Securing Your ICS Software with the AttackSurface Host Analyzer (AHA)

Adam Hahn
Washington State University
School of Electrical Engineering &
Computer Science
Pullman, WA

Ali Tamimi
Washington State University
School of Electrical Engineering &
Computer Science
Pullman, WA

Dave Anderson
Washington State University
School of Electrical Engineering &
Computer Science
Pullman, WA

ABSTRACT

Implementing a secure development lifecycle (SDL) presents increasing challenges to software developers as they must ensure software correctly integrates both underlying operating system security features while also managing dependencies on third-party libraries or executables. There are a growing number of security functions that require a close integration between the OS security features and software builds to ensure strong protection. Furthermore, as software platforms grow in complexity, they present many opportunities for misconfigurations and inadequate defenses. This challenge is especially prevalent for industrial control systems (ICS), which often depend on both legacy software platforms, or out of date operating systems. This paper presents the AttackSurface Host Analyzer (AHA) tool, which is used to assess the security of a software platform through its integration with a host operating system. The tool collects data from the various platforms running on an OS, evaluates an array of security properties, and then introduces metrics and visualizations to provide feedback on the system's attack surface based on the external interconnections and the completeness of the available security protections. The paper then explores the attack surface of a variety of industry-standard ICS platforms to provide insight into the current degree of protection enabled by them.

ACM Reference format:

Adam Hahn, Ali Tamimi, and Dave Anderson. 2018. Securing Your ICS Software with the AttackSurface Host Analyzer (AHA). In *Proceedings of 2018 Industrial Control Systems Security (ICSS) Workshop, San Juan, Puerto Rico USA, December 2018 (ICSS 2018)*, 7 pages. <https://doi.org/10.1145/3295453.3295458>

1 INTRODUCTION

Developing secure software platforms introduces many challenges for modern ICS. Not only do they need to worry about the quality of the software, but must also explore security capabilities of the underlying operating system platform and must ensure all applications adequately integrate these underlying OS security features. Furthermore, because software platforms can have many different

applications, interconnections, and dependencies; it presents many opportunities where the misconfigurations or misunderstandings about the software integration can introduce vulnerabilities.

In recent years, substantial progress has been made in the security for both software and operating systems. For example, modern operating systems provide a variety of security functions, including Address Space Layout Randomization (ASLR), Data Execution Protection (DEP), control flow integrity schemes such as Control Flow Guard (CFG), and code signing/integrity features. However, there is still remains a gap in techniques to ensure that software is compiled and configured to utilize the optimal set of security features enabled by some target platforms, especially in many ICS platforms. Furthermore, software often maintains dependencies on binaries from many different sources, including in-house development or externally developed libraries. This creates substantial challenges in the analysis of the software interconnectivity and security posture. Unfortunately, there is currently a lack of tools available to help vendors and consumers understand the system-level risk of software platform based on the utilization of best-practice security configurations and the complexity of the system's attack surface.

This work introduces AHA, the Attack Surface Host Analyzer [1], that helps software developers and system owners understand the broader security posture of their platforms. It explores a variety of different security properties from the operating system regarding running software packages and the system's configuration. It then performs graph-based analysis of the system as an interconnected set of processes and provides both per/process and per/host attack surface scores. Further, it provides a graphical visualization to provide the end user with the ability to more clearly understand key areas of risk in the system. The tool is intended to be used within both testing and production system environments.

AHA provides the following key capabilities:

- (1) Perform the exploration of system processes and interconnections (especially to external systems),
- (2) Provide an overview of exploit mitigations and security mechanisms within a platform,
- (3) Evaluate the effectiveness of mechanisms to isolate privileges of various processes,
- (4) Compute an attack surface score based on a processes' level of security and external exposure, and
- (5) Provides a graphical user interface to produce an intuitive and explorable perspective of a platform's security.

This paper will explain the operation of the AHA tool and will identify various use-case scenarios where the tool can help software developers and consumers identify risky components within their platforms. Furthermore, in the development of AHA, the authors

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICSS '18, December 4, 2018, San Juan, PR, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6220-7/18/12...\$15.00
<https://doi.org/10.1145/3295453.3295458>

SDL Activities	Tools
Establish Design Requirements	DHS CSET [2]
Attack Surface Analysis Reduction	AHA
Use Threat Modeling	MS Threat Modeling Tool [3]
Use Approved Tools	GCC, LLVM, Clang
Deprecate Unsafe Functions	Coverify, Fortify, Lint
Perform State Analysis	Findbugs, DHS SWAMP [4]
Dynamic Analysis	AppScan, Purify, Valgrind
Fuzz Testing	Spike, American Fuzzy Lop
Attack Surface Review	AHA

Table 1: SDL Activities and Related Tools

identified a number of unique Microsoft Windows and Linux based properties which directly influence the security of applications running on these platforms. Therefore, the paper will also highlight these. It will identify some future tasks necessary to help AHA perform more complete attack surface analysis.

2 RELATED WORK

Multiple research efforts have explored the development of attack surface metrics. Research by [5] explores software-based metrics that includes entry and exits points within an application, along with communication channels and data items to create a single metric through the summation of these various properties. The metrics are then used to analyze and compare multiple real-world software platforms to demonstrate its effectiveness. Research by [6] introduces a Relative Attack Surface Quotient (RASQ) metric that enables multiple system configurations to be measured against each other. The metric analyzes systems based on combinations of defined targets, enablers, channels, protocols, and access rights sets. Example RASQ attack vectors include open sockets, named pipes, RPC endpoints, running services, weak ACLs, and user accounts. Work by [7] has demonstrated the ability to reduce system attack surface by reducing unnecessary OS system calls. Further work in [8] introduces system-level metrics to compare the attack surface of different certificate validation approaches.

Microsoft has introduced an Attack Surface Analyzer tool that can monitor a system for a variety of configurations that influence its attack surface and produce a comprehensive report [3]. This tool can be used to compare various system configurations to baseline attack surfaces for various operating systems images. Furthermore, the Cyber Independent Testing Laboratory (CITL) is currently identifying tools and techniques to quantify software risks through a combination of both static (e.g., exploit mitigations) and dynamic (e.g., fuzz testing) software security properties [9].

Software developers commonly follow a secure development lifecycle (SDL) which incorporates an array of model and testing methodologies to help prevent, identify, and mitigation possible vulnerabilities. During this process, a number of previously developed tools have been produced to assist with this SDL. Table 1 identifies various activities within the Microsoft SDL [10] and a set of potential tools available to support that activity. While most activities have a number of available tools, there is currently an insufficient focus on tools that can help in the attack surface analysis reduction and attack surface review activities.

Figure 1: System-level Attack Surface Analysis

3 ATTACK SURFACE DEFINITION

While previous work has explored attack surface from both a software [5] and network [8] perspective, this work will focus on the assessment from a single host which includes multiple software platforms. Based on this scope, the attack surface of the platform will be defined as the combination of the (i) degree of interconnectivity to other external systems, (ii) the degree to which state-of-the-art OS-level security mechanisms are enabled/supported by the platform, and (iii) the criticality of the data processed by the platforms. Figure 1 provides a conceptual overview of the proposed attack surface from for a specific process p , while the system's attack surface is defined through the system-level analysis of all software components.

- (1) *Connectivity*: The amount of connectivity between processes to external systems, along with other internal processes and the number of listening ports is a primary contributor to the attack surface. Therefore connectivity analysis is necessary to explore the number of various entry points to that software.
- (2) *Security Control*: In order to protect the process from various exploits, it is critical to implement state-of-the-art exploit mitigation techniques provided by many operating systems and compilers. These significantly decrease the likelihood that a software vulnerability can be leveraged by an attacker to remotely access that system. Furthermore, features such as code signing are important to ensure that all software originates from trustworthy sources.
- (3) *Privileges*: Finally, the privileges of the process should be included in the attack surface calculation. If privileges are not properly set, the attacker could gain access to excessive system resources by successfully compromising a process. Therefore, this component will be used to evaluate that the principle of least-privilege is adequately followed.

Based on the proposed attack surface domains, AHA will collect a variety of information and through the combination of two tools, one that performs the data collection and a second that performs the analysis, metrics, and visualization. The following section will provide further details on the proposed approach and the AHA tool.

Figure 2: AHA Functional Overview

4 AHA OVERVIEW

Figure 2 provides an overview of the proposed tool, which collects various data regarding the processes running on a host and then computes attack surface scores for the various system processes based on the collected data. The tool also provides a GUI to provides users with an easily understood interface to understand the overall attack surface of the platform. This section will provide a detailed overview of the data collection agent (AHA-Scraper), along with the analysis algorithms and GUI features (AHA-GUI).

4.1 Collection Agent (AHA-Scraper)

Data collected for AHA is primarily based on a software agent deployed on the target host, which will scan for a variety of system and process configurations that will be used for later attack surface analysis algorithms. There are currently two collector agents, one for both Windows and Linux. Both agents must be deployed on the host under evaluation and will utilize a variety of existing platform utilities and included executables to collect data from that host. While both Windows and Linux provide different security features and data collection approaches, each tool will be discussed individually. The data collection approach for the system will utilize the following phases, (1) the agent will scan the system for running processes, (2) it will extract a variety of security-related features from that process, and (3) it will identify network connections from that process to other processes and systems. Table 2 identifies the various tools used to support the Windows and Linux agents. The output of the collection agent is a log file that will be used for the later attack surface analysis algorithms (Section 4.2).

4.1.1 Windows Agent. The Windows agent utilizes PowerShell to perform the data collection. First, the tool monitors all running processes to get an index of what programs are currently executing on the system. Next, the CurrPorts (cports.exe) tool [11] is used to provide a list of interconnected processes.

After the process-layer data collection, additional information is collected from each executable. During this step, the tool performs an analysis of the process's binary (PE). Currently, much of the PE analysis is performed with the Get-PESecurity PowerShell script [12], which enables the collection of numerous security properties. Currently, the tool identifies the following security controls: Address Space Layout Randomization (ASLR), Data Execution Prevention (DEP), Code signing (Authenticode), Control Flow Guard (CFG), High Entropy (64 bit) ASLR (which Microsoft

Component	Windows Agent	Linux Agent
Platform	PowerShell, Currports, Get-PESecurity	Bash
Proc Name/Id	Currports	PS/Netstat
Connection	Currports	Netstat
ASLR	PE Header	Kernel/Proc
CFG	PE Header	LLVM CFI
Authenticode	PE Header	NA
SafeSEH	PE Header	NA
RELRO	NA	ELF

Table 2: Data collected by the host analysis agent

calls "HighEntropyVA"). Once completed, the agent combines the information gathered during the various scan stages into one output CSV (Comma Separated Value) file.

4.1.2 Linux Agent. The Linux agent utilizes a Bash script to perform the data collection. The script utilizes a number of existing Linux-based utilities to collect this data including *ps*, *netstat*, *readelf*, and *sysctl*. Furthermore, it also explores a variety of filesystem locations including */proc* and */sys*, to produce similar results to the Windows version. Since security features, such as control flow integrity, vary by compiler and particular Linux distribution, attaining feature parity with the Linux version is still future work. Binary authentication and signing is also an area of future work on the Linux scraper, as most Linux distributions only sign install packages rather than the executables on disk, and thus this will require further research to arrive at a workable solution to verify an installed system's binaries.

4.2 Attack Surface Analysis (AHA-GUI)

Once the AHA-Scraper has aggregated the data from the user input and the AHA-SCRAPER, it can perform a variety of different analysis modes for the system's attack surface. First, the processes and connections are modeled as a graph, $G = (V, E)$, where each process is represented by a node (V), as well as each external system and an external node (*ext*) representing remotely listening ports. Furthermore, each edge (E) shows some interconnectivity between the processes or between the process and external systems. The

attack surface analysis follows two phases, the first phase computes a process-specific metrics, while the second phase computes the system metric.

4.2.1 Process-Level Metrics. The process metric computes a score between 0-100 for each individual process based on the set of available security mechanisms integrated into the process, as demonstrated in Algorithm 1. The algorithm defines a base score of 0 for each process and then iterates through a list of platform-specific mitigations and adds the perspective value for that mitigation. The following list identifies the mitigation's scanned for and the default scoring applied for each.

- **ASLR & HighEntropyVA:** A score of 10 is added for both ASLR and HighEntropyVA. While ASLR provides some level of security, 32-bit binaries have limited benefits from this mechanisms. However, HighEntropyVA ensures 64-bit entropy and therefore provides significantly greater security.
- **DEP -** The DEP score of 10 is added if the process applies Data Execution Prevention which prevents writable pages in memory from being executed, which prevents attackers from writing code to a page of memory and then executing it. DEP combines a set of software mitigations with the hardware NX (no execute) bit.
- **ControlFlowGuard -** Control Flow Guard awards a score of 30 if found to be active. Control Flow Guard is awarded such a high score because it can prevent complex return-oriented programming attacks which other methods are not able to mitigate.
- **Authenticode:** If Authenticode is enabled and the process code is correctly signed, then a score of 10 is provided to emphasize the importance of having code properly signed by the developer and penalize binaries who is origins can not be properly verified.

4.2.2 System-Level Metrics. The system score (sys_{score}) expands on the process-level score to incorporate the connectivity of the process to both external and internal processes to explore the overall attack surface. The system score of a process shows how the security of other connected processes could have a propagative impact and lead to an increase of the attack surface for other processes. For the evaluation of the system score, the interconnection between the processes should be defined. If an external node which is listening to a port is connected to an internal process, the external process is referred to as the *parent* of the internal process. The other type of connectivity is the connection between two internal processes which is called *sibling*. For calculating the system score, the impact of parents and siblings should be evaluated. We define different types of scores for each process as follow.

- $N_{score}(i)$ Since we want to evaluate the propagative impacts of the processes to each other, we need a metric for calculation that shows the larger value as more influential in the propagative risk. While the score calculated in the previous part is larger when a process is more protected, we normalize it and subtract it from 1. Then we have a metric where the larger value represents a larger attack surface.

Algorithm 1 Process/System Attack Surface Analysis Algs

```

function PROCSCORE(process  $p$ )
   $pScore[] \leftarrow 0$ 
   $win\_mitigate = \{ASLR : 10, DEP : 10,$ 
     $authenticode : 10,$ 
     $Arch64 : 10,$ 
     $ControlFlowGuard : 30,$ 
     $HighEntropyVA : 10,$ 
     $Strongname : 10\}$ 
   $lin\_mitigate = \{relo\_true : 10,$ 
     $relo\_part : 5,$ 
     $stackcanary : 10\}$ 
   $win\_privs = \{localservice : 10, system : -50\}$ 
   $mitigate = \{win\_mitigate, lin\_mitigate, win\_privs\}$ 
  for  $i \in mitigate$  prvs do
    if  $p(i) == TRUE$  then
       $pScore[i] \leftarrow pScore[i] + mitigate[i]$ 
    end if
  end for
return  $pScore$ 
end function

function SYSTEMSCORE(process  $process$ )
   $endAlg \leftarrow 0$ 
  while  $endAlg == 0$  do
    for all  $i \in process$  do
      if  $Par(i) == null$  and  $Sib_i == null$  then
         $sys_{sco}(i) \leftarrow N_{sco}(i)$ 
      end if
      if  $Par(i) \neq null$  and  $Sib(i) == null$  then
        if  $sys_{sco}(j) \exists \forall j \in Par(i)$  then
           $P_{sco}(i) \leftarrow N_{sco}(i) * \sum_{j \in Par(i)} R_{sco}(j)$ 
           $sys_{sco}(i) \leftarrow P_{sco}(i)$ 
        end if
      end if
      if  $Par(i) == null$  and  $Sib_i \neq null$  then
         $S_{sco}(i) \leftarrow N_{sco}(i)$ 
        if  $S_{sco}(j) \exists \forall j \in sib(i)$  then
           $sys_{sco}(i) \leftarrow N_{sco}(i) * \sum_{j \in sib(i)} S_{sco}(j)$ 
        end if
      end if
      if  $Par(i) \neq null$  and  $Sib_i \neq null$  then
        if  $sys_{sco}(j) \exists \forall j \in Par(i)$  then
           $P_{sco}(i) \leftarrow N_{sco}(i) * \sum_{j \in Par(i)} sys_{sco}(j)$ 
           $N_{sco}(i) \leftarrow P_{sco}(i)$ 
          if  $S_{sco}(j) \exists \forall j \in sib(i)$  then
             $sys_{sco}(i) \leftarrow P_{sco}(i) + N_{sco}(i) * \sum_{j \in sib(i)} S_{sco}(j)$ 
          end if
        end if
      end if
    end for
    if  $\exists sys_{sco}(i) \forall i \in Process$  then
       $endAlg \leftarrow 1$ 
    end if
  end while
return  $sys_{sco}$ 
end function

```

- $P_{Score}(i)$ shows the effect of parents on the process. This score is needed for calculating the system score ($sys_{Score}(i)$) and siblings score ($S_{Score}(i)$). It shows how the parent processes have a propagative impact on the process.
- $S_{Score}(i)$ shows the impact of the process to its siblings. The siblings use this score to evaluate their system score. This score shows how an external process has a propagative impact on the processes that are not connected to it directly.

Moreover, there are two lists for each process, $Parents(i)$ and $Siblings(i)$, which contain the parent processes and sibling processes respectively. Table 3 shows the calculation of the system score ($sys_{Score}(i)$) in each case.

This table demonstrates that when a node does not have any parent or sibling connection, it is not affected by other processes. Therefore, the score of it is calculated using only its properties. In the other case, the process is not connected to an external node. Therefore, the $P_{Score}(i) = 0$ and the propagative impact to other processes is only from its properties that is $N_{Score}(i)$. In the third case, the process has connections with external nodes, but there is not any connection with the internal process. Therefore, the $P_{Score}(i)$ which is the impact of the external nodes is the only metric for calculation of system score ($sys_{Score}(i)$). Since there is no internal connection, there is no sibling and $S_{Score}(i) = 0$. Finally, the general case is when a process connected to both external and internal processes. The $P_{Score}(i)$ which is calculated the same as the previous case shows the impact of the external processes.

Moreover, it is used for $S_{Score}(i)$ to show the impact of the external processes to the internal processes which are not directly connected to them. For calculating the system score, we use the impact of both external processes and internal processes. Algorithm 1 shows the mechanism of calculating the system score. The low sys_{Score} shows that the impact of poorly protected processes is not high on the process. On the other hand, if the sys_{Score} is high, the process is highly affected by propagative impacts of other processes.

In addition to system-level metrics, AHA also produces a harmonic mean score for both the external and internal processes. The harmonic mean provides a stronger emphasis on the lowest valued process in the system and therefore will provide a lower value if any processes provide very low score (i.e., are not properly secured). Therefore, the external harmonic mean demonstrates the processes immediate exposure to remote compromises, while the internal harmonic mean represents the overall level of protection within the platform, but which may not be directly vulnerable to remote compromise.

5 VISUALIZATION

Finally, a visualization component is provided to help provide a broad picture of the system's attack surface. The visualization demonstrates the graph previously identified along with connectivity to external systems. Figure 3 provides an overview of the GUI. On the left are example connections to external systems and also remotely listening ports, which are highlighted red as they are assumed to be untrusted. The nodes on the right represent the processes running with the target platform and have colors ranging from green to red based on the previously discussed score. The

visualization displays all the computed scores, including the harmonic means. Furthermore, it also has an inspector agent, which can be used to display the individual features of each process which were used to compute the attack surface metrics, such as the set of deployed security mechanisms, privileges, connections, and their resulting scores.

6 VALIDATION

AHA has been evaluated on a variety of modern software platforms commonly used within the electric power grid. This area is especially in need of tools to help analyze system attack surface as it heavily depends on software platforms that integrate legacy software. These platforms typically lack modern security protections, though such properties are often difficult for utilities to identify and assess.

This section will introduce five case studies of AHA's analysis of these software platforms to demonstrate both their individual process scores and system-level scores. Table 4, shows these five different platforms and the associated operating systems. The first three platforms are common control center platforms, while, the two last platforms are historian software platforms. Different platforms will be evaluated on different operating systems to demonstrate how the different operating system capabilities (i.e., improved exploit mitigation mechanisms) impact the resulting attack surface. Figures 4 to 8 show the visualization of the case studies. The Min sys_{Score} column shows the best process system score for each case study. A process with low system score is protected well and not affected (or affected low) by other processes. The Max sys_{Score} column shows the worst process system score. A process with high system score is poorly protected and/or is affected by propagative impacts of other processes.

The results are shown in Table 4, which suggest that the Control Center Platform B has the worst harmonic mean of scores for both externally accessible and internally accessible processes between all five case studies. On the other hand, the Historian Platform A has the best harmonic mean of scores for both external and internal process, and also maintains a low number of external connections. Furthermore, the worst process of the second case study (Control Center Platform B) has the score of 5, suggesting it does not sufficiently implement security protections. Moreover, with 23 external connections, it presents a very large attack surface.

7 CHALLENGES AND FUTURE WORK

While the previous section demonstrates the result from AHA's analysis of various industry standard ICS software platforms, during this analysis a variety of challenges were discovered. This section will introduce currently known limitations where current work is being done to assess this.

7.1 Platform Uniqueness

The development of independent Windows and Linux agents introduced some interesting differences regarding the security capabilities between the platforms. This section will document these, as while they may be known by many security concerned developers

Figure 3: AHA Graph Visualization

Process _i		Scores		
<i>Parents(i)</i>	<i>Siblings(i)</i>	Parent Score ($P_{Score}(i)$)	Sibling Score ($S_{Score}(i)$)	System Score ($sys_{Score}(i)$)
Null	Null	-	-	$N_{Score}(i)$
Null	not Null	-	$N_{Score}(i)$	$N_{Score}(i) * \sum_{j \in Siblings(i)} S_{Score}(j)$
not Null	Null	$N_{Score}(i) * \sum_{j \in Parents(i)} sys_{Score}(j)$	$P_{Score}(i)$	$P_{Score}(i)$
not Null	not Null	$N_{Score}(i) * \sum_{j \in Parents(i)} sys_{Score}(j)$	$P_{Score}(i)$	$P_{Score}(i) + N_{Score}(i) * \sum_{j \in Siblings(i)} S_{Score}(j)$

Table 3: System Score Calculation

Platform	# Processes	Harmonic Mean of scores		Min sys_{score}	Max sys_{score}
		Externally accessible	Internally accessible		
Control Center Platform A (Windows Server 2016)	12	38.53	74.78	0.068	1.859
Control Center Platform B (Windows server 2008R2)	43	9.53	8.22	0.177	6.690
Control Center Platform C (Windows Server 2016)	38	29.44	55.55	0.034	3.630
Historian Platform A	14	80	80	0.034	1.859
Historian Platform B	25	70.94	62.22	0.017	2.988

Table 4: Case Studies of AHA Analysis on Industry-Standard ICS Software Platforms

Figure 4: Historian Platform A

Figure 5: Historian Platform B

Figure 6: Control Center Platform C

and administrators, there seems to be insufficient resources dedicated to the identification of these differences and their potential impact to the development of secure platforms.

- (1) Code Signing: While code signing is a critical security mechanism to ensure executed software has a verifiable

providence, the infrastructure to validate code signing is inconsistent across platforms. Windows currently supports this through Authenticode, while most Linux distributions do not implement this by default. While many Linux distributions have mechanisms to enable an authenticated downloaded and installation of binaries (e.g., APT, YUM),

Figure 7: Control Center Platform B

this only protects the initial installation and does not prevent future tampering of binary. Furthermore, while code-signing validate tools are available to Linux (e.g. chktrust), the most existing binaries lack the signing from the source and therefore this cannot be fully validated.

- (2) Control Flow Integrity (CFI, CFG, etc): CFI has been discussed in academia, but its application varies widely. While modern Microsoft Windows systems implement CFG, Linux based implementations are primarily compiler-based, which complicates the assessment methodology.

7.2 Agent Data Collection

The transient nature of many processes and communications introduce a challenge for the data collection process and provides a limitation of the current approaches ability to produce an exhaustive description of system functions. To address this, the agent is being expanded to continually analyze the process execution and connectivity to provide a complete view of the processes and connection. To address this, it implements a continual scan to attempt to identify new processes or connections. While the scan rate can be customized, to identify connections, it can use the TCP TimeWait (e.g., 90 seconds) to ensure it does not miss critical connections between processes.

8 CONCLUSION

Modern ICS systems must face growing cyber threats and are increasingly interconnected to external systems. Furthermore, many applications depend on both out-of-date operating systems or legacy binaries, which do not support modern security mechanisms. This paper introduces, AHA, a tool to help assess the attack surface of software platforms through the analysis of the system's connectivity, along with the completeness of the security mechanisms integrated into the process to protect it from exploitation. AHA provides both a lightweight scraper to collect data from both Windows and Linux environments, along with a graphic interface to which computes a variety of attack surface analysis metrics based on a per-process and per-host level to enable the comparison of attack surface between multiple platforms. To demonstrate the benefit of AHA, the tools has been executed on a number of

Figure 8: Control Center Platform A

anonymized ICS software platforms to demonstrate its ability to detect poorly secured executables within this platform and also to demonstrate its ability to help system operators easily compare the security posture of competing platforms. Finally, while AHA is still a work-in-progress, the paper discusses some current limitations and planned future directions for its development, along with some other platform level oddities which were discovered during the development and testing process.

ACKNOWLEDGEMENTS

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000830. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

REFERENCES

- [1] Github. Attacksurface host analyzer (aha). [Online]. Available: <https://aha-project.github.io>
- [2] D. of Homeland Security. Cyber security evaluation tool (cset). [Online]. Available: <https://cset.inl.gov/SitePages/Home.aspx>
- [3] Microsoft. (2012) Attack surface analyzer. [Online]. Available: <https://www.microsoft.com/en-us/download/confirmation.aspx?id=24487>
- [4] D. of Homeland Security/Morgridge Institute for Research. Software assurance marketplace (swamp). [Online]. Available: <https://www.mir-swamp.org/>
- [5] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 371–386, May 2011.
- [6] M. Howard, J. Pincus, and J. M. Wing, "Measuring relative attack surfaces," *Proceedings of Workshop on Advanced Developments in Software and Systems Security*, December 2003.
- [7] A. Kurmus, R. Tartler, D. Dorneanu, B. Heinloth, V. Rothberg, A. Ruprecht, and D. Lohmann, "Attack surface metrics and automated compile-time os kernel tailoring," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2013.
- [8] E. Osterweil, D. McPherson, and L. Zhang, "The shape and size of threats: Defining a networked system's attack surface," in *2014 IEEE 22nd International Conference on Network Protocols*, Oct 2014, pp. 636–641.
- [9] S. Zatzko, T. Carstens, P. Stach, P. Thompson, and Mudge. (2018, January) How risk is the software you use. [Online]. Available: <https://shmoo18.cyber-itsl.org/>
- [10] Microsoft. What is the security development lifecycle ? [Online]. Available: <https://www.microsoft.com/en-us/sdl/>
- [11] NirSoft. Currports. [Online]. Available: <https://www.nirsoft.net/utils/cports.html>
- [12] NetSPI. Peseurity. [Online]. Available: <https://github.com/NetSPI/PESecurity/>