Container Intrusions: Assessing the Efficacy of Intrusion Detection and Analysis Methods for Linux Container Environments

GIAC GCIA Gold Certification and RES 5500

Author: Alfredo Hickman, ahusmc@yahoo.com
Advisor: Sally Vandeven

Accepted: 12/29/2017

Abstract

The unique and intrinsic methods by which Linux application containers are created, deployed, networked, and operated do not lend themselves well to the conventional application of methods for conducting intrusion detection and analysis in traditional physical and virtual machine networks. While similarities exist in some of the methods used to perform intrusion detection and analysis in conventional networks as compared to container networks, the effectiveness between the two has not been thoroughly measured and assessed: this presents a gap in application container security knowledge. By researching the efficacy of these methods as implemented in container networks compared to traditional networks, this research will provide empirical evidence to identify the gap, and provide data useful for identifying and developing new and more effective methods to secure application container networks

1. Introduction

This research systematically assesses the efficacy of intrusion detection and analysis methods as applied to Docker Linux application container environments compared to the effectiveness of similar methods applied in traditional networks. Linux application container technologies can provide many benefits, but can also introduce complexity and vulnerabilities. Furthermore, the means and methods for securing container environments are young and not evolving as rapidly as the container technologies themselves. With the rapid evolution and adoption of Linux application container technologies in the enterprise, not much scholarly research exists on how to balance the benefits that containers provide with the vulnerabilities that they introduce.

The unique and intrinsic methods by which Linux application containers are created, deployed, networked, and operated do not lend themselves well to the conventional application of methods for conducting intrusion detection and analysis in traditional physical and virtual machine networks. While similarities exist in some of the ways used to perform intrusion detection and analysis in conventional networks as compared to container networks, the effectiveness between the two has not been measured systematically and analyzed: this presents a gap in application container security knowledge. By researching the efficacy of intrusion detection and analysis methods as implemented in container networks compared to traditional networks, this research will provide empirical evidence to identify the gap, and provide data useful for conceiving and developing new and effective methods to secure container networks

As such this research will attempt to answer the following question: How effective are methods for conducting intrusion detection and analysis in Docker Linux application container networks when compared with the efficacy of similar methods in traditional networks?

2. Linux Containers and Docker – a Brief History

Linux application containers as they are known today, are most directly the product of Cgroups, which was formally introduced in the Linux Operating System Kernel version 2.6.24, in 2007 (Bottomley & Emelyanov, 2014). At a high-level, Linux application containers are lightweight virtual machines that share the same underlying operating system kernel, consume the same or shared resources, and contain the code, tools, dependencies, and settings required to function. Due to the benefit of containerized applications sharing the same underlying host kernel, container hosts can reach a much higher deployment density than traditional dedicated application or virtual machine hosts. In addition to the application deployment density benefit, Linux container deployments also benefit from a shared kernel, with application dependencies residing within the individual containers. This benefit allows developers, I.T. operators, and system owners to reduce the equipment, software, and operational overhead required to service application workloads and their associated costs: Both reasons are why Linux application container technologies have soared in popularity over the last few years (Mohallel, Bass, & Dehghantaha, 2016).

In 1979, Bell Laboratories released Unix v7, which introduced chroot into the Unix ecosystem. Change root, or chroot for short, gives the operating system the capability to change the logical root directory of a running process and its child processes to isolate the processes from awareness and access to neighboring processes and resources. The chroot feature enables efficient and more secure practices for application context isolation and testing, and it set the conceptual stage for Linux application containers almost 30 years later. After the development of chroot in 1979, it was not until the early 2000's that new applications of process isolation and control, more resembling the current Linux application containers we know today, began to emerge. Systems such as FreeBSD Jails, Linux VServer, Solaris Zones, and others set the stage for contemporary Linux application container and management technologies such as LXC, RKT, Kubernetes, and most famously, Docker (Petazonni, 2015).

While relatively new to the mainstream, Linux application containers have been around since 2006, when Google developers, Paul Menage and Rohit Seth developed Control Groups (cgroups). Control Groups are a Linux kernel feature that enables group process management and accounting. Another critical and foundational technology that allowed the creation of modern Linux application containers is Linux namespaces. Linux namespaces, introduced in the Linux Kernel version 2.4.19, while similar to cgroups that came after it, is different and complementary to cgroups. Namespaces serve to isolate groups of processes into logical units that are restricted to the unit and limited in their interaction with and consumption of host system resources (Bottomley & Emelyanov, 2014). In essence, the foundational technologies that enable Linux application containers are cgroups for resource consumption management and accounting, and namespaces for logical partitioning and regulation of host resource access and consumption.

A lot has changed in the Linux application container world since the development of chroot and the adoption of namespaces and cgroups into the Linux kernel. Moreover, while still relatively new in the general enterprise, companies such as Google, AWS, and Facebook have been using containers for the better part of a decade (Winkel, 2016). So, since Linux application containers have been around for years, why are we only now seeing the general adoption of the technology into the enterprise? The likely answer to that is - Docker. Solomon Hykes, launched the Docker project while working as an engineer at dotCloud in France. Hykes realized that while Linux application containers were readily available and decently mature for production implementation, the technologies were overly complicated and not yet palatable for general enterprise adoption (Hykes, 2013).

At Pycon 2013, with this realization in mind, Hykes released Docker for open source distribution. At a high-level, Docker is a Linux application container management system which abstracts away much of the complexities associated with containerized application development and host infrastructure operations (Mohallel, Bass, & Dehghantaha, 2016). However, since the public release of Docker and with the associated and significant increase in its development and adoption in the enterprise, many vulnerabilities in the underlying and related technologies have been discovered. Furthermore, the complexity associated with developing, delivering, deploying, and operating containerized applications and their host infrastructures have introduced new challenges and paradigms in the way that security professionals secure such environments.

2.1. Linux Container Security – a New World

The unique methods by which application containers are created, deployed, networked, and operated present unique challenges when designing, implementing, and operating security systems for these environments. Due to the frequent practice of binding containers to non-standard network ports, deploying application workloads dynamically over distributed hosts, integrating rapidly evolving application code on containers in production, and having specific container instances provisioned for brief periods of times, container technologies have become prime targets for adversary attack and exploitation. Just as the security industry evolved to secure the enterprise during the introduction of computer virtualization, the security industry will need to evolve again, and more rapidly, to secure application container infrastructures if the industry hopes to keep up with the rapid rate of change.

3. Intrusion Detection Systems and Analysis in a Dockerized World

The existing body of scholarly literature related to developing methods and systems for conducting intrusion detection and analysis in application container networks is limited. However, there does exist a body of foundational scholarly research and literature in application container security, intrusion detection methods, and analysis on which to build. Furthermore, there are many sources available related to traditional methods and systems for conducting intrusion detection and analysis to compare to new and evolving techniques used in container networks.

For instance, Abed, Clancy, and Levy, 2015 found that due to the way container technologies enable a single host operating system kernel to provide resources to containers, attacks on the container based-applications could result in compromises of the container hosts, other containers, and even other networks. With this realization, Abed et al. proposed the adaptation of the Bag of System Calls (BoSCs) method, sometimes used in traditional host-based intrusion detection, to create a container-based host intrusion detection system. The technique that Abed and team developed requires the monitoring of system call frequency between individual container processes and the host operating system kernels for anomaly detection (2015). By recording the frequency of system calls between the container host kernel and container processes, the BoSC system could learn what normal system call behavior is and then identify statistical deviations from normal to identify anomalous and potentially malicious behavior.

Such adaptations of existing methods for conducting intrusion detection and analysis in traditional networks to container networks is an emerging and promising trend in container security. OSSEC is one traditional HIDS that can leverage the Linux Audit logic to parse system calls and enable BoSC implementations. Such adapted methods aim to port proven security methods to mitigate emerging threats and vulnerabilities that, while not unique to container deployments, are only exacerbated by the typically high volume and speed in which containers are deployed and operated.

Vulnerabilities such as the kind that Gao et al., 2017 discovered indicate how incomplete and ineffective methods used for partitioning and allocating host operating system kernel resources to application containers in multi-tenant cloud environments resulted in information leakages. In the Synergistic-Power Attack proof-of-concept, the authors demonstrated how attackers could use aggregated container, and container host leaked data to potentially orchestrate a distributed power spike attack in a multi-tenant container-cloud to cause power supply faults and electrical outages in a data center.

With the research that Gao et al., 2017 conducted, intrusion detection and analysis methods could be created to detect the behavior associated with container and container hosts information leakage, and the techniques, tactics, and procedures (TTP), that an attacker would use to conduct the Synergistic-Power Attack. For example, a BoSC based system could monitor the system calls associated with information leakage between a Docker host and container to identify such vulnerabilities.

The adaptation, creation, and implementation of container-centric intrusion detection and analysis methods and systems becomes even more pressing due to research findings which indicate that more vulnerabilities are present in container application deployments than traditional physical or virtual system deployments. For instance, Mohallel, Bass, and Dehghantaha, 2016 conducted quantitative research into how attack surface area differs between applications deployed in traditional physical or virtual machine implementations as compared to container-based implementations. The authors discovered that the amount of vulnerabilities introduced into a container host equals the sum of the vulnerabilities found within the host operating system, the container's base image, and the software packages contained within the containers. The research indicates that deploying applications in containers can increase the number of vulnerabilities present on a container host.

Not only does research indicate an increase in the number of vulnerabilities introduced by application container implementations, but it also shows an increased scope and criticality of the vulnerabilities. For example, Winkel, 2017 found that attackers could exploit vulnerabilities present in versions of the Linux kernel to escape the process, resource, and permissions security mechanisms provided by the operating system to the application container. Like what Abed et al., 2015 discovered, this type of exploit could result in an attacker escaping the container and then exploiting the underlying host system and possibly other systems accessible through the network. While similar escape exploits exist and are detectable in traditional virtual machine environments, due to the unique nature of container networking, resource allocation, and deployment methods, the same is not the case in container environments. The complexity of container technologies and operations, the vulnerabilities associated with the technologies, and the immaturity of available security systems, warrants research into adapted and new means for securing such environments.

In contrast to host-based methods for intrusion detection and analysis, such as BoSC, Winkel proposes a network security monitoring (NSM) approach to collect telemetry and provide forensic visibility to human analysts conducting intrusion detection and analysis in Docker container networks. Colm Kennedy, 2016 also proposes a network-based approach that can adapt to container networks. Kennedy's method calls for using network decoys which mimic production systems to coax would-be attackers to exploit the systems. However, these honeypot decoys would be instrumented and monitored in a manner that would facilitate intrusion detection and analysis.

Complimenting the decoy method, Patrick Neise, 2016 proposes the idea of using network flow and graphs to identify relationships between hosts and events in a network to aid in intrusion detection and analysis. While the networking and deployment methodologies that application container networks employ are significantly different from traditional TCP/IP network implementations, the methods that Neise describes are analogous to sFlow and relational graph (link) analysis based methods that have been employed to gain visibility into Docker container networks.

As such, network-based intrusion detection and analysis methods such as implementing decoys, flow analysis, and relational graph (link) analysis provide analogous examples to host-based methods such as BoSC and kernel system call tapping. Also, both host and network-based techniques lend themselves well to building container-based intrusion detections systems and comparing the efficacy between their analogous implementations in traditional networks.

This literature review represents some of the latest research in methods for detecting data leakage, anomalous behavior, vulnerabilities, and exploitation methods in container based environments. Furthermore, the non-container related literature reviewed here represents practices that can and have been adapted to create application container security systems.

3.1. Intrusion Detection and Analysis in Traditional and Virtual Networks (Normal IDS & A)

Much literature exists about intrusion detection and analysis in traditional physical and virtual networks. At a high-level, the two standard, mature, and capable approaches to the practice are network-based and host-based intrusion detection and analysis. Tracing their conceptual origins to events in 1986, computer network intrusion detection and analysis gained prominence when Cliff Stoll, a systems manager at the Lawrence Berkeley National Laboratory, a U.S. government research facility, noticed financial discrepancies in an accounting system. This incident resulted in a dramatic investigation which discovered that the accounting discrepancies were not due to flawed computer logic or an accident by a human accountant, but were due to coordinated intrusions by a foreign state-sponsored agent (Bejtlich, 2013). The event is relevant the practice of intrusion detection and analysis in that it served to raise awareness at the highest levels of the U.S. government to the importance of securing sensitive computer networks and developing national strategic capabilities for conducting computer network defense and offense. In many ways, the events at the Berkeley Lab in 1986 spawned the intrusion detection and analysis industry that we know today (Bejtlich, 2013). Moreover, while much has changed in intrusion detection and analysis since the 1980's, at its core, today's traditional approaches to the practice remain much the same.

At a high-level, modern intrusion detection and analysis systems monitor and assess networks and hosts for patterns and conditions that are indicative of potentially malicious activities and vulnerabilities. Moreover, while the technologies involved in evaluating malicious activities and vulnerabilities have evolved significantly over the years, it is still the predefined or near-real-time definition of malicious activities or vulnerabilities which underpin intrusion detection and analysis methods available today. Even with advances in artificial intelligence, machine learning, and threat information sharing, intrusion detection, and analysis systems rely on patterns of expected normal behavior, definitions of malicious behavior, and identification of deviations from "normal" conditions to identify potential malicious activities and vulnerabilities.

For example, many traditional applications of network intrusion detection and analysis systems are dependent on consistent and pre-defined bindings of an application's network port assignments for analysis. Also, these systems are often reliant on the pre-defined or near-real-time definition of normal or abnormal network or host activities. These systems will then match signatures against associated events or identify deviations from normal conditional thresholds to produce alerts or automated responses (Bejtlich, 2013). It is easy to see that in environments where what is "normal" for one instance of a provisioned application that may only exist for minutes and be configured with non-standard network port bindings presents severe challenges to the traditional network and host intrusion detection and analysis paradigms. With the advent of Linux containerized application deployments, that is usually the case.

3.2. Intrusion Detection Systems and Analysis in Dockerized Networks

As is often the case in Linux application container deployments, application instances and the containers that host them exist for short periods of time and are regularly provisioned with non-standard network port assignments bound to the underlying host. Furthermore, with best practices for deploying containerized applications calling for microservice architectures, one application deployment could require the provisioning of tens of containers to service the overall application (Hayden, 2015). Microservice architectures in container deployments require that individual services be provisioned one per container and grouped in a logical manner that facilitates services to the whole application instance and its dependencies (Winkel, 2016).

The idea behind microservices architectures in Linux application container networks is to limit the interaction between adjacent services, to continuously deploy and improve the individual services, and to scale resources as required more efficiently. However, it is in many ways the adoption of microservice architectures and the complexity and variance that they introduce into the network which exacerbates the already challenging nature of monitoring and securing Linux application container networks. However, the value that application containers provide, coupled with the vulnerabilities and challenges that the technologies introduce have correspondingly stimulated the evolution of the security industry.

4. Research Methods - a Tale of Two IDSs

For this research, attack, analysis, and capability experiments were conducted in a lab to assess the efficacy of intrusion detection and analysis capabilities in Docker container networks compared to the effectiveness of similar methods in traditional networks. The lab consists of a single network with deployments of both traditional and container-centric intrusion detection systems. The tests were conducted on Ubuntu 16.04 LTS hosts. All hosts were up to date at the time of the experimentation and were instrumented with the OSSEC HIDS and a Splunk universal forwarder. The OSSEC HIDS configurations are identical across all the implementations and have log, malware, and file integrity monitoring enabled. The Splunk universal forwarders are configured with all default inputs enabled and to transmit syslog to a Splunk unified indexer and search head for collection and analysis. On Docker container deployments, the Monitoring Docker Splunk App, installed on the Splunk forwarder, facilitates inter-container and host telemetry collection.

All test hosts serve the Damn Vulnerable Web App (DVWA), which will be the primary target for assessing the efficacy of the various intrusion detection and analysis systems. Furthermore, Security Onion 14.04 is deployed in the lab with the Snort NIDS, OSSEC HIDS, Bro for traffic monitoring, and ELSA, Squert, Wireshark, and associated tools for analysis. Security Onion enables the efficacy assessments of the intrusion detection and analysis experiments conducted in the traditional application host environment, as well as the application of traditional NIDS and HIDS in the Docker host and containerized application environment. In implementations covered by Security Onion, the Snort NIDS and OSSEC HIDS configurations are identical and have all rules enabled. Wazuh with the OSSEC HIDS and Sysdig Falco with the falco-probe host kernel module, for tapping and assessing Linux container host and intra-container activities, enable the efficacy assessments of the container intrusion detection and analysis use cases.

Once the lab infrastructure was deployed and configured, the attack experiments were conducted from a Kali Linux host. The attack experiments represent various phases of the Cyber Kill-Chain (Lockheed Martin), and they serve to assess the intrusion detection and analysis capabilities of the various systems. The attack types, test cases, and required capabilities are located in the appendix. Testing artifacts were collected from the various intrusion detection and analysis systems. The artifacts and testing results serve to measure the effectiveness and capabilities of the multiple systems to detect and enable analysis of the various attacks and intrusions.

4.1. Effectiveness Criteria

The effectiveness of the various intrusion detection and analysis systems are measured against the following criteria and associated test cases: **Note**: The associated test cases are located in the appendix.

1. Detection of scanning activity
2. Detection of application attacks
3. Detection of malware deployment
4. Detection of malware execution
5. Detection of malicious command and control
6. Detection of malicious privilege escalation
7. Detection of malicious data exfiltration
8. Detection of file integrity violations
9. Detection of leaked system data
10. Auto-detection of anomalous behavior
11. Auto-detection of attacker, victim, infrastructure relationship
12. Capability for forensic artifact retrieval (PCAP, Flow, Logs,)

4.2. Measurement Criteria

A scoring system is used to measure the effectiveness of the intrusion detection systems to detect and provide analysis capabilities of the associated test case experiments. Each test case experiment will have a maximum of three points awarded. Points are weighted as follows:

One Point: Not Effective (Method did not work).

Two Points: Moderately Effective (Method worked, but did not allow for complete functionality, or equivalent to traditional network implementation).

Three Points: Effective (Method worked as effectively as traditional network implementation).

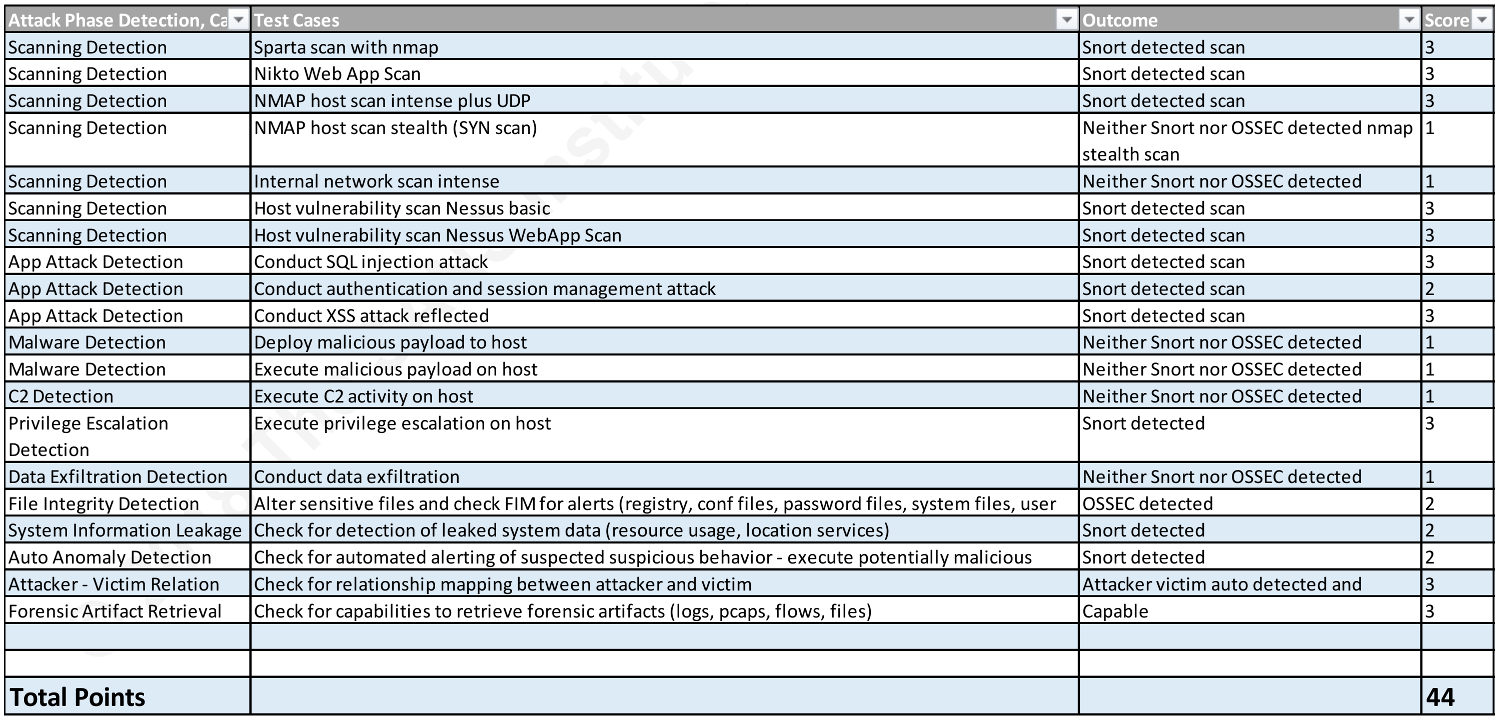
The point-based measurements of effectiveness will describe the efficacy of intrusion detection and analysis methods as applied in container networks, compared to the effectiveness of similar methods employed in traditional networks. Also, the findings of this research and the scoring of the effectiveness criteria could aid in the identification and development of new methods for securing container networks.

5. Research Findings – the Answers to Life, the Universe, and Everything

The following are the effectiveness results and analysis of the various intrusion detection and analysis methods assessed. **Note:** Where applicable, the NIDS and HISD configurations are identical and vary only in implementation or capabilities provided by the analysis platforms, such as Security Onion, Splunk, or Wazuh.

Table 1.

Damn Vulnerable Web App Hosted on Traditional Virtual Machine and Protected by Security Onion.



Attack Phase Detection, Ca	Test Cases	Outcome	Score
Scanning Detection	Sparta scan with nmap	Snort detected scan	3
Scanning Detection	Nikto Web App Scan	Snort detected scan	3
Scanning Detection	NMAP host scan intense plus UDP	Snort detected scan	3
Scanning Detection	NMAP host scan stealth (SYN scan)	Neither Snort nor OSSEC detected nmap stealth scan	1
Scanning Detection	Internal network scan intense	Neither Snort nor OSSEC detected	1
Scanning Detection	Host vulnerability scan Nessus basic	Snort detected scan	3
Scanning Detection	Host vulnerability scan Nessus WebApp Scan	Snort detected scan	3
App Attack Detection	Conduct SQL injection attack	Snort detected scan	3
App Attack Detection	Conduct authentication and session management attack	Snort detected scan	2
App Attack Detection	Conduct XSS attack reflected	Snort detected scan	3
Malware Detection	Deploy malicious payload to host	Neither Snort nor OSSEC detected	1
Malware Detection	Execute malicious payload on host	Neither Snort nor OSSEC detected	1
C2 Detection	Execute C2 activity on host	Neither Snort nor OSSEC detected	1
Privilege Escalation Detection	Execute privilege escalation on host	Snort detected	3
Data Exfiltration Detection	Conduct data exfiltration	Neither Snort nor OSSEC detected	1
File Integrity Detection	Alter sensitive files and check FIM for alerts (registry, conf files, password files, system files, user	OSSEC detected	2
System Information Leakage	Check for detection of leaked system data (resource usage, location services)	Snort detected	2
Auto Anomaly Detection	Check for automated alerting of suspected suspicious behavior - execute potentially malicious	Snort detected	2
Attacker - Victim Relation	Check for relationship mapping between attacker and victim	Attacker victim auto detected and	3
Forensic Artifact Retrieval	Check for capabilities to retrieve forensic artifacts (logs, pcaps, flows, files)	Capable	3
Total Points			**44**

In this use case, Security Onion was deployed with the Snort network-based intrusion detection system with the Emerging Threats ruleset completely enabled, and the OSSEC host-based intrusion detection system on the protected virtual machine application host.

For the scanning portion of the tests, Snort detected all but the Nmap stealth and network range scans. OSSEC did not detect the Nessus host and web application vulnerability scans, or Nmap scans during the software service and version enumeration portions of the scans.

For the attack portion of the tests, Snort detected all the attacks. However, Snort only detected the authentication and session management attack via the curl detection policy which triggered when curl was used to pull the session ID token from DVWA. For this, I subtracted one point. OSSEC did not detect any of the attacks.

For the malware portion of the tests, neither Snort nor OSSEC detected the downloading of the EICAR test file nor the execution of the EICAR payload in a shell script.

Neither Snort nor OSSEC detected the command and control activities that were conducted on the victim host using both SSH and Netcat.

Snort detected privilege escalation. OSSEC did not detect privilege escalation attempts on the victim host.

Neither Snort nor OSSEC detected the exfiltration of the passwd and shadow files from the protected /etc/ directory.

OSSEC detected file integrity modifications in protected directories. However, one point was subtracted due to Security Onion not surfacing the alerts automatically or in real time via Sguil. Hunting was required to find the associated alerts in ELSA. Snort did not detect file integrity attacks.

Snort detected the leakage of certain system information such as software names and version numbers. However, one point was subtracted due to Security Onion not surfacing the alerts automatically or in real-time via Sguil. Hunting was required to find the associated alerts in ELSA. OSSEC did not detect system information leakage.

Sguil automatically surfaced Snort detections of potentially anomalous behavior. However, one point was subtracted due to Security Onion not surfacing the associated OSSEC alerts automatically or in real time via Sguil. Hunting was required to find the associated alerts in ELSA.

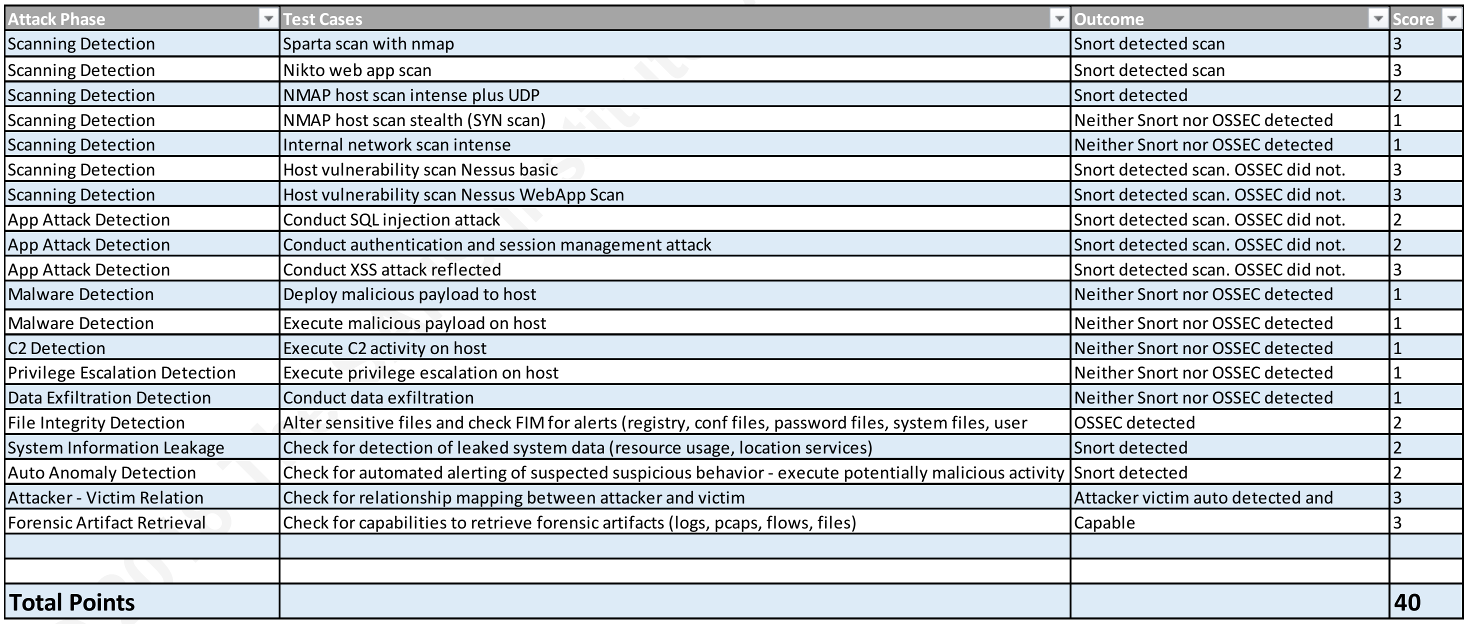
Security Onion was able to efficiently and dynamically depict attacker to victim relationships via collected telemetry.

Security Onion was able to produce logs, pcaps, flow data, and associated files.

Of the intrusion detection and analysis platforms evaluated, Security Onion with the Snort NIDS and OSSEC HIDS deployed to protect a traditional virtual machine application host was the most effective platform and received a score of 44 points.

Table 2.

Damn Vulnerable Web App Hosted in a Docker Container and Protected by Security. Onion.



Attack Phase	Test Cases	Outcome	Score
Scanning Detection	Sparta scan with nmap	Snort detected scan	3
Scanning Detection	Nikto web app scan	Snort detected scan	3
Scanning Detection	NMAP host scan intense plus UDP	Snort detected	2
Scanning Detection	NMAP host scan stealth (SYN scan)	Neither Snort nor OSSEC detected	1
Scanning Detection	Internal network scan intense	Neither Snort nor OSSEC detected	1
Scanning Detection	Host vulnerability scan Nessus basic	Snort detected scan. OSSEC did not.	3
Scanning Detection	Host vulnerability scan Nessus WebApp Scan	Snort detected scan. OSSEC did not.	3
App Attack Detection	Conduct SQL injection attack	Snort detected scan. OSSEC did not.	2
App Attack Detection	Conduct authentication and session management attack	Snort detected scan. OSSEC did not.	2
App Attack Detection	Conduct XSS attack reflected	Snort detected scan. OSSEC did not.	3
Malware Detection	Deploy malicious payload to host	Neither Snort nor OSSEC detected	1
Malware Detection	Execute malicious payload on host	Neither Snort nor OSSEC detected	1
C2 Detection	Execute C2 activity on host	Neither Snort nor OSSEC detected	1
Privilege Escalation Detection	Execute privilege escalation on host	Neither Snort nor OSSEC detected	1
Data Exfiltration Detection	Conduct data exfiltration	Neither Snort nor OSSEC detected	1
File Integrity Detection	Alter sensitive files and check FIM for alerts (registry, conf files, password files, system files, user	OSSEC detected	2
System Information Leakage	Check for detection of leaked system data (resource usage, location services)	Snort detected	2
Auto Anomaly Detection	Check for automated alerting of suspected suspicious behavior - execute potentially malicious activity	Snort detected	2
Attacker - Victim Relation	Check for relationship mapping between attacker and victim	Attacker victim auto detected and	3
Forensic Artifact Retrieval	Check for capabilities to retrieve forensic artifacts (logs, pcaps, flows, files)	Capable	3
Total Points			**40**

In this use case, Security Onion was deployed with the Snort network-based intrusion detection system with the Emerging Threats ruleset completely enabled, and the OSSEC host-based intrusion detection system on the protected Docker application container host.

For the scanning portion of the tests, Snort detected all but the Nmap stealth and network range scans. OSSEC did not detect the Nessus host and web application vulnerability scans, or Nmap scans during the software service and version enumeration portions of the scans.

For the attack portion of the tests, Snort detected all the attacks. However, Snort only detected the authentication and session management attack via the curl detection policy which triggered when curl was used to pull the session ID token from DVWA. Furthermore, Security Onion did not surface the associated SQL injection attack alert automatically or in real time via Sguil. Hunting was required to find the associated alerts in ELSA. For these two deficiencies, one point per attack was deducted. OSSEC did not detect any of the attacks.

For the malware portion of the tests, neither Snort nor OSSEC detected the downloading of the EICAR test file nor the execution of the EICAR payload in a shell script.

Neither Snort nor OSSEC detected the command and control activities that were conducted on the victim host using both SSH and Netcat.

Neither Snort nor OSSEC detected the privilege escalation attempts on the victim host.

Neither Snort nor OSSEC detected the exfiltration of the passwd and shadow files from the protected /etc/ directory.

OSSEC detected file integrity modifications in protected directories. However, one point was subtracted due to Security Onion not surfacing the alerts automatically or in real-time via Sguil. Hunting was required to find the associated alerts in ELSA. Snort did not detect file integrity attacks.

Snort detected the leakage of certain system information such as software names and version numbers. However, one point was subtracted due to Security Onion not surfacing the alerts automatically or in real-time via Sguil. Hunting was required to find the associated alerts in ELSA. OSSEC did not detect system information leakage.

Sguil automatically surfaced Snort detections of potentially anomalous behavior. However, one point was subtracted due to Security Onion not surfacing the associated OSSEC alerts automatically or in real time via Sguil. Hunting was required to find the associated alerts in ELSA.

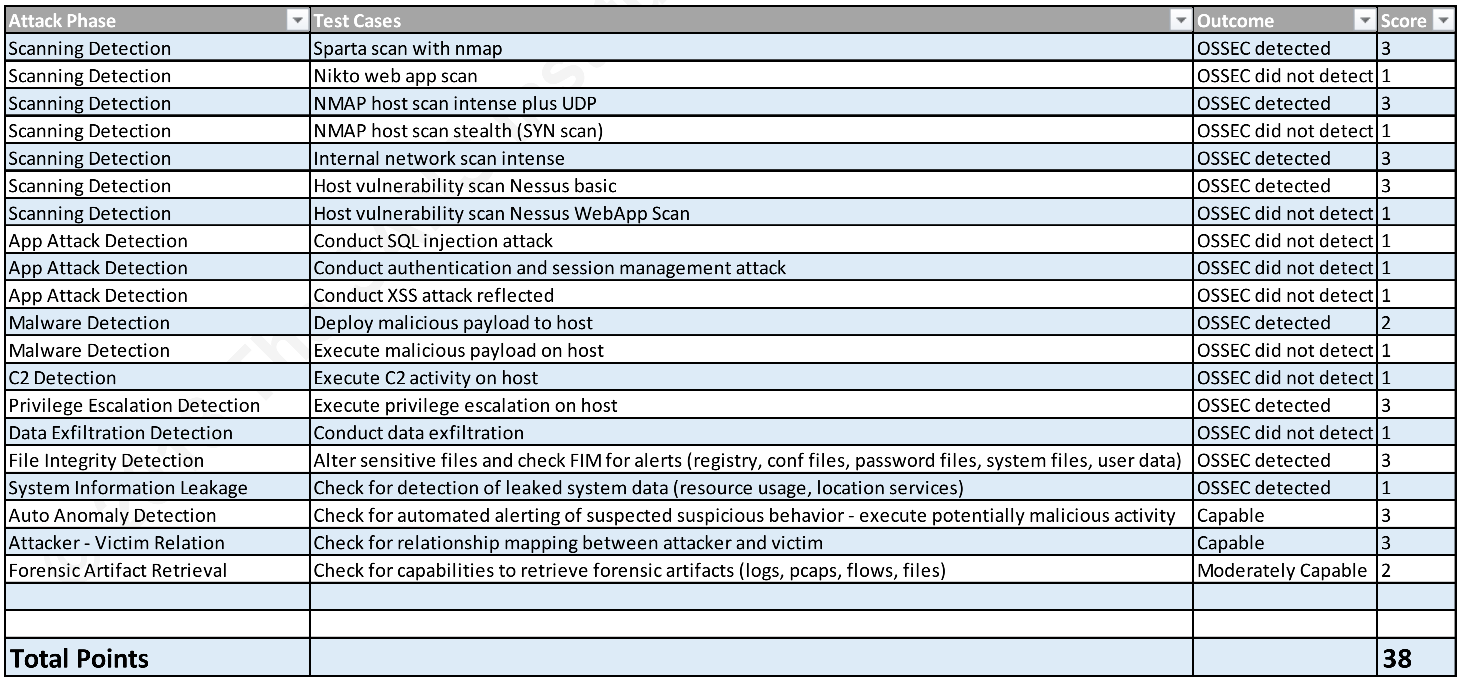
Security Onion was able to efficiently and dynamically depict attacker to victim relationships via collected telemetry.

Security Onion was able to efficiently produce logs, pcaps, flow data, and associated files.

Of the intrusion detection and analysis platforms evaluated, Security Onion with the Snort NIDS and OSSEC HIDS deployed to protect a Docker application container host and workloads was the second most effective platform and received a score of 40 points.

Table 3.

Damn Vulnerable Web App Hosted in a Docker container and Protected by Wazuh.



Attack Phase	Test Cases	Outcome	Score
Scanning Detection	Sparta scan with nmap	OSSEC detected	3
Scanning Detection	Nikto web app scan	OSSEC did not detect	1
Scanning Detection	NMAP host scan intense plus UDP	OSSEC detected	3
Scanning Detection	NMAP host scan stealth (SYN scan)	OSSEC did not detect	1
Scanning Detection	Internal network scan intense	OSSEC detected	3
Scanning Detection	Host vulnerability scan Nessus basic	OSSEC detected	3
Scanning Detection	Host vulnerability scan Nessus WebApp Scan	OSSEC did not detect	1
App Attack Detection	Conduct SQL injection attack	OSSEC did not detect	1
App Attack Detection	Conduct authentication and session management attack	OSSEC did not detect	1
App Attack Detection	Conduct XSS attack reflected	OSSEC did not detect	1
Malware Detection	Deploy malicious payload to host	OSSEC detected	2
Malware Detection	Execute malicious payload on host	OSSEC did not detect	1
C2 Detection	Execute C2 activity on host	OSSEC did not detect	1
Privilege Escalation Detection	Execute privilege escalation on host	OSSEC detected	3
Data Exfiltration Detection	Conduct data exfiltration	OSSEC did not detect	1
File Integrity Detection	Alter sensitive files and check FIM for alerts (registry, conf files, password files, system files, user data)	OSSEC detected	3
System Information Leakage	Check for detection of leaked system data (resource usage, location services)	OSSEC detected	1
Auto Anomaly Detection	Check for automated alerting of suspected suspicious behavior - execute potentially malicious activity	Capable	3
Attacker - Victim Relation	Check for relationship mapping between attacker and victim	Capable	3
Forensic Artifact Retrieval	Check for capabilities to retrieve forensic artifacts (logs, pcaps, flows, files)	Moderately Capable	2
Total Points			**38**

In this use case, Wazuh was deployed with the OSSEC host-based intrusion detection system on the protected Docker application container host, and the Wazuh PCI DSS extension enabled.

For the scanning portion of the tests, OSSEC detected all but the Nikto and Nessus web application scans and the Nmap stealth scan.

For the attack portion of the tests, OSSEC did not detect any of the attacks.

For the malware portion of the tests, OSSEC detected the placement of the EICAR payload shell script in the protected /etc/ directory. However, it is unlikely that OSSEC would have detected, in real time, the test malware file if it was deposited and executed from a non-protected directory. OSSEC, as configured on all the test hosts, conducts daily malware checks.

OSSEC did not detect the execution of the EICAR payload shell script.

OSSEC did not detect the command and control activities that were conducted on the victim host using both SSH and Netcat.

OSSEC detected the privilege escalation attempts on the victim host via the Wazuh PCI DSS extension.

OSSEC did not detect the exfiltration of the passwd and shadow files from the protected /etc/ directory.

OSSEC detected file integrity modifications in protected directories.

OSSEC did not detect the leakage of certain system information such as software names and version numbers.

OSSEC automatically surfaced potentially anomalous behavior.

Wazuh was able to efficiently and dynamically depict attacker to victim relationships via collected telemetry.

Wazuh was only capable of producing limited alert and log reports. Wazuh was unable to produce specific logs, pcaps, flow data, and associated files.

Of the intrusion detection and analysis platforms evaluated, Wazuh with the OSSEC HIDS deployed to protect a Docker application container host and workloads was the least effective platform and received a score of 38 points.

Table 4.

Damn Vulnerable Web App Hosted in a Docker container and Protected by Sysdig Falco.

Attack Phase	Test Cases	Outcome	Score
Scanning Detection	Sparta scan with nmap	Detected	3
Scanning Detection	Nikto web app scan	Detected	3
Scanning Detection	NMAP host scan intense plus UDP	Detected	3
Scanning Detection	NMAP host scan stealth (SYN scan)	Not Detected	1
Scanning Detection	Internal network scan intense	Detected	3
Scanning Detection	Host vulnerability scan Nessus basic	Detected	3
Scanning Detection	Host vulnerability scan Nessus WebApp Scan	Detected	3
App Attack Detection	Conduct SQL injection attack	Detected	2
App Attack Detection	Conduct authentication and session management attack	Detected	2
App Attack Detection	Conduct XSS attack reflected	Detected	2
Malware Detection	Deploy malicious payload to host	Not Detected	1
Malware Detection	Execute malicious payload on host	Not Detected	1
C2 Detection	Execute C2 activity on host	Not Detected	1
Privilege Escalation Detection	Execute privilege escalation on host	Detected	3
Data Exfiltration Detection	Conduct data exfiltration	Not Detected	1
File Integrity Detection	Alter sensitive files and check FIM for alerts (registry, conf files, password files, system files, user data)	Detected	3
System Information Leakage	Check for detection of leaked system data (resource usage, location services)	Not Detected	1
Auto Anomaly Detection	Check for automated alerting of suspected suspicious behavior - execute potentially malicious activity	Detected	3
Attacker - Victim Relation	Check for relationship mapping between attacker and victim	Detected	2
Forensic Artifact Retrieval	Check for capabilities to retrieve forensic artifacts (logs, pcaps, flows, files)	Capable	2
Total Points			**43**

In this use case, Sysdig Falco was deployed with the falco-probe Linux kernel module on the Docker host. The falco-probe kernel module facilitates the tapping of bi-directional container host to container and container to container system call communications. Furthermore, Falco is a headless application that can surface alerts to numerous output destinations such as standard output, syslog, flat files, and local programs. In this use case, Falco alerts, and telemetry was sent to a central Splunk instance via a Splunk universal forwarder and the Monitoring Docker Splunk app installed on the test host. All intrusion analysis was done via Splunk.

For the scanning portion of the tests, Falco detected all but the Nmap stealth scan.

Falco did not detect any of the attacks. One point was subtracted per test case due to the alerts surfacing through log management capabilities in the Monitoring Docker Splunk app used in the falco implementation.

Falco did not detect any of the malware test cases.

Alfredo Hickman, ahusmc@yahoo.com

Falco did not detect the command and control activities that were conducted on the victim host using both SSH and Netcat.

Falco detected the privilege escalation attempts on the victim host.

Falco did not detect the exfiltration of the passwd and shadow files from the protected /etc/ directory.

Falco detected file integrity modifications in protected directories.

Falco did not detect the leakage of certain system information such as software names and version numbers.

Falco automatically surfaced potentially anomalous behavior.

Falco was not able to efficiently and dynamically depict attacker to victim relationships. One point was subtracted due to associated correlations surfacing through the log management capabilities in the Monitoring Docker Splunk app used in the falco implementation.

Falco was only capable of producing limited alert and log reports. One point was subtracted due to Falco's inability to produce specific logs, pcaps, flow data, and associated files.

Of the intrusion detection and analysis platforms evaluated, Sysdig Falco with the falco-probe kernel module and Monitoring Docker for Splunk app deployed to protect a Docker application container host and workloads was the most effective platform and received a score of 43 points.

6. What Now – Recommendations and Implications for Security and a Better Tomorrow

The research presented in this paper indicates that while technology can do much to enable security, it can also do much to hinder security and introduce vulnerabilities. As such, experienced security professionals skilled in their tools, tactics, and procedures are paramount to security. Defense in depth is still critical to security. This research indicates that no one security technology, nor single security platform can detect all the attacks, vulnerabilities, and threats to an environment.

Capability, capacity, configuration, and implementation architecture define security coverage. If the security tooling deployed and implemented is incapable, misconfigured, or deployed in a position of incomplete coverage, it will not be effective. Furthermore, exclusive reliance on the fidelity and capability of security tooling to prevent, detect, and surface all attacks, vulnerabilities, and threats present in an environment, even if correctly configured and implemented, is unrealistic and unwise. Proactive threat hunting and centralized log management are required to mitigate the tool capability gap. The capability gap was demonstrated in the research in instances where attack experiments resulted in telemetry that was not surfaced as an alert in the security tooling user interfaces but instead was detected in the SIEM or NSM.

Vulnerability assessments of application containers and their associated images are essential to overall container environment security. By integrating purpose-built container and image vulnerability scanning into the continuous integration and continuous deployment (CI/CD) pipeline, security professionals can dynamically detect when vulnerabilities are introduced into the images used to create containers and into the software packages, application logic, and dependencies used when presenting the applications. With this capability, security professionals can then remediate or mitigate the discovered vulnerabilities.

6.1. Recommendations for More Effective IDS solutions in Application Container Environments

Hardening, instrumenting, monitoring, and segmenting application container hosts and management platforms are critical to container environment security. The Center for

Internet Security publishes security configuration benchmarks for the most common Linux operating systems and web servers used in container implementations. Furthermore, CIS also published benchmarks for both the community and enterprise versions of Docker. The CIS benchmarks are located here: https://www.cisecurity.org/cis-benchmarks/

This research indicates that instrumenting application container hosts with security tooling is critical. As such, host-based systems such as Sysdig Falco with its Linux kernel module that can monitor system calls between the host and containers to detect malicious activities is key to container environment security. The research also indicates that monitoring application container hosts with non-kernel module HIDS, such as those relying on Linux Audit, is also useful. However, in-depth analysis of container host and intra-container communications are only possible with kernel level tapping modules.

Hand-in-hand with proper instrumentation is active monitoring of container environments by experienced and skilled security professionals. Application container deployments introduce even more complexity and telemetry into environments than traditional network implementations. Furthermore, as described in the research findings, even when telemetry is generated in container networks and ingested into security platforms, alerts are not guaranteed to be produced or surfaced. In these instances, hunting conducted by security professionals is crucial to the prevention, detection, alerting, response, and remediation of associated vulnerabilities, threats, attacks, and intrusions.

Appropriate segmentation of application container networks can also assist in intrusion detection and analysis. Due to the typically high deployment densities of containerized applications on hosts, and the complex orchestration of containerized workloads, non-standard network port assignments are common in container environments. This complexity makes traditional network firewall and intrusion detection impractical for securing individual containerized workloads. However, segmenting application container hosts within secured networks and then deploying traditional network firewalls and intrusion detection systems can aid in securing the overall container environment by restricting access to the network and alerting when unusual activity occurs. Furthermore, implementing container-aware web application firewalls that can dynamically associate container instances with application traffic and network port assignments can help overall security.

6.2. Implications for Future Research

The practice of application container security is ripe for research. For instance, one of the most recent and compelling technologies developed to secure web applications is RASP, or runtime application self-protection. RASP is built into the application and is executed at runtime allowing for the detection and response of malicious activities at the application layer. At this time, RASP technologies are restricted to web application deployments based a limited set of webservers and custom application runtime environments. However, RASP technology is promising and developing rapidly. Furthermore, RASP applied to containerized applications is nascent and prime for development. RASP presents exciting and potentially valuable opportunities for future research.

Container network-based intrusion detection is also prime for future research. By solving for dynamic application container behavior profiling and network application port mapping, advances in container firewalls have set the stage for the development of container NIDS. Especially compelling is the potential value in combining data and information gained from container HIDS, with container network security telemetry generated by application and network aware container firewalls, to facilitate the development of container NIDS.

Another point of future research is the development of machine-learning applications to facilitate the development of active container intrusion detection and analysis systems. The dynamic nature of containerized application development and operations makes securing these environments difficult, especially when operating under traditional security paradigms. As such, automation provided by machine learning can augment security operations. Methods, such as Bag of System Calls, briefly covered in this research, can provide such assistance. Using machine learning systems such as

BoSC, security tooling and procedures can be developed to automatically detect, alert, and respond to unusual and potentially malicious activities and conditions.

7. Conclusion

Application container technologies are evolving rapidly, their adoption into the enterprise is soaring, and the implementation use cases are growing in proportion, criticality, and complexity. Furthermore, the vulnerabilities introduced by application container implementations and the attacks being developed to exploit the vulnerabilities are also evolving rapidly. Combine this landscape with the rapid digital transformation of business processes and the widespread adoption of public cloud technologies, commonly used to host containerized applications, and the necessity to develop effective container intrusion detection and analysis systems become evident. As the research suggests, no one security platform was able to secure the whole container environment. It appears that securing application container environments both at the network and at the host-level is key to effective security. Furthermore, centralized collection and analysis of container network and host telemetry were beneficial to the security of the environments tested.

The research presented here is limited to assessing the effectiveness of methods for conducting intrusion detection and analysis in Docker Linux application container networks when compared with the efficacy of similar methods in traditional networks. For this purpose, Security Onion with the familiar Snort NIDS and OSSEC HIDS, Wazuh with the OSSEC HIDS, and Sydig Falco, with its kernel tapping module were selected. This research attempts to remove biases by scoring against absolute effectiveness, absolute ineffectiveness, and moderate effectiveness. However, moderate effectiveness can be judged subjectively due to the assessor's definition of the term. While not exhaustive, this research presents experiments which are representative of typical attack types depicted in the Cyber Kill-Chain. Furthermore, the techniques and tools utilized during the experiments are representative of those commonly used by security professionals when plying their trade. In sum, this research aims to identify gaps in current knowledge and capabilities available to secure application container networks and to spur the development of new research, techniques, and technologies to secure such environments.

References

Abed, A. S., Clancy, C., & Levy, D. S. (2015). Intrusion Detection System for Applications Using Linux Containers. *Security and Trust Management Lecture Notes in Computer Science,* 123-135. doi:10.1007/978-3-319-24858-5_8

Alonso, A. A. (n.d.). Intrusion Detection Through Relationship Analysis. Retrieved August 22, 2016, from https://www.sans.org/reading-room/whitepapers/detection/intrusion-detection-relationship-analysis-37352

Accessed from the SANS Reading Room

Bejtlich, R. (2013). *The Practice of Network Security Monitoring: Understanding Incident Detection and Response*. San Francisco: No Starch Press.

Bosco, P. (2016, January 20). Intrusion Detection and Prevention Systems Cheat Sheet: Choosing the Best Solution, Common Misconfigurations, Evasion Techniques, and Recommendations. Retrieved February 22, 2017, from https://www.sans.org/reading-room/whitepapers/detection/intrusion-detection-prevention-systems-cheat-sheet-choosing-solution-common-misconfigurations-evasion-techniques-recommendations-36677

Accessed from the SANS Reading Room

Bottomley, J., & Emelyanov, P. (2014). Operating Containers. USENIX, 39(5). Retrieved December 11, 2017, from https://www.usenix.org/system/files/login/articles/login_1410_02-bottomley.pdf

Goyal, P. (2017, July 6). CIS Docker Community Edition Benchmark [PDF]. East Greenbush: Center for Internet Security.

Alfredo Hickman, ahusmc@yahoo.com

Davidoff, S., & Ham, J. (2012). Network Forensics Tracking Hackers Through Cyberspace. Upper Saddle River: Prentice Hall.

Gao, X., Gu, Z., Kayaalp, M., Pendarakis, D., & Wang, H. (2017, June). ContainerLeaks: Emerging Security Threats of Information Leakages in Container Clouds [PDF]. Williamsburg: College of William and Mary.

Presented at the 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks

Hayden, M. (2015, July 26). Securing Linux Containers. Retrieved February 23, 2017, from https://www.sans.org/reading-room/whitepapers/linux/securing-linux-containers-36142 Accessed from the SANS Reading Room

Hosburgh, M. (n.d.). Offensive Intrusion Analysis: Uncovering Insiders with Threat Hunting and Active Defense. Retrieved July 6, 2017, from https://www.sans.org/reading-room/whitepapers/detection/offensive-intrusion-analysis-uncovering-insiders-threat-hunting-active-defense-37885

Accessed from the SANS Reading Room

HOW TO SECURELY CONFIGURE A LINUX HOST TO RUN CONTAINERS [PDF]. (2017). San Francisco: Twistlock.

Accessed from https://www.twistlock.com/resources/securely-configure-linux-host-run-containers/

Hykes, S. (Writer). (2017, November 20). The future of Linux Containers. Live performance in Pycon U.S. 2013: Santa Clara Convention Center, Santa Clara.

Kennedy, C. (2016, June 29). Deception Techniques as Part of Intrusion Detection Strategy. Retrieved February 22, 2017, from https://www.sans.org/reading-room/whitepapers/detection/deception-techniques-intrusion-detection-strategy-37140

Accessed from the SANS Reading Room

Lockheed Martin Corporation. (n.d.). *The Cyber Kill Chain* [Brochure]. Author. Retrieved October 23, 2018, from https://www.lockheedmartin.com/us/what-we-do/aerospace-defense/cyber/cyber-kill-chain.html

Mohallel, A. A., Bass, J. M., & Dehghantaha, A. (2016). Experimenting with docker: Linux container and base OS attack surfaces. 2016 International Conference on Information Society (i-Society). doi:10.1109/i-society.2016.7854163

Petazonni, J. (2015, August 15). Anatomy of a Container: Namespaces, cgroups & Some Filesystem Magic. Retrieved July 30, 2017, from https://www.slideshare.net/jpetazzo/anatomy-of-a-container-namespaces-cgroups-some-filesystem-magic-linuxcon

Presentation at LinuxCon 2015

Robinson, A. (2016, November 18). A Checklist for Audit of Docker Containers. Retrieved February 23, 2017, from https://www.sans.org/reading-room/whitepapers/auditing/checklist-audit-docker-containers-37437

Accessed from the SANS Reading Room

Souppaya, M., Morello, J., & Scarfone, K. (n.d.). Draft (2nd) NIST Special Publication 800-190 Application Container Security Guide (USA, NIST). Retrieved July 13, 2017, from http://csrc.nist.gov/publications/drafts/800-190/sp800-190-draft2.pdf

Winkel, S. (2016, November 18). Security Assurance of Docker Containers. Retrieved February 23, 2017, from https://www.sans.org/reading-room/whitepapers/assurance/security-assurance-docker-containers-37432

Accessed from the SANS Reading Room

Winkel, S. (2017, July 9). Forensicating Docker with ELK. Retrieved July 30, 2017, from https://www.sans.org/reading-room/whitepapers/forensics/forensicating-docker-elk-37870

Accessed from the SANS Reading Room

Alfredo Hickman, ahusmc@yahoo.com

Appendix

Attack Phase	Test Cases	Test Case Commands
Scanning Detection	Sparta scan with nmap	sparta
Scanning Detection	Nikto web app scan	nikto -h http://192.168.1.19/*.*
Scanning Detection	NMAP host scan intense plus UDP	nmap -sS -sU -T4 -A -v 192.168.1.24
Scanning Detection	NMAP host scan stealth (SYN scan)	nmap -sS host_ip
Scanning Detection	Internal network scan intense	nmap -T4 -A -v 192.168.1.0/24
Scanning Detection	Host vulnerability scan Nessus basic	
Scanning Detection	Host vulnerability scan Nessus WebApp Scan	
App Attack Detection	Conduct SQL injection attack	1' OR 1=1 UNION SELECT null, version()#
App Attack Detection	Conduct authentication and session management attack	Use BrutePWPwnage.txt in Kali-MS-DVWA DB Folder
App Attack Detection	Conduct XSS attack reflected	<script>alert(123)</script>
Malware Detection	Deploy malicious payload to host	wget http://www.eicar.org/download/eicar.com and
Malware Detection	Execute malicious payload on host	sh eicar.sh
C2 Detection	Execute C2 activity on host	SSH and Netcat
Privilege Escalation Detection	Execute privilege escalation on host	SU with brute-forced creds
Data Exfiltration Detection	Conduct data exfiltration	nc -l -p 7777 > filename & nc 192.168.1.22 7777 < filename
File Integrity Detection	Alter sensitive files and check FIM for alerts (registry, conf files, password files, system files, user	add test to the end of passwd and shadow
System Information Leakage	Check for detection of leaked system data (resource usage, location services)	nmap -sV --script http-apache-server-status 192.168.1.17
Auto Anomaly Detection	Check for automated alerting of suspected suspicious behavior - execute potentially malicious	
Attacker - Victim Relation	Check for relationship mapping between attacker and victim	
Forensic Artifact Retrieval	Check for capabilities to retrieve forensic artifacts (logs, pcaps, flows, files)	