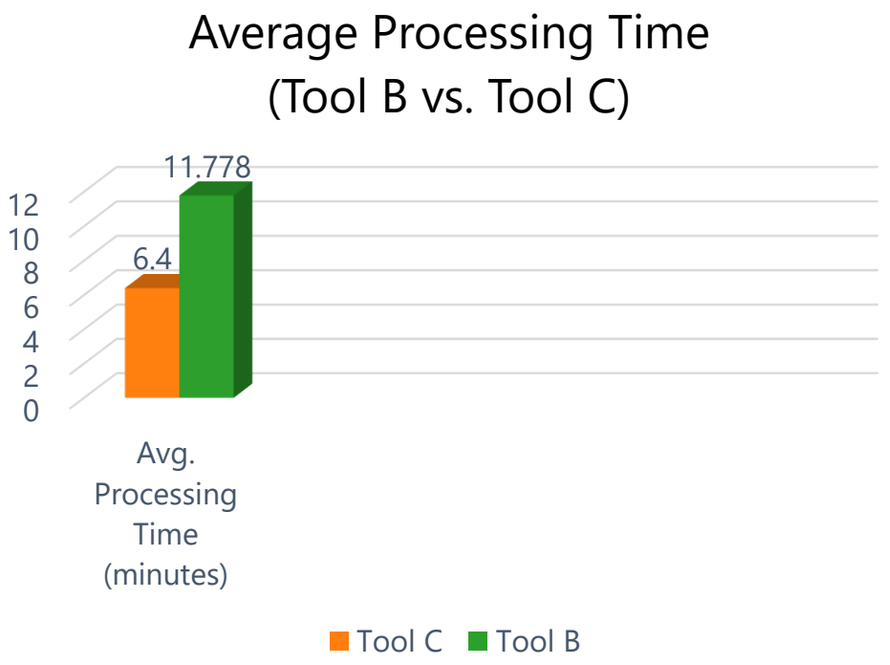
#	Notable File Types	Architecture	File Size
Sample 1	Zip files Motorolla s-record files	PPC32	6.56 MB
Sample 2	Executable files Source code files Windows setup files	PPC32	1.09 GB
Sample 3	Zip files VxWorks Javascript ARM binaries	ARM	195 MB
Sample 4	Zip files VxWorks Source code files	i386	28.0 MB
Sample 5	7zip files Java files Docker images	x86	1.65 GB

CyTRICS
SCA & SBOM Tool Analysis
Analysis Results
Edge Cases
Edge Case Results
Edge Case Results Summary
Takeaways
Future Work

Firmware Sample Testing – Notable Findings

- The hybrid on-premise instance took longer to process the files
- Of the SaaS instances, Tool C had a shorter processing time
  - This could be influenced by the ability to handle certain file types
  - Need more data points for accurate results





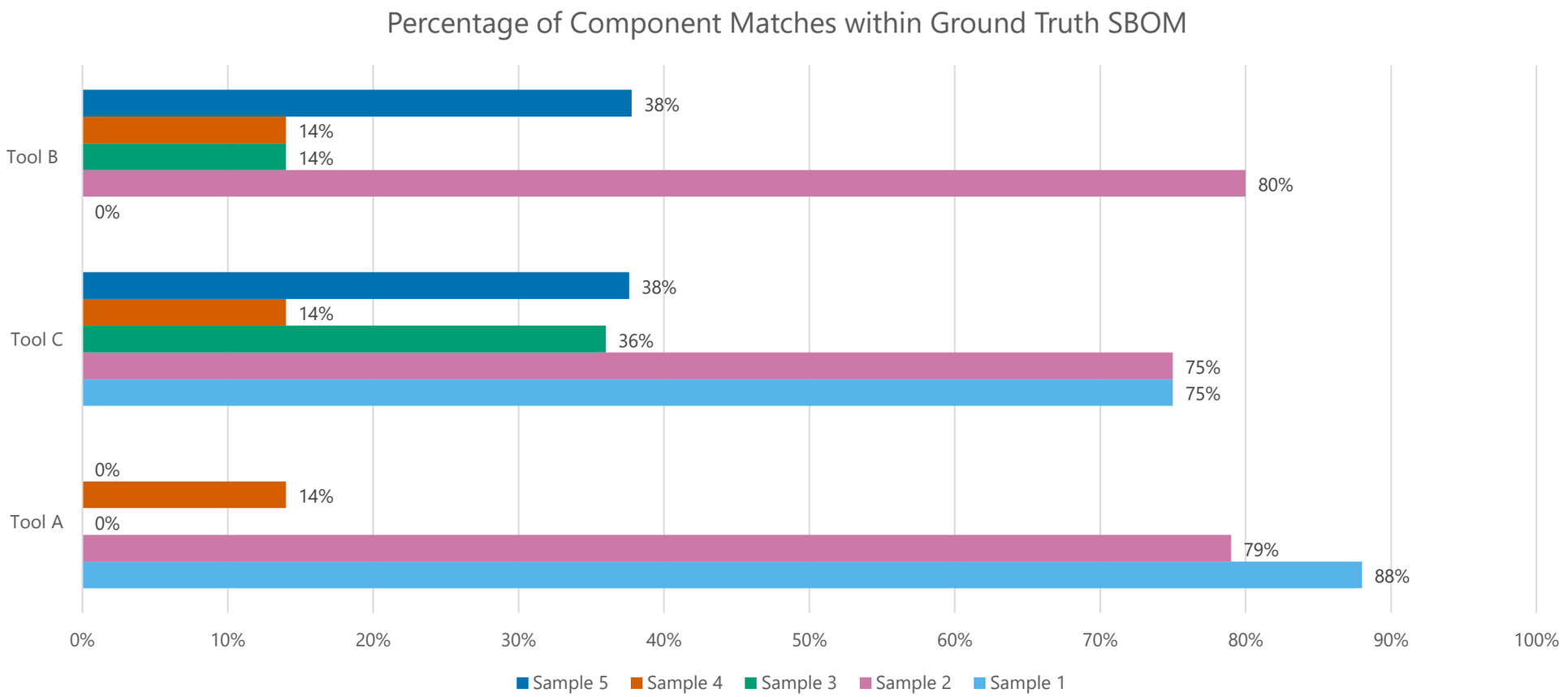
Firmware Sample Testing – Notable Findings

- The tools had trouble processing and identifying components in several of the firmware samples
- Tool C, a SaaS instance, initially failed to produce results for 3 samples but later implemented modifications to enhance its analysis capabilities which resulted in better outcomes.

#	Tool A	Tool B	Tool C
Sample 1	✓	✗	✓
Sample 2	*Initial upload had large processing time. Second Submission was successful	✓	* Initial upload failed, modifications made for reanalysis
Sample 3	*File processed but had few results	✓	* Initial upload had few results, modifications made for reanalysis
Sample 4	✓	✓	✓
Sample 5	✗	✓	* Initial upload failed, modifications made for reanalysis

Firmware Sample Testing - Results

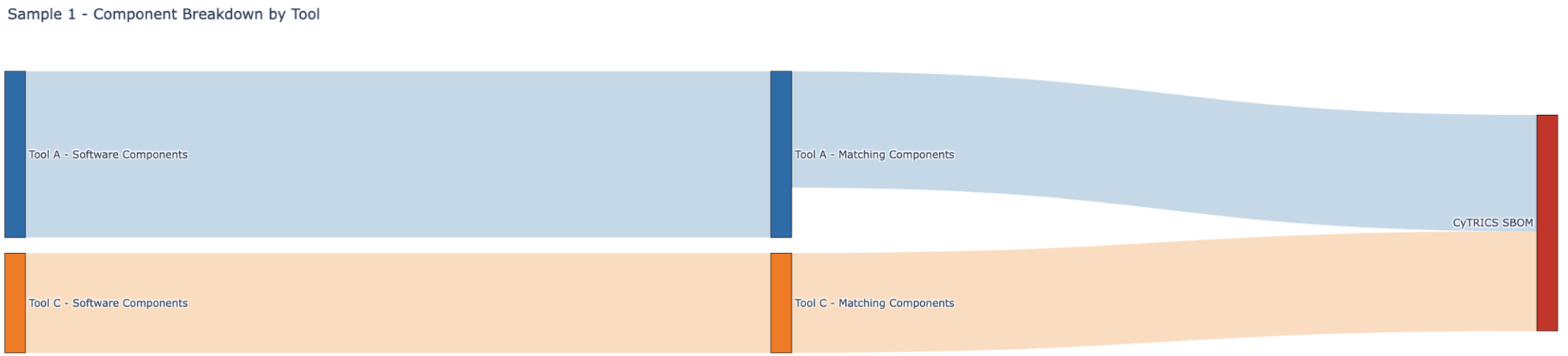
- Each tools' SBOM results were compared against SBOM outputs from the CyTRICS Test Operations team, and in one case against a vendor provided SBOM.
- No one tool consistently outperformed the others on all 5 samples.



Firmware Sample 1- Results

#	Notable File Types	Architecture	File Size
Sample 1	Zip files Motorolla s-record files	PPC32	6.56 MB

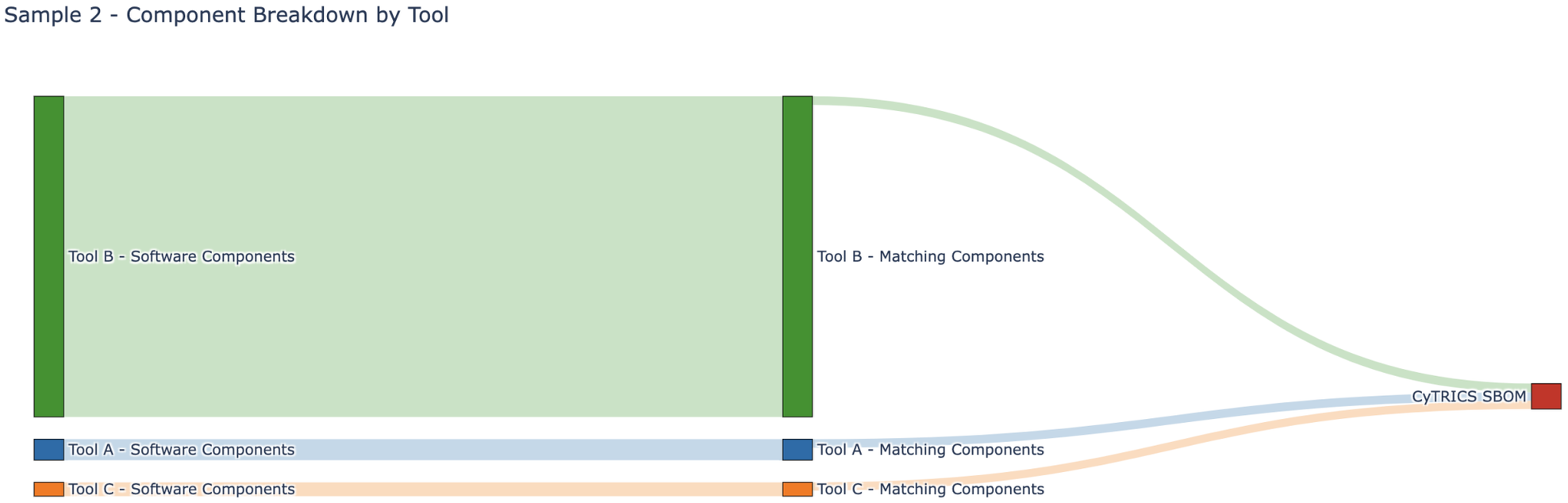
- Tool B was unable to find any components
- Tool A identified 87.5% of the components within the ground truth SBOM, and Tool C identified 75%



Firmware Sample 2 - Results

#	Notable File Types	Architecture	File Size
Sample 2	Executable files Source code files Windows setup files	PPC32	1.09 GB

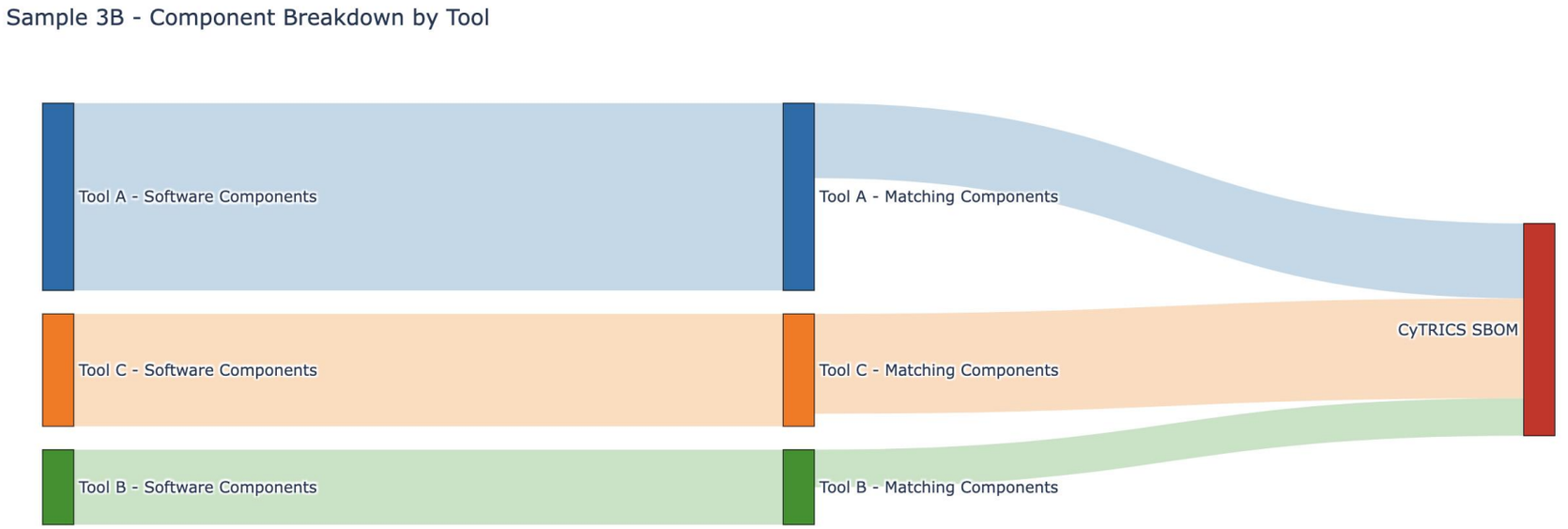
- All tools found a similar number of matching components.
- The range between the highest and lowest values was 9.



Firmware Sample 3 - Results

#	Notable File Types	Architecture	File Size
Sample 3	Zip files VxWorks Javascript ARM binaries	ARM	195 MB

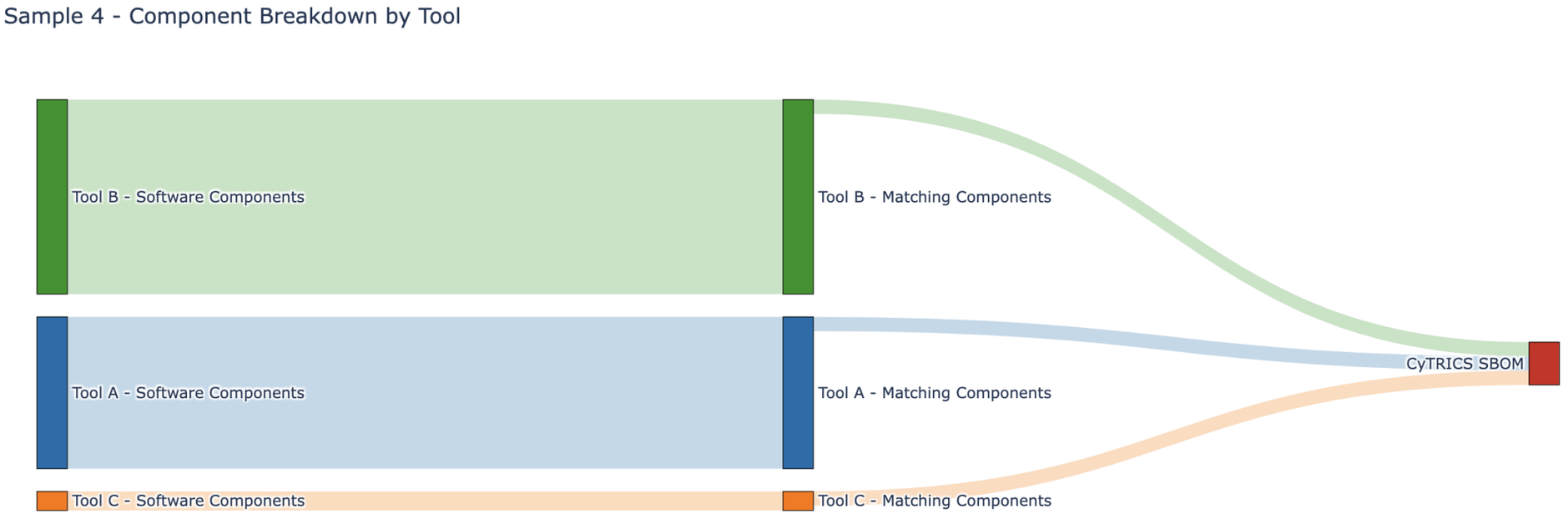
- Tool A encountered difficulties analyzing Sample 3, which included Sample 3b, but demonstrated improved performance on the subset



Firmware Sample 4 - Results

#	Notable File Types	Architecture	File Size
Sample 4	Zip files VxWorks Source code files	i386	28.0 MB

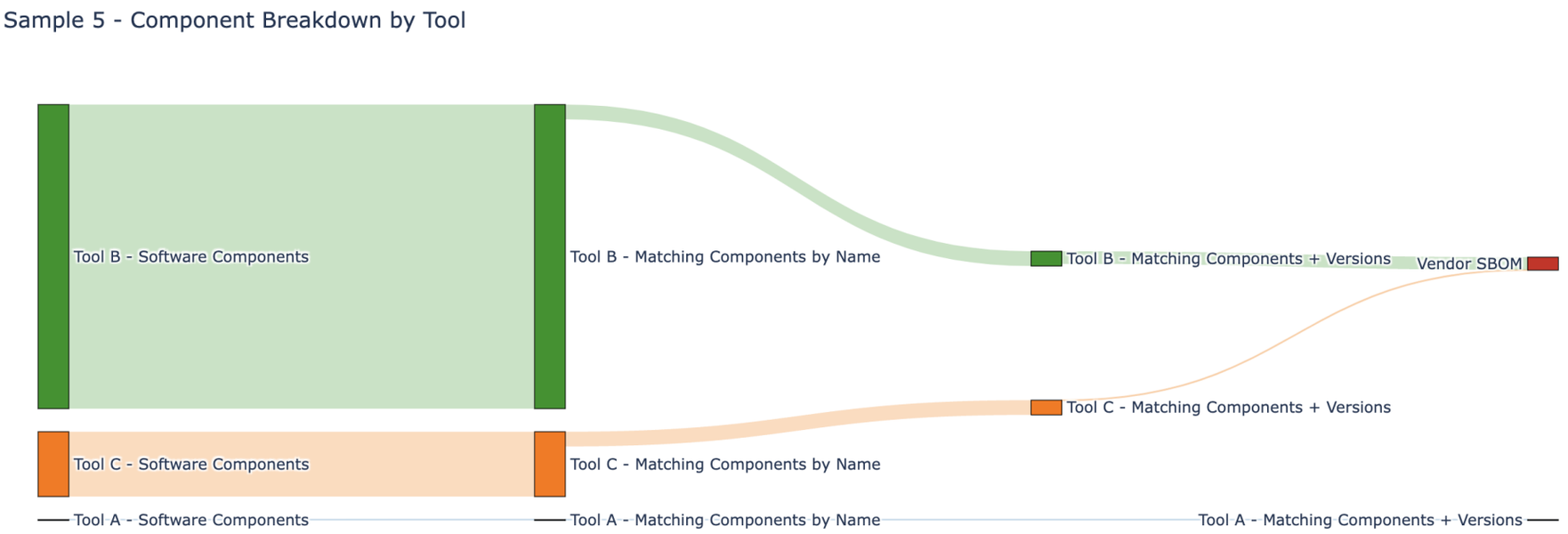
- Tools varied in the number of components identified overall, but all identified 3 matching components (yes, the same components)



Firmware Sample Testing - Results

#	Notable File Types	Architecture	File Size
Sample 5	7zip files Java files Docker images	x86	1.65 GB

- Tools B & C matched a similar number of components by name but, Tool B had more accurate versions for those components when compared to the vendor provided SBOM
- Overall, the tools identified many components, but not all of those matched the components listed within the CyTRICS SBOMS
  - Limitation of our dataset
- Likewise, the CyTRICS SBOMs generated by manual analysis identified components that the tools didn't identify



Evaluation Scoring - Results

- Each tool has its own strengths
  - Tool A: Licensing Compliance, Community and User Feedback
  - Tool B: Scalability, Support and Documentation, and Integration and Compatibility
  - Tool C: Ease of use, Customization and Configurability, Comprehensive Reporting
- Performance in the Accuracy and Completeness category varied by tool in the firmware samples, so we decided to do some more investigating

Criteria	Tool A	Tool B	Tool C
Accuracy and Completeness	3	3	3
Ease of Use	2	3	4
Integration and Compatibility	2	4	3
Customization and Configurability	3.5	2	4
Scalability	2.5	5	3
Security Features	3	4	3
Comprehensive Reporting	3	4	4
Licensing Compliance	4	3	3
Support and Documentation	2	5	4
Cost and Licensing Model	2	3	3
Community and User Feedback	3	3	2
Regulatory Compliance	2	2	2
Total (60)	32	41	38

Value	Description
1	Does not meet expectations
2	Slightly below expectations
3	Meets expectations
4	Exceeds expectations
5	Greatly exceeds expectations

CyTRICS

SCA & SBOM Tool Analysis

Analysis Results

Edge Cases

Edge Case Results

Edge Case Results Summary

Takeaways

Future Work

Why Edge Cases

- The CyTRICS program has several years experience manually generating SBOMs from binaries
- Encountered several instances where the peculiarities of the systems we look at make automation “difficult”
- difficulty == completeness and accuracy

Edge Case Selection

- Took list of known-issues we had encountered
- Added several brainstormed issues that an automated tool could run into
- Includes two main perspectives:
  - Intentional supply chain attacks
  - Common development practices
- Varied “difficulty”

Edge Case Descriptions

- Real Library (control)
- In Name Only
- Changed Filename
- Changed Name
- Complete String Fake
- Limited String Fake
- Strings in Text File
- Changed Version
- Changed Version Compiled
- Statically Compiled
- CGO_SSL
- CGO_SSL_UPX
- Backported Patches
- Backdoored

CyTRICS

SCA & SBOM Tool Analysis

Analysis Results

Edge Cases

Edge Case Results

Edge Case Results Summary

Takeaways

Future Work

Real Library (control)

1

- OpenSSL 1.1.1f
  - 64-bit Elf - libssl.so.1.1
- Pulled from Ubuntu 22.04 LTS

✓ Success: Identified as OpenSSL 1.1.1f

✗ Failure: Identified as anything else.

Tool A	Tool B	Tool C
✓	✓	n/a

In Name Only

1

- An ASCII text file renamed to “libssl.so.1.1”
- echo “This is just a text file.” >> libssl.so.1.1

✓ Success: Identified as ASCII data

✗ Failure: Identified as OpenSSL 1.1.1f

Tool A	Tool B	Tool C
✓	✓	n/a

Changed Filename

1

- libssl.so.1.1 → libmml.so.1.1

- Otherwise, the same

✓ Success: Identified as OpenSSL 1.1.1f

✗ Failure: Identified as anything else

Tool A	Tool B	Tool C
✓	✓	✓

Changed Name

2

- Copy of OpenSSL 1.1.1f binary
- All instances of “SSL” and “ssl” were changed to “SSM” and “ssm” respectively
- libssm.so.1.1

✓ Success: Identified as OpenSSL 1.1.1f

✗ Failure: Identified as unknown binary

Tool A	Tool B	Tool C
✗	✓	✗

Strings in Text File

1

- An ASCII text file containing strings pulled from the real libssl binary, then renamed to libssl.so.1.1

- strings libssl.so.1.1 > strings.txt
- mv strings.txt libssl.so.1.1

✓ Success: Identified as ASCII data

✗ Failure: Identified as OpenSSL 1.1.1f

Tool A	Tool B	Tool C
✓	✓	n/a

Complete String Fake

2

- A compiled and stripped shared object (library) consisting of a toy function, and
- a complete copy of *all* 4389 ASCII strings from the real libssl stored as a global array

✓ Success: Identified as an unknown ELF binary

✗ Failure: Identified as OpenSSL 1.1.1f

Tool A	Tool B	Tool C
✗	✗	✗

```c
#include <stdio.h>

#define STRINGCOUNT 4389
#define STRINGLENGTH 100

extern void aFunc();
char stringArray [STRINGCOUNT][STRINGLENGTH] = {
        "/q T",
        "T6AB\"",
        "@@-X",
        "DR0\"Dp",
        "B@90",
        "VUh;",
        "*x3V|",
        "jXR%|",
        "h[LoyY",
        "+Q=g6",
        "([j4",
        "c%{^",
        "kP:QF",
        "w\\$|",
        "YIs{",
        "(,(P,",
        "[{\"1",
        "]W     #"
```

Limited String Fake

2

- A compiled and stripped shared object (library) consisting of a toy function, and
- a complete copy of a *subset* (41) of ASCII strings from the real libssl stored as a global array, and
- an ELF executable of the same code
- grep -i OPENSSL libssl.so.1.1 >> limited_strings.txt

✓ Success: Identified as an unknown ELF binary

✗ Failure: Identified as OpenSSL 1.1.1f

Tool A	Tool B	Tool C
✗	✗	✗

```c
#include <stdio.h>

#define STRINGCOUNT 41
#define STRINGLENGTH 30

extern void aFunc();
char stringArray [STRINGCOUNT][STRINGLENGTH] = {
        "OPENSSL_sk_new_null", "OPENSSL_sk_find", "OPENSSL_sk_push", "OPENSSL_sk_free",
        "OPENSSL_cleanse", "OPENSSL_sk_pop_free", "OPENSSL_sk_num", "OPENSSL_sk_value",
        "OPENSSL_sk_new_reserve", "OPENSSL_LH_new", "OPENSSL_LH_retrieve", "OPENSSL_LH_free",
        "OPENSSL_LH_insert", "OPENSSL_sk_set_cmp_func", "OPENSSL_DIR_read", "OPENSSL_DIR_end",
        "OPENSSL_sk_shift", "OPENSSL_sk_pop", "OPENSSL_sk_new", "OPENSSL_sk_sort",
        "OPENSSL_sk_dup", "OPENSSL_sk_delete", "OPENSSL_sk_insert", "OPENSSL_cipher_name",
        "OPENSSL_atexit", "OPENSSL_init_ssl", "OPENSSL_init_crypto", "OPENSSL_LH_num_items",
        "OPENSSL_LH_delete", "OPENSSL_LH_get_down_load", "OPENSSL_LH_set_down_load",
        "OPENSSL_LH_doall_arg", "OPENSSL_1_1_0", "OPENSSL_1_1_0d", "OPENSSL_1_1_1",
        "OPENSSL_1_1_1a", "OPENSSL_1_1_0i", "OPENSSL_1_1_0f", "OPENSSL_DIR_read(&ctx, '",
        "OPENSSL_init_ssl", "OpenSSL 1.1.1f  31 Mar 2020"
};

void aFunc(){
        for (int i=0; i<STRINGCOUNT; i++){
                printf("%s\n", stringArray[i]);
        }
}

int main() {

        aFunc();

        return 0;
}
```

Changed Version

2

- Copy of the real OpenSSL library version 1.1.1f
- The version string was changed to match version 1.1.1q using a hex editor

✓ Success: Identified as OpenSSL 1.1.1f

✗ Failure: Identified as OpenSSL 1.1.1q

Tool A	Tool B	Tool C
n/a	✗	✗

Changed Version Compiled

3

- OpenSSL library version 1.1.1f
- The version string *and* version number were changed to match version 1.1.1q
- No other changes
- Compiled to shared object

✓ Success: Identified as OpenSSL 1.1.1f

✗ Failure: Identified as OpenSSL 1.1.1q

Tool A	Tool B	Tool C
n/a	✗	✗

Statically Compiled

2

- OpenSSL 1.1.1f compiled into an archive for static linking into other programs
- libssl.a and libcrypto.a

✓ Success: Identified as OpenSSL 1.1.1f

✗ Failure: Identified as anything else

Tool A	Tool B	Tool C
n/a	✓	✓

CGO_SSL

2

- Golang application, with OpenSSL 1.1.1f statically compiled into the golang binary using Go's cgo interface.
- Application calls OPENSSL_init_ssl()
- 1) go-wrapped-ssl_unstripped
- 2) go-wrapped-ssl_stripped

✓ Success: Identified as including OpenSSL 1.1.1f

✗ Failure: OpenSSL 1.1.1f not identified

Tool A	Tool B	Tool C
n/a	✓	✓

```go
package main

import "fmt"

// #cgo CFLAGS: -I/home/edgecases/cgo_ssl/openssl/include/
// #cgo LDFLAGS: /home/edgecases/cgo_ssl/libssl.a /home/edgecases/cgo_ssl/libcrypto.a
// #include <openssl/ssl.h>
import "C"

func main() {
        fmt.Printf("Initializing library\n")
        C.OPENSSL_init_ssl(C.OPENSSL_INIT_LOAD_SSL_STRINGS, C.OPENSSL_INIT_new())
        fmt.Printf("Done\n");
}
```

CGO_SSL_UPX

3

- Golang application, with OpenSSL 1.1.1f statically compiled into the golang binary using Go's cgo interface.
- Stripped binary packed using UPX
- upx --brute go-wrapped-ssl_stripped

✓ Success: Identified as including OpenSSL 1.1.1f

✗ Failure: OpenSSL 1.1.1f not identified

Tool A	Tool B	Tool C
n/a	✓	✗

Backported Patches

3

- OpenSSL 1.1.1f with the following CVE's patched before compiling into shared object
- CVE-2023-2650 (Medium Severity) fixed in 1.1.1u
- CVE-2023-0286 (High Severity) fixed in 1.1.1t
- CVE-2022-4450 (Medium Severity) fixed in 1.1.1t

✓ Success: The patched CVEs are not listed

✗ Failure: The patched CVEs are listed as applicable

Tool A	Tool B	Tool C
n/a	✗	✗

Backdoored

3

- OpenSSL 1.1.1f modified with an obvious backdoor
- OPENSSL_init_ssl() calls added function named *backdoor()*
- *Backdoor()* kicks off a thread which runs a bind-shell on TCP port 4444

✓ Success: Backdoor is identified

✗ Failure: Backdoor is not identified

Tool A	Tool B	Tool C
n/a	✓	✗

CyTRICS

SCA & SBOM Tool Analysis

Analysis Results

Edge Cases

Edge Case Results

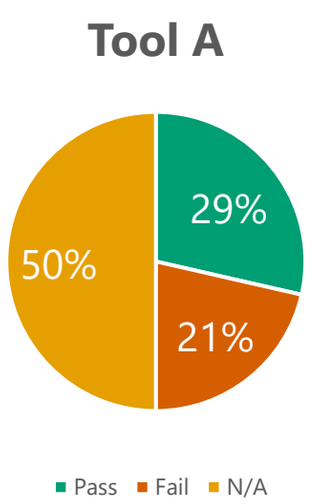
Edge Case Results Summary

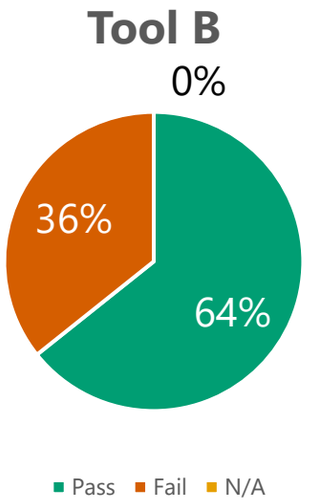
Takeaways

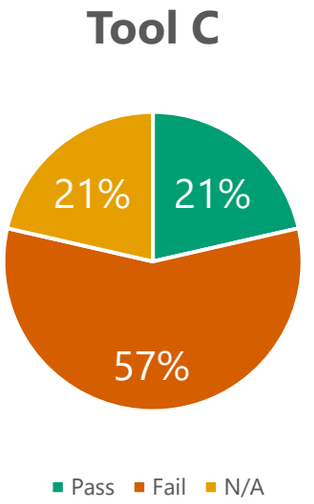
Future Work

Edge Case Testing - Results

- Submitted 14 edge case samples for processing by each tool
- Testing limitations:
  - One tool was unavailable for half the edge case tests, and another had difficulty processing some of the edge cases

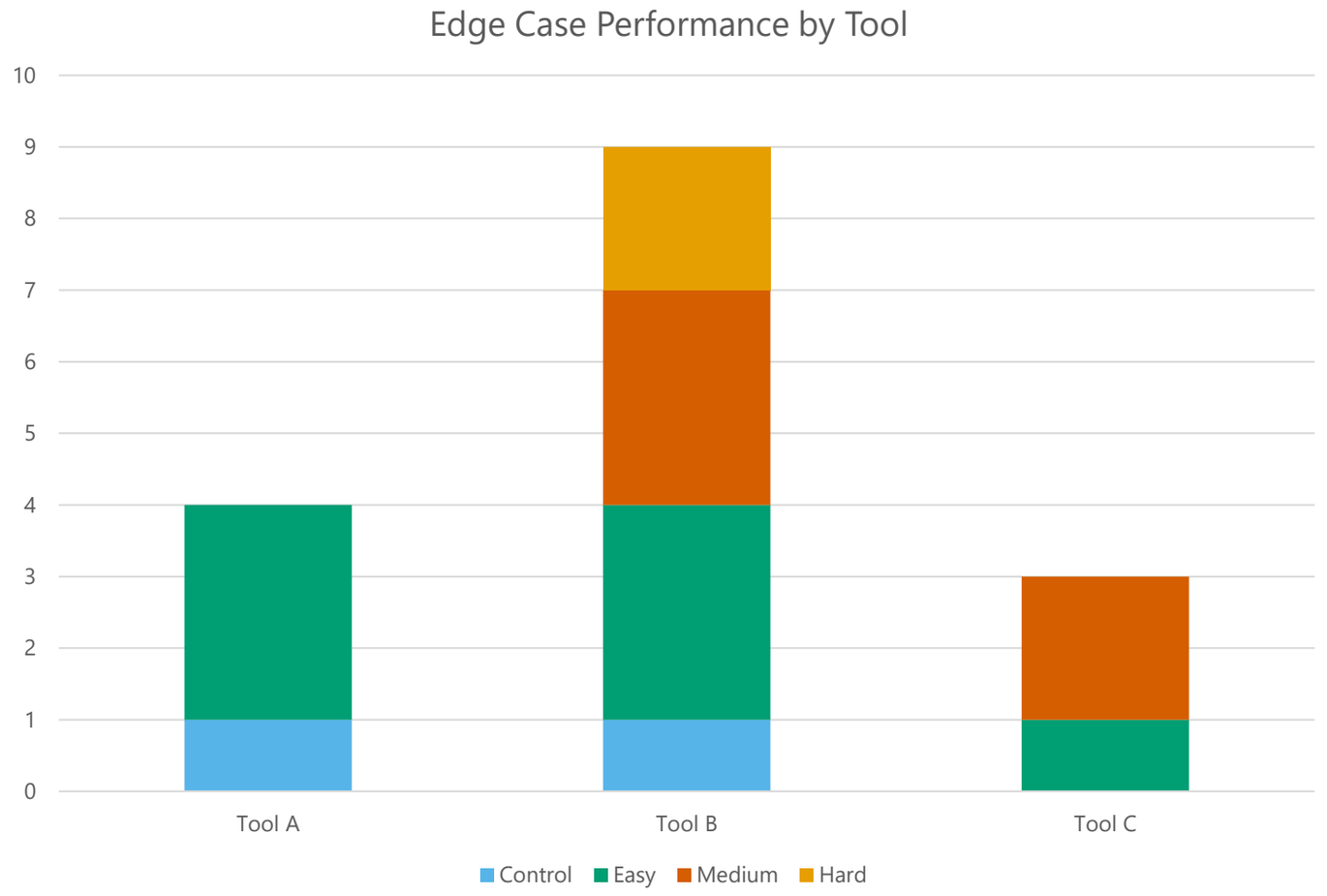






Edge Case Testing - Results

- Overall, Tool B passed the most number of tests and was able to handle multiple difficulty levels



Edge Case – Pass/Fail Breakdown

- Here's an overview of the tools' performance in the control case and the easy edge cases

Case	Tool A	Tool B	Tool C
Real_library	✓	✓	n/a
Changed_filename	✓	✓	✓
strings_in_text_file	✓	✓	n/a
In_name_only	✓	✓	n/a

✓ = passed edge case ✗ = failed edge case n/a = unable to process

Edge Case – Pass/Fail Breakdown

- Here's an overview of the tools' performance in the medium difficulty edge cases

Case	Tool A	Tool B	Tool C
Changed_name	✗	✓	✗
changed_version	n/a	✗	✗
Statically_compiled	n/a	✓	✓
complete_string_fake	✗	✗	✗
limited_string_fake	✗	✗	✗
Cgo_ssl.zip	n/a	✓	✓

✓ = passed edge case ✗ = failed edge case n/a = unable to process

Edge Case – Pass/Fail Breakdown

- Here's an overview of the tools' performance in the hard difficulty edge cases

Case	Tool A	Tool B	Tool C
Backported_patches.zip	n/a	✗	✗
Changed_version_compiled	n/a	✗	✗
Cgo_ssl_upx	n/a	✓	✗
backdoor	n/a	✓	✗

✓ = passed edge case ✗ = failed edge case n/a = unable to process

Edge Case – Difficulty Breakdown

- Tool C was unable to process the real library
- Tool A seemed comparable to Tool B, but was inoperable for the more difficult tests

Difficulty	Test	Points	Tool A	Tool B	Tool C
0	real_library	1	1	1	n/a
1	changed_filename	1	1	1	1
1	strings_in_text_file	1	1	1	n/a
1	in_name_only	1	1	1	n/a
2	changed_name	2	0	2	0
2	changed_version	2	n/a	0	0
2	statically_compiled	2	n/a	2	2
2	complete_string_fake	2	0	0	0
2	limited_string_fake	2	0	0	0
2	cgo_ssl.zip	2	n/a	2	2
3	backported_patches.zip	3	n/a	0	0
3	changed version_compiled	3	n/a	0	0
3	cgo_ssl_upx	3	n/a	1.5	0
3	backdoor	3	n/a	3	0
	Total	**28**	**4**	**14.5**	**5**

CyTRICS

SCA & SBOM Tool Analysis

Analysis Results

Edge Cases

Edge Case Results

Edge Case Results Summary

Takeaways

Future Work

Takeaways

- Our evaluation was not perfect and reflects a very specific use case
- BCA/SCA/SBOM tools are great for extracting information and giving overall results
- For firmware analysis, there is work to be done towards consistency and accuracy
- For the edge case analysis, some tools may rely heavily on strings for matches, instead of function/behavioral indicators

CyTRICS

SCA & SBOM Tool Analysis

Analysis Results

Edge Cases

Edge Case Results

Edge Case Results Summary

Takeaways

Future Work

Edge Case Categorization

- Define and document different classes/categories of edge cases
- Used to generate novel test cases
- E.g. file-based categorization

File Type Categorization Example

- Stand-Alone Files
  - Compiled Programs
    - Machine byte-code
    - Intermediate language
    - Multi-architecture Binaries
  - Interpreted Programs
    - Bash Scripts
    - HTML, JavaScript, Python, etc
  - Text Files
    - Source Code
    - Configuration Files
    - SREC, iHEX, etc
- Embedded Files
  - Statically compiled libraries
  - MSI installers, WiX, etc
- Compressed/Archived
  - Tar, 7z, zip, etc
- Packages
  - .deb, .rpm, etc
- Metadata
  - Debugging info
    - DWARF, PDB
  - Programmatic
    - golang

Thank you!