Article

Analyzing Docker Vulnerabilities through Static and Dynamic Methods and Enhancing IoT Security with AWS IoT Core, CloudWatch, and GuardDuty

Vishnu Ajith, Tom Cyriac *, Chetan Chavda *, Anum Tanveer Kiyani, Vijay Chennareddy and Kamran Ali

Department of Computer Science, Middlesex University, London NW4 4BT, UK; vishofficial27@gmail.com (V.A.); a.kiyani@mdx.ac.uk (A.T.K.); chennareddyvijay992@gmail.com (V.C.); k.ali@mdx.ac.uk (K.A.)
* Correspondence: tomcyriac031@gmail.com (T.C.); cchetan13@gmail.com (C.C.)

Abstract: In the age of fast digital transformation, Docker containers have become one of the central technologies for flexible and scalable application deployment. However, this has opened a new dimension of challenges in security, which are skyrocketing with increased technology adoption. This paper discerns these challenges through a manifold approach: first, comprehensive static analysis by Trivy, and second, real-time dynamic analysis by Falco in order to uncover vulnerabilities in Docker environments pre-deployment and during runtime. One can also find similar challenges in security within the Internet of Things (IoT) sector, due to the huge number of devices connected to WiFi networks, from simple data breaches such as brute force attacks and unauthorized access to large-scale cyber attacks against critical infrastructure, which represent only a portion of the problems. In connection with this, this paper is calling for the execution of robust AWS cloud security solutions: IoT Core, CloudWatch, and GuardDuty. IoT Core provides a secure channel of communication for IoT devices, and CloudWatch offers detailed monitoring and logging. Additional security is provided by GuardDuty's automatized threat detection system, which continuously seeks out potential threats across network traffic. Armed with these technologies, we try to build a more resilient and privacy-oriented IoT while ensuring the security of our digital existence. The result is, therefore, an all-inclusive work on security in both Docker and IoT domains, which might be considered one of the most important efforts so far to strengthen the digital infrastructure against fast-evolving cyber threats, combining state-of-the-art methods of static and dynamic analyses for Docker security with advanced, cloud-based protection for IoT devices.

Keywords: Docker; container; mobile cloud; Ad hoc cloud; IoT; Falco; Trivy; GuardDuty; IoT Core; security; AWS cloud

Citation: Ajith, V.; Cyriac, T.; Chavda, C.; Kiyani, A.T.; Chennareddy, V.; Ali, K. Analyzing Docker Vulnerabilities through Static and Dynamic Methods and Enhancing IoT Security with AWS IoT Core, CloudWatch, and GuardDuty. *IoT* **2024**, *5*, 592–607. https://doi.org/10.3390/iot5030026

Academic Editor: Amiya Nayak

Received: 14 June 2024
Revised: 13 August 2024
Accepted: 20 August 2024
Published: 4 September 2024

1. Introduction

The rapid advancement of digital technologies, including Docker and the Internet of Things (IoT), has greatly boosted IT infrastructure by enhancing operational efficiency. Yet, this progress has also brought forth crucial security weaknesses that require prompt and thorough mitigationplans [1]. This study attempts to investigate vulnerabilities in Docker by utilizing static and dynamic analysis tools like Trivy and Falco. It also seeks to improve IoT security by leveraging AWS cloud security services such as IoT Core, CloudWatch, and GuardDuty. This research aims to enhance the security of Docker and IoT platforms by creating a strong security architecture, making a substantial contribution to the cybersecurity sector. It highlights a proactive strategy for cybersecurity, adjusting to the ever-changing landscape of digital risks. This paper positions itself within the broader academic discourse on Docker and IoT security vulnerabilities, analysis methods, and cloud security services by referencing the current literature. This study focuses on evaluating the usefulness of analysis tools in detecting Docker vulnerabilities and the impact of AWS services on IoT security through qualitative analysis. This introduction prepares the reader for an in-depth examination of cybersecurity measures within the realm of advanced digital technologies.

Information technology (IT) has gone through a tremendous evolution over the past few decades. From the early days of mainframe computers to the current era of cloud computing and artificial intelligence, IT has continuously transformed how businesses operate and how individuals interact with one another [2]. This evolution has led to increased efficiency, productivity, and connectivity in a wide array of areas, but it has also introduced new challenges, particularly in the realm of cybersecurity.

Docker, a platform for developing, shipping, and running containers with the ability to package applications in containers, has revolutionized software deployment and scaling. Containers allow applications to run in different computing environments in a consistent manner [3]. From the developer's local machine to production servers, this technology has made application deployment and management much easier but has, at the same time, introduced new security considerations that need to be addressed.

The Internet of Things (IoT) refers to the network of interconnected physical devices, with the electronics built into software, sensors, and network connectivity enabling these things to collect and share data [4]. IoT solutions are found in a wide variety of areas, including smart homes, industrial automation, healthcare, agriculture, and urban planning. They serve the purposes of enhancing efficiency, improving decision making through data analysis, and creating new services and experiences for users. However, the vast number of connected devices also poses huge security challenges.

To address these challenges, this study explores the use of the following AWS cloud security services:

- AWS IoT Core: Managed cloud service that allows devices to interoperate securely and safely with cloud applications and other devices [5].
- CloudWatch: A monitoring and observability service that gives AWS, on-premises, other cloud applications, and infrastructure both data and actionable insights and resources [6].
- GuardDuty: A threat detection service that continuously keeps an eye on AWS accounts and workloads to help identify malicious activity and deliver detailed security findings for visibility and remediation [7].

The contributions of this paper are as follows: (i) In particular, we underline our combined usage of static and dynamic techniques in Docker vulnerability analysis as well as the new integration of AWS IoT Core, CloudWatch, and Guard Duty to provide reinforcement for IoT security. (ii) In this research article, we also highlight IoT security concerns related to cyber attacks and brute force attacks on IoT devices. We discuss the security concerns on the platforms AWS IoT Core, CloudWatch, and Guard Duty. (iii) We performed simulation and testing on AWS Core and present the results.

The remainder of this paper is structured as follows: Section II discusses related works. Section III reviews the security mechanisms together with managing vulnerabilities and threats in docker deployment. Section IV discusses the IoT services used in deployment and building a simulation. Section V reviews the outcomes of a brute force attack and results. Section VI discusses future prospects of this study. Section VII concludes the research paper.

2. Related Works

The basis for this review is the recognition that traditional security standards and evaluation frameworks that are designed for non-IoT settings are vastly different and many may not directly address the security requirements of IoT-based smart environments. As a result, this paper explores the potential of conventional security standards and assessment frameworks to tackle some of the security concerns in IoT-based smart environments by highlighting some core areas of focus as well as the background of the field.

Although the benefits and possibilities of an expanded IoT-based smart environment are significant and still growing rapidly, the attack surface is also broad. Thus, with the increasing number of IoT devices, the greater ecosystem, and increased integration, many vulnerable endpoints are being identified regularly, particularly in smart homes, smart cities, global businesses, and critical infrastructures. While IoT-based smart environments are an ever-growing trend, expansion comes with a lot of complexity, integration, and security challenges across various application domains. Given these factors, a review of existing conventional security standards and assessment frameworks is vital to identifying the key and ongoing security concerns in IoT-based smart environments.

Hence, the pivotal findings and conclusions of this study are expressly derived from harnessing the principles embedded within established conventional security standards and assessment frameworks, recognized for their applicability within IoT-driven smart environments. Additionally, the study delineates and deliberates on unresolved issues and hurdles, concurrently suggesting a taxonomy of challenges specific to IoT-based smart environments. This taxonomy maps the identified challenges to prospective solutions aimed at mitigating current and anticipated security concerns in the realm of IoT.

The advent of the Internet of Things (IoT) has sparked a transformative shift in the realm of networks, owing to its vast array of intelligent applications. These applications encompass various domains, such as healthcare, energy grids, banking, and a myriad of other services, collectively shaping a smarter future. Like many other fields, the IoT domain is growing very fast. However, with this growth come many cybersecurity challenges [8].

2.1. Security Challenges

The Internet of Things has revolutionized the way we live and interact with technology [9]. However, with this increased connectivity and integration of devices, there are numerous security concerns that need to be addressed. These concerns stem from the fact that IoT devices collect and transmit a vast number of data, making them attractive targets for hackers and cyber criminals. Some of the major security concerns in the evolving landscape of the Internet of Things include the following:

- Unauthorized access: With the growing number of connected devices in the IoT ecosystem, unauthorized access becomes a major concern. Hackers may exploit vulnerabilities in IoT devices to gain unauthorized access and control over them, potentially leading to various malicious activities, such as data theft, privacy breaches [8], or even physical harm [10].
- Data breaches: IoT devices collect and transmit a significant number of sensitive data, including personal and financial information. Therefore, data breaches pose a significant risk in the IoT landscape. Hackers may intercept or manipulate these data, leading to identity theft, financial fraud, or other serious consequences.
- Lack of encryption: IoT devices often lack robust encryption mechanisms to protect the data they transmit. This makes them vulnerable to interception and unauthorized access.
- Inadequate firmware and software security: Many IoT devices have vulnerabilities in their firmware or software that can be exploited by attackers. These vulnerabilities can allow hackers to gain unauthorized access to the device, manipulate its functionality, or use it as a gateway to attack other devices on the network.
- Lack of standard security protocols: The IoT ecosystem lacks uniform security protocols and standards, making it challenging to ensure consistent security across different devices and networks [4,10].
- Insufficient authentication and access controls: Weak authentication mechanisms and inadequate access controls make it easier for attackers to impersonate authorized users or gain unauthorized access to IoT devices and networks [4].
- Insufficient device management and updates: Many IoT devices lack proper management systems and mechanisms for regular software updates. This leaves them vulnerable to new security threats that may arise over time, as there are no mechanisms in place to patch vulnerabilities or fix software bugs [9]. Addressing these security concerns is crucial to the successful and safe implementation of the Internet of Things [4].

2.1.1. Hacking Techniques

As we look at the benefits of what IoT devices bring to daily life, be it personal or in the enterprise environment, it is important to note that these devices come with their own vulnerabilities. Malicious actors or hackers can employ several techniques in order to take advantage of weaknesses in IoT devices. Malicious actors most often employ the following methods:

- Remote code execution: Viral code can be remotely executed by attackers by taking advantage of flaws in IoT devices' firmware or software. Once in control of the device, they can alter its operations, pilfer data, or utilize it as a component of a botnet to launch coordinated assaults.
- Man-in-the-middle attacks: In such attacks, the communication between IoT devices and their intended recipients is intercepted and altered by the attacker. They may modify the access to private information being transmitted, introduce malicious commands or code, or gain unauthorized access. Based on the research carried out by [1] on the Rain Monitor app, "it uses OpenXC to collect sensitive data (such as location, wind-shield status, and speed) of a car, and transmit it to a remote Web service, where it is collected and used to inform drivers of possible showers in their area". It is very much evident that the app is subject to leakage of sensitive data to the Internet, which can be considered a great privacy issue.

2.1.2. Challenges in Stopping IoT Attacks

In recent years, the Internet of Things has seen exponential growth and adoption in various industries and sectors. However, along with this growth comes an increase in security challenges and vulnerabilities. As highlighted in a study, the interconnected nature of IoT devices creates new attack surfaces and potential entry points for malicious actors [11]. These attacks can lead to serious consequences, including data breaches, privacy violations, and disruptions in critical infrastructure. One of the challenges in stopping IoT attacks is the lack of comprehensive research on smart contract vulnerabilities and their incorporation in IoT systems. Another challenge identified in the literature is the issue of maintaining privacy and ensuring data security. Studies by [12], as well as [13], emphasize the inadequate research on privacy threats in the IoT and the need for legislation to address these concerns [14]. Additionally, the complexity of IoT systems and the sheer number of devices connected can make it difficult to detect and respond to attacks in a timely manner. Furthermore, the lack of standardization in data sharing and collection mechanisms performed by IoT devices poses a challenge to addressing security vulnerabilities. Furthermore, the authors also state that informed consent, privacy, information security, physical safety, and trust have been explored in the literature and are valuable foundations that underpin ethical research practices in the realm of IoT security challenges.

2.2. Initiatives and Solutions

The growth of the Internet of Things brings forward numerous security challenges, which have prompted several initiatives and solutions, some of which are discussed in the following sections.

2.2.1. Intrusion Detection Systems

Machine learning and deep learning-based intrusion detection systems are being developed to spot unusual network traffic patterns and potential threats in IoT networks. The idea is to use intelligent algorithms that can learn from a vast number of data and identify anomalies that may indicate a cybersecurity threat [15].

2.2.2. Secure Boot

Secure Boot is a fundamental security standard that IoT devices can utilize to prevent the execution of unauthorized software during the device's booting process, which helps in protecting against a wide range of attacks [11].

2.2.3. Encryption

Encryption aims to ensure that the data transmitted and stored by IoT devices are encrypted to protect them from being intercepted and read by unauthorized entities [11].

2.2.4. Device Authentication

Implementing strong, multi-factor authentication mechanisms for devices to validate their identity can prevent unauthorized devices from gaining access [11].

2.2.5. Regular Updates and Patches

Keeping IoT devices and software updated with the latest security patches can address known vulnerabilities [14].

2.2.6. Education and Awareness

Increasing awareness among both users and developers regarding the importance of security in the IoT context is vital to the overall safety of these devices [14].

2.2.7. Standardization

Standardization refers to the development of global security standards for IoT devices to set a clear security benchmark for manufacturers to meet [15]. Utilizing a multifaceted strategy incorporating various technological measures, regulatory frameworks, standardization efforts, and educational initiatives stands as imperative in cultivating a secure Internet of Things (IoT) ecosystem and grappling with the intricate security quandaries it encounters. The proliferation of threats in this domain is incessant, with ongoing attacks reaching unprecedented levels, potentially numbering in the billions. Concurrently, market dynamics exert significant influence over the development of IoT products, introducing inherent risks. The paramount concern herein lies not only in safeguarding users' privacy but also in thwarting criminal exploitation of information that could pose threats to safety. The overarching message conveyed by this study resonates not only with technology professionals but also with end-users, underlining the collective responsibility in fortifying IoT security.

Docker's incorporation into software deployment has transformed the DevOps sector by improving the efficiency of application distribution in various situations. Nevertheless, this change has brought forth notable security issues, such as container breakout hazards, image vulnerabilities, and network mis-configurations. Tools like Trivy and Falco have played a crucial role in performing static and dynamic evaluations, providing a means to address these vulnerabilities prior to deployment.

Docker utilizes many security measures, including namespaces for isolating processes, control groups for limiting resources, Docker Secrets for managing sensitive information, and Docker Bench for Security for evaluating compliance. Although efforts have been made to enhance Docker's security, there are still vulnerabilities due to its fast-paced development and integration of new functionalities.

Comparative assessments show that Docker has a distinct security position compared with traditional VMs and other orchestration solutions, such as Kubernetes. This highlights Docker's efficient security strategy as well as its possible weaknesses. Further studies are needed to investigate the long-term effectiveness of vulnerability scanning tools and to create improved intrusion detection systems specifically designed for containerized settings, as indicated in the literature.

This analysis emphasizes the importance of ongoing innovation in Docker security measures to tackle the intricate, changing environment of cyber threats. It recommends a comprehensive strategy involving technology, policy, and education to improve the security of containerized applications [16].

3. Comprehensive Security Mechanisms in Docker Deployment: Managing Vulnerabilities and Threats

3.1. Preliminaries and Background

Docker is an open-source platform that enables developers to create, deploy, and oversee programs in lightweight, adaptable containers. These containers package an application along with all its dependencies to maintain uniformity across various computing environments and streamline development, testing, and deployment procedures. Docker's containerization technology is a fundamental aspect of modern DevOps processes, providing remarkable flexibility and mobility. Nevertheless, the implementation of Docker containers brings about security obstacles. Failure to control vulnerabilities in Docker can undermine its isolation capabilities. Docker Inc. has improved its platform's security by implementing robust measures, including Docker Secrets for securely handling sensitive information and Docker Bench for Security, which performs automated compliance checks against recognized security standards. Implementing these strategic enhancements is crucial to integrating security across the container lifecycle, protecting containerized applications from advancing cyber threats. It is essential to make strategic enhancements to incorporate security throughout the whole lifecycle of containers to safeguard containerized applications against advancing cyber threats, as detailed in Docker's official documentation [8,10].

Docker containers, known for their efficiency and adaptability, persist in encountering unique security issues that are not fully resolved by traditional security approaches. The advancement of Docker technology has not alleviated these issues, which primarily stem from its inherent design features. Recent studies and developments have further highlighted these security concerns as follows:

- Shared kernel architecture: Docker containers leverage the host system's kernel, distinguishing them from virtual machines. This shared kernel architecture economizes resources but introduces substantial security vulnerabilities, especially in multi-tenant systems where containers from various users share the same host kernel [4].
- Security challenges of short-lived containers and image use: Containers' temporary and short-lived nature, often created from images, presents unique security challenges. These container images can harbor vulnerabilities, and their ephemeral existence complicates the application of traditional, long-term security measures [9].
- Isolation and resource sharing: While containers provide self-contained environments, they share common resources like network and storage with other containers on the same host. Inadequately secured resource sharing can lead to unauthorized data access [2].
- Container sprawl: The ease of deploying Docker containers can result in container sprawl, where an unmanaged proliferation of containers leads to operational complexity and security oversight. Proper governance and lifecycle management are necessary to mitigate the risks associated with outdated or unnecessary containers [1].

3.2. Security Architecture and Vulnerability Assessment in Docker Environments

3.2.1. Static and Dynamic Security Assessment Strategy

Our study outlines a two-pronged approach to thoroughly evaluate the security of Docker containers in a cloud setting, utilizing both static and dynamic analysis techniques. This concept is implemented in a practical environment by using Docker containers managed on Amazon Web Services (AWS) EC2 instances, replicating real-world cloud computing situations.

3.2.2. Static Analysis Using Trivy

Trivy is a crucial open-source vulnerability scanner specifically created for container images to enhance container security in static analysis frameworks. The scanning capabilities are extensive and cover Docker images, file systems, Git repositories, Infrastructure as Code (IaC) files such as Terraform, Kubernetes manifests, and AWS CloudFormation templates, along with application dependencies in several programming languages. Trivy's multidimensional methodology allows it to efficiently find vulnerabilities prior to deployment, effortlessly integrating into CI/CD pipelines for early detection. Trivy ensures accurate identification of security risks in fast-changing digital environments by keeping an updated vulnerability database, which helps minimize false positives. Trivy's wide scanning spectrum supports its role in improving the security of container images and related infrastructure, ensuring strong protection against various cyber attacks. The Figure 1 shows the process flow which clearly demonstrates the idea of doing the static analysis within the environment.

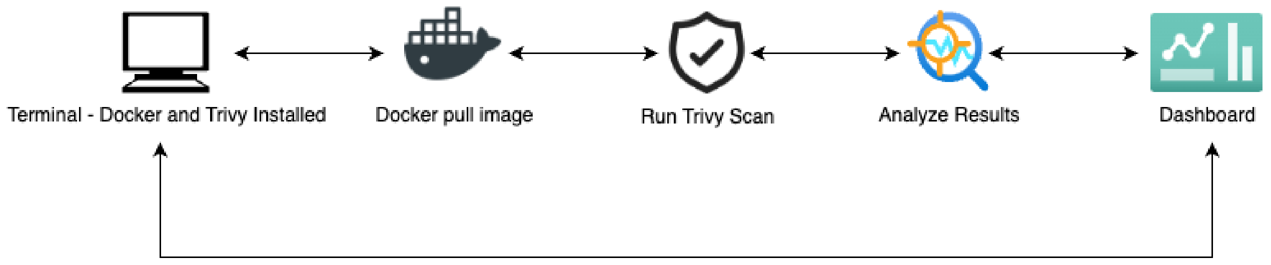


Figure 1. Process flow—Trivy.

We extensively depend on Trivy in our research to uncover vulnerabilities of different severity levels in Docker images. Early detection facilitated by Trivy is crucial for prompt resolution and improving the security protocols of container images before they are used in production. We have enhanced the functionality of Trivy by creating a custom Python script to interpret, assess, and graphically display the output data from Trivy. This script simplifies the process of generating an AWS-hosted dashboard that provides a user-friendly and thorough overview of the security status of the Docker images being assessed. Figure 2 showcases vulnerabilities, categorizes them based on severity, and offers remedy insights, greatly assisting in the proactive handling of container security.

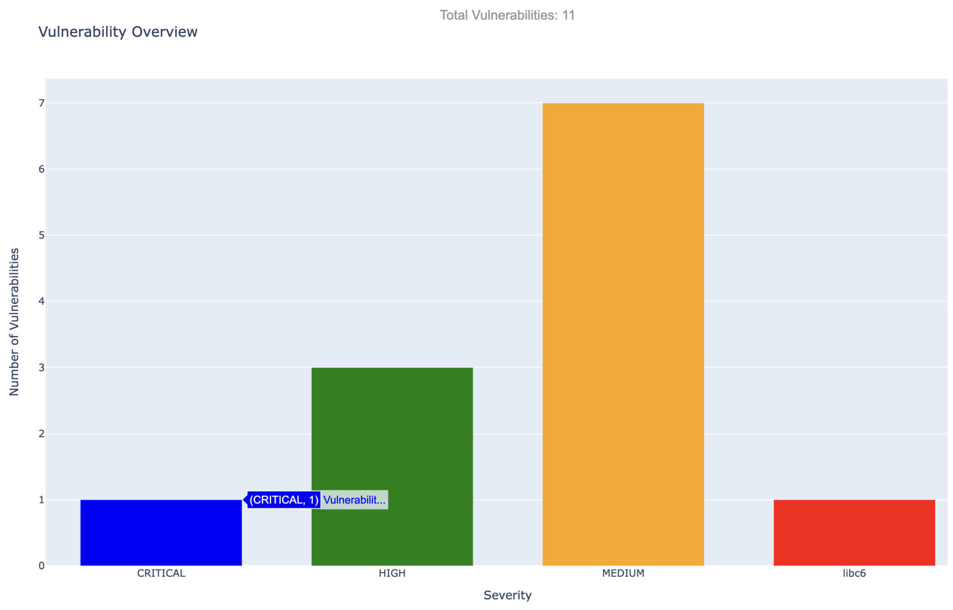


Figure 2. The graphical representation of vulnerabilities detected in the Docker container image.

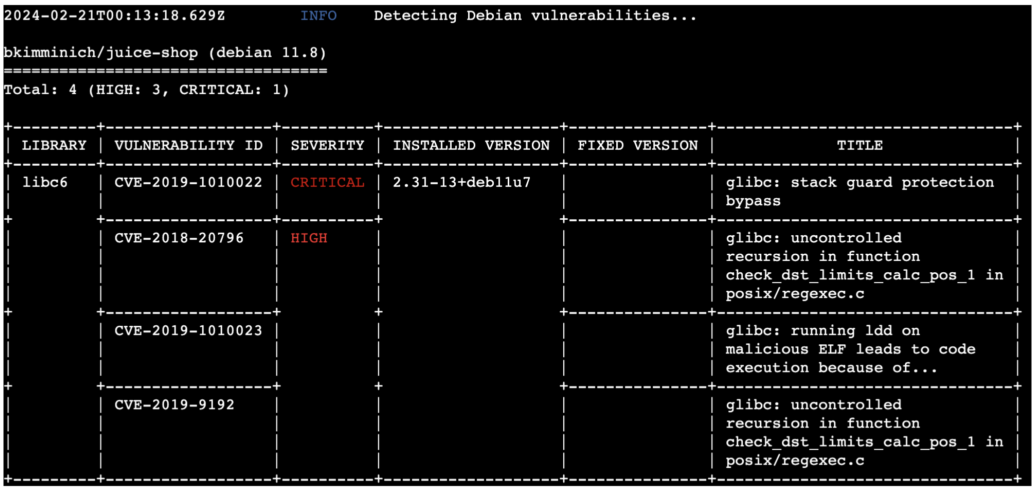
Static vulnerability analysis is a pre-deployment process that involves scanning Docker container images and configurations for known vulnerabilities. Tools like Trivy check for issues in software packages, dependencies, and configurations, allowing developers to address security flaws early in the development cycle, reducing risks before containers are deployed.

Implementation Steps

- Environment Setup: Make sure that Docker and Trivy are installed on the system.
- Pull the Docker image: Pull the Docker image for analysis by using the appropriate Docker commands.
- Run Trivy scan: Run Trivy scan on the Docker image with a command like "trivy image <image-name>". This command will generate a detailed list of vulnerabilities found in the image.
- Results analysis: Review the results and classify the vulnerabilities according to their criticality, such as critical, high, or medium. Address the identified vulnerabilities by updating dependencies, fixing them, or changing configurations.

Results and Metrics for Trivy

The Figure 3 shows the static examination conducted using Trivy disclosed eleven distinct vulnerabilities in the scrutinized Docker container image. The vulnerabilities were categorized based on their severity: seven of medium concern, three of high concern, and one deemed critical. The most serious issue discovered was linked to `libc6` (CVE-2019-1010022), which involves a circumvention of stack guard protection. The Figure 3, generated from Trivy's JSON results by using a custom Python script, provides an intuitive and immediate visual interpretation of the vulnerabilities, emphasizing their severity and enabling an efficient prioritization for remediation efforts.



```
2024-02-21T00:13:18.629Z        INFO    Detecting Debian vulnerabilities...

bkimminich/juice-shop (debian 11.8)
===================================
Total: 4 (HIGH: 3, CRITICAL: 1)

+---------+------------------+----------+-------------------+---------------+--------------------------------+
| LIBRARY | VULNERABILITY ID | SEVERITY | INSTALLED VERSION | FIXED VERSION |             TITLE              |
+---------+------------------+----------+-------------------+---------------+--------------------------------+
| libc6   | CVE-2019-1010022 | CRITICAL | 2.31-13+deb11u7   |               | glibc: stack guard protection  |
|         |                  |          |                   |               | bypass                         |
+         +------------------+----------+                   +---------------+--------------------------------+
|         | CVE-2018-20796   | HIGH     |                   |               | glibc: uncontrolled            |
|         |                  |          |                   |               | recursion in function          |
|         |                  |          |                   |               | check_dst_limits_calc_pos_1 in |
|         |                  |          |                   |               | posix/regexec.c                |
+         +------------------+          +                   +---------------+--------------------------------+
|         | CVE-2019-1010023 |          |                   |               | glibc: running ldd on          |
|         |                  |          |                   |               | malicious ELF leads to code    |
|         |                  |          |                   |               | execution because of...        |
+         +------------------+          +                   +---------------+--------------------------------+
|         | CVE-2019-9192    |          |                   |               | glibc: uncontrolled            |
|         |                  |          |                   |               | recursion in function          |
|         |                  |          |                   |               | check_dst_limits_calc_pos_1 in |
|         |                  |          |                   |               | posix/regexec.c                |
+---------+------------------+----------+-------------------+---------------+--------------------------------+
```

Figure 3. Trivy scan command-line output showing detailed vulnerability information.

These vulnerabilities are spread throughout critical system libraries and signal the necessity for urgent patching or updates to secure iterations. The dashboard rendered through our bespoke Python script offers a user-friendly interface that maps out the identified vulnerabilities according to their severity, assisting in the stratification of response measures. Vulnerabilities classified as critical and high were marked for swift resolution, with Trivy providing remediation recommendations for the compromised packages.

This analysis is crucial to reinforcing the defense mechanisms of Docker containers prior to their deployment in production settings, ensuring that we proactively address the most significant threats as detected by Trivy's extensive scanning functionality.

3.2.3. Dynamic Analysis with Falco

Falco is a state-of-the-art security solution designed for cloud environments. It excels at detecting and alerting users about abnormal activities that may signal security threats in real time. Falco's dynamic analysis enhances our security protocols through continuous real-time surveillance of Docker containers. The primary strength of the system is its advanced behavioral analysis, which surpasses static analysis by monitoring and interpreting real-time container activity to detect threats through abnormal behavior patterns. Falco's rules can be tailored extensively to precisely define abnormal behavior within a particular operational setting. Its smooth interface with orchestration systems such as Kubernetes is essential to ensuring security in intricate DevOps workflows. Falco offers many alerting options, such as emails and connection with third-party communication systems like Slack, to immediately notify teams of potential security risks [17]. Falco's skills are essential to comprehending and reducing the ever-changing security threats encountered by Docker containers, guaranteeing ongoing operational reliability [14].

The process flow of the implementation is briefly explained in Figure 4. Dynamic vulnerability analysis complements static analysis by monitoring the runtime behavior of Docker containers. Falco and other tools monitor system calls and activities in real time to detect anomalies, which include unauthorized access or unexpected network connections that would identify and mitigate threats that might be overlooked when using only static analysis.

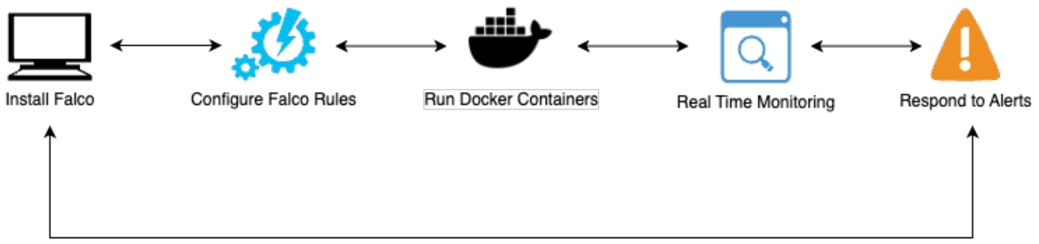


Figure 4. Process flow—Falco.

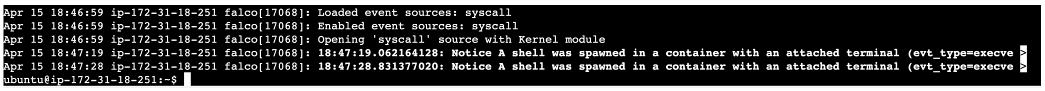
Implementation Steps

- Install Falco: Install Falco and make sure it is set up to watch the Docker containers of interest for analysis.
- Set up monitoring: Configure Falco rules such that the rules define the kind of activity that you consider malicious for the specific use cases of your application. For example, you could define rules to detect any unauthorized shell access or rules to detect unexpected network activity.
- Run the containers: Start the Docker containers. Falco will automatically start monitoring their activity in real time.
- Respond to alarms: When there is an event of abnormal activity, alerts are generated by Falco. This is the best way to react by looking into these alert messages ASAP, realize whether there is a real threat in them, and take necessary steps to reduce such risk.

Results and Metrics for Falco

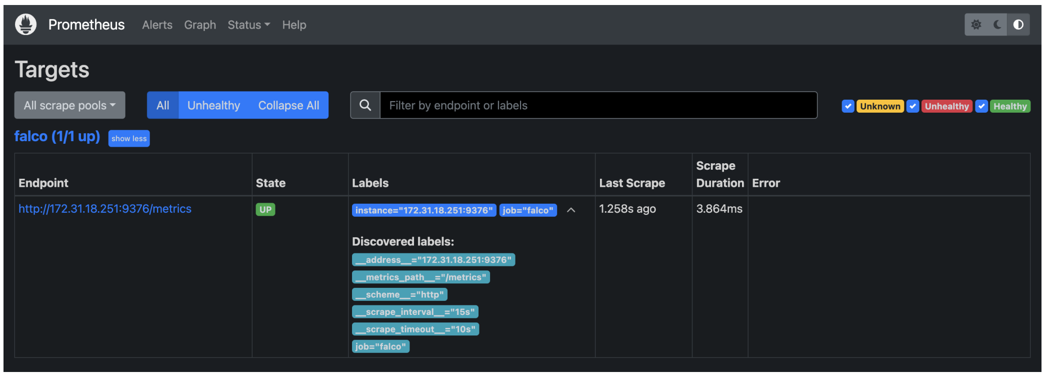
This implementation strategy serves as the framework for our security assessment, enabling us to comprehensively evaluate the security of Docker containers in a cloud-based environment. By combining static and dynamic analysis tools, we aim to enhance the security measures for Docker containers, addressing both known vulnerabilities and real-time threats effectively. Falco is used in our security architecture because of its skill of monitoring in real time and its ability to quickly identify abnormalities like illegal shell calls inside Docker containers. We may view a real-time recording of security issues, such as unusual file system modifications, network intrusions, and suspicious process activities, by using the "journalctl -fu falco" command within the terminal of an Ubuntu server.

The Figure 5 shows the output of anomalies detected by Falco. We use PromQL to analyze these data by integrating Falco with Prometheus, which allows us to accurately identify and measure unusual activities. Security Administrators can quickly respond to and look into any risks by utilizing Prometheus's alerting features, which we can be set up to create rules that send out notifications for particular Falco-detected patterns. The output is showcased in Figure 6.



```
Apr 15 18:46:59 ip-172-31-18-251 falco[17068]: Loaded event sources: syscall
Apr 15 18:46:59 ip-172-31-18-251 falco[17068]: Enabled event sources: syscall
Apr 15 18:46:59 ip-172-31-18-251 falco[17068]: Opening 'syscall' source with Kernel module
Apr 15 18:47:19 ip-172-31-18-251 falco[17068]: 18:47:19.062164128: Notice A shell was spawned in a container with an attached terminal (evt_type=execve >
Apr 15 18:47:28 ip-172-31-18-251 falco[17068]: 18:47:28.831377020: Notice A shell was spawned in a container with an attached terminal (evt_type=execve >
ubuntu@ip-172-31-18-251:~$
```

Figure 5. Terminal output showcasing anomalies detected in real time by Falco.



Targets

falco (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://172.31.18.251:9376/metrics	UP	instance="172.31.18.251:9376" job="falco" Discovered labels: __address__="172.31.18.251:9376" __metrics_path__="/metrics" __scheme__="http" __scrape_interval__="15s" __scrape_timeout__="10s" job="falco"	1.258s ago	3.864ms	

Figure 6. Metrics loaded by Falco to Prometheus.

Through this integration, a dynamic and responsive security apparatus is built, which is essential to preserving the integrity of containerized systems and quickly resolving a variety of security risks [18].

3.3. Future Directions

In the future, container security will require continuous adaptation and innovation. Docker's official documentation and the wider security community highlight many crucial areas that are expected to be essential for future study as follows:

- Improved network security for containers by investigating advanced network segmentation and encryption methods to enhance isolation and safeguard container traffic.
- Utilizing AI and ML algorithms to enhance the identification of complex risks and anomalies in container behavior.
- Incorporating security measures into the CI/CD pipeline to help automate security checks and maintain security as a constant priority during the application lifecycle.
- Exploring methods for implementing immutable containers to mitigate runtime vulnerabilities by replacing them instead of modifying them [8].

The instructions demonstrate the changing nature of container security and the ongoing requirement for creative methods to combat emerging threats. Future research can enhance the development of stronger and more durable security measures for Docker containers and the wider range of containerized applications by concentrating on these specific areas.

4. Implementation Environment—IoT

For the implementation part, AWS IoT Core (secure communication), Lambda (code execution), S3 (data storing), CloudWatch (metric monitoring), and GuardDuty (threat monitoring) AWS services are used.

4.1. AWS IoT Core

AWS IoT Core, offered by Amazon Web Services (AWS), is a managed cloud platform designed to facilitate secure and reliable communication between connected devices and cloud applications. It serves as a central hub for IoT solutions, providing features like diverse communication protocol support, robust security measures for authentication and data encryption, and device management functionalities including onboarding and registration at scale. With its Rules Engine, AWS IoT Core enables real-time data processing and routing based on predefined conditions, seamlessly integrating with other AWS, like storage, databases, and analytics. It ensures scalability, high availability, and reliability, making it easier to develop, deploy, and manage IoT applications while handling large fleets of devices efficiently [5,19].

4.2. AWS Lambda

A computing service called Lambda enables to run code without a server. Background processing resources are automatically managed by Lambda. Only the computing time used is charged to the customer, not the running code. An Amazon Web Services lambda function is a stateless piece of code [20].

4.3. AWS GuardDuty

GuardDuty is a security service in AWS that helps to protect the environment by continuously monitoring for malicious activity and unauthorized behavior. It analyses logs and network traffic to detect threats such as compromised accounts, malware, and unauthorized access attempts [7].

4.4. Amazon S3

Productivity, security, scalability, and data availability are among the attributes of Amazon Simple Storage Service (Amazon S3), an object storage solution. A vast array of use cases, including data lakes, websites, mobile applications, backup and restore, archiving, business applications, IoT devices, and big data analytics, can be served by using Amazon S3 to store and safeguard any volume of data for clients of all sizes and sectors [21].

4.5. AWS CloudWatch

The real-time monitoring of Amazon Web Services (AWS) resources and applications is provided via Amazon CloudWatch. CloudWatch is a tool for gathering and monitoring metrics, or variables, related to the applications and resources. When a threshold is crossed, we can set up alarms to automatically adjust the resources we are monitoring or observe metrics and send messages. CloudWatch can be used for security analytics in the following way.

Metric monitoring—A custom CloudWatch metric is set up to monitor key IoT metrics such as incoming message rate, connection status, and data transfer volume. These metrics can help to detect abnormal behavior due to a security threat.

Log monitoring—IoT Core logs and monitors for authentication failures, unauthorized access attempts, and other security-related events. CloudWatch Alarms can be set up to trigger notifications for specific log events [6].

5. Simulation—Brute Force Attack on IoT Smart Door

The goal of simulating a brute force attack on an IoT door lock system is to try every possible combination of users and passwords in an attempt to gain unauthorized access until the right credentials are found. The first step in the procedure is that data on legitimate usernames and possible passwords are gathered. The assault is then automated by using either custom-built or pre-existing tools and scripts, which are used to configure parameters such the target system, lists of usernames and passwords, and any applicable constraints. The simulation runs, methodically testing every combination until either an attack is stopped by security measures or successful access is obtained. The analysis of the attack's outcomes, including any successful breaches or vulnerabilities found, is made possible by continuing to watch its development. After the simulation, vulnerabilities in the security of the IoT door lock system that have been found can be fixed by strengthening the system as a whole against possible threats and introducing improved authentication procedures.

5.1. Implementation

To demonstrate a security compromise in an Internet of Things house door lock system, the first step is to configure virtual IoT devices to mimic the features of the door lock system, including locking and unlocking. Then, these virtual devices are managed by AWS IoT Core, which makes sure that encrypted channels are established by using MQTT. Thereafter, strong authorization and authentication protocols are set up in AWS IoT Core to carefully control who has access to the door lock system by utilizing cutting-edge techniques like device certificates and IAM roles. Next, scripts are created to mimic different security breach scenarios, such as tampering or unauthorized access attempts, and to efficiently communicate with the virtual devices over AWS IoT Core for administrators. After that, in order to successfully interact with the virtual devices through AWS IoT Core, scripts or programs are written to simulate different security breach scenarios, such as unauthorized access attempts or tampering. In order to strengthen the security architecture, elements like logging and monitoring are turned on to carefully record device interactions and security occurrences. Services like AWS CloudWatch are integrated for thorough analysis. Additionally, reaction protocols are carefully put into place. These include using Lambda functions to lessen security breaches, which may involve locking down the door lock system or sending out instant alerts to administrators.

The Figure 7 shows the process flow of brute force attack simulation from a smart door lock enabled by IoT, where the aim is to find out the resilience of an IoT enablement system in the case of unauthorized access trials. It systematically tries out different combinations of usernames and passwords until access is gained into the system. The use of AWS services in the simulation is going to be targeted for device management in the same way as AWS IoT Core and AWS Lambda for the automatic configuration of the attack script through scripting, plus AWS CloudWatch for monitoring, as follows:

- IoT smart door lock: representing the target of the attack.
- Brute force attack script: It provides additional post-exploitation modules within the C2, automated with AWS Lambda, in trying to guess credential combos.
- AWS IoT Core: It decides all operations of communication and messaging between the smart door lock and the cloud.
- AWS CloudWatch: It logs every attempt and fires the alert on suspicious activities like multiple attempts of failed login.
- AWS GuardDuty: It provides threat detection that includes any anomalies related to the brute force attack.

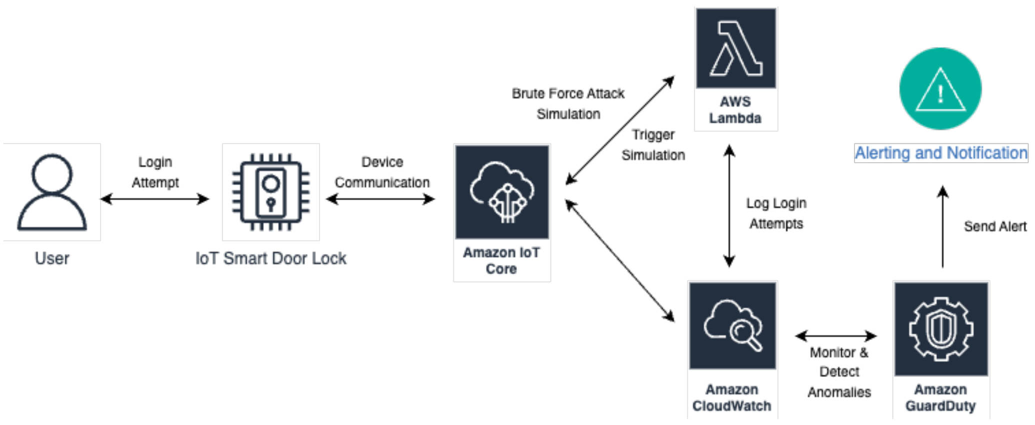


Figure 7. Process flow—IoT brute force attack simulation.

5.2. Results and Metrics

The brute force attack simulation was successfully executed with the use of various AWS services. The Lambda function successfully simulated the attack by attempting different combination of credentials.

Metrics like error count and success rate were collected (Figure 8) and stored to access the simulation performance. In order to ensure a quick response to possible security threats, CloudWatch alarms were set up to monitor unsuccessful login attempts, and an email notification alert was enabled when a particular thresholds was exceeded. Figures 9 and 10 shows the alarm generated by CloudWatch indicating the failed login attempt and the corresponding email notification that was send.

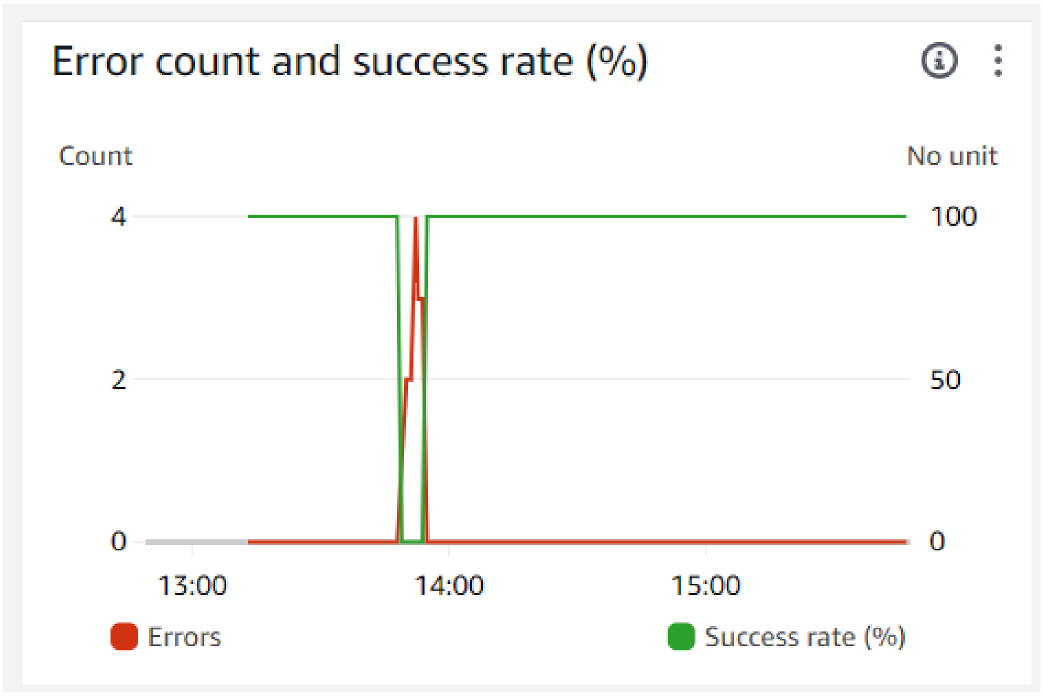


Figure 8. Error count and success rate with respect to the login attempts triggered by the brute force attack.

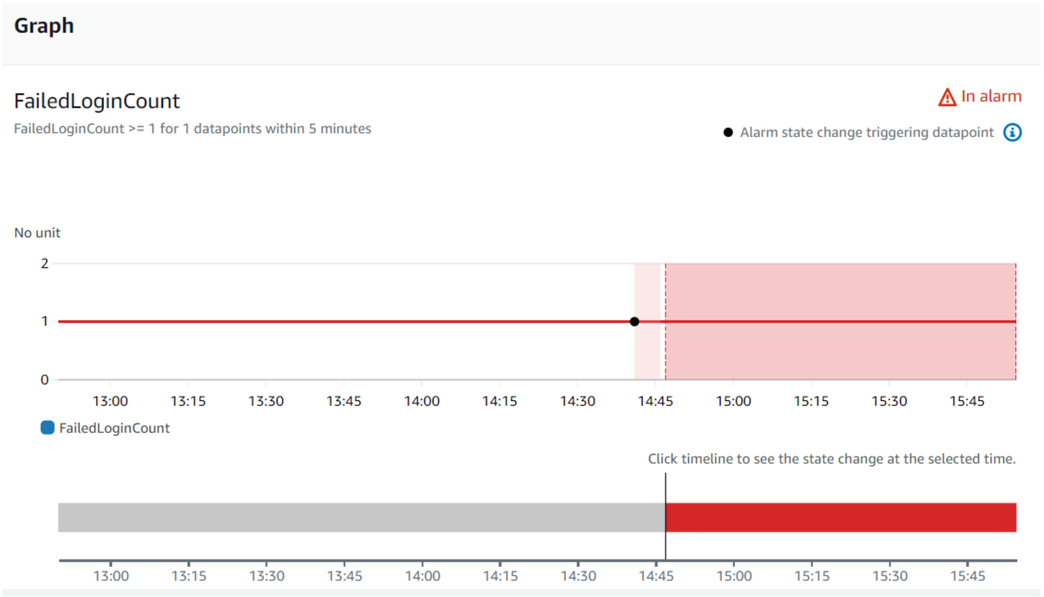
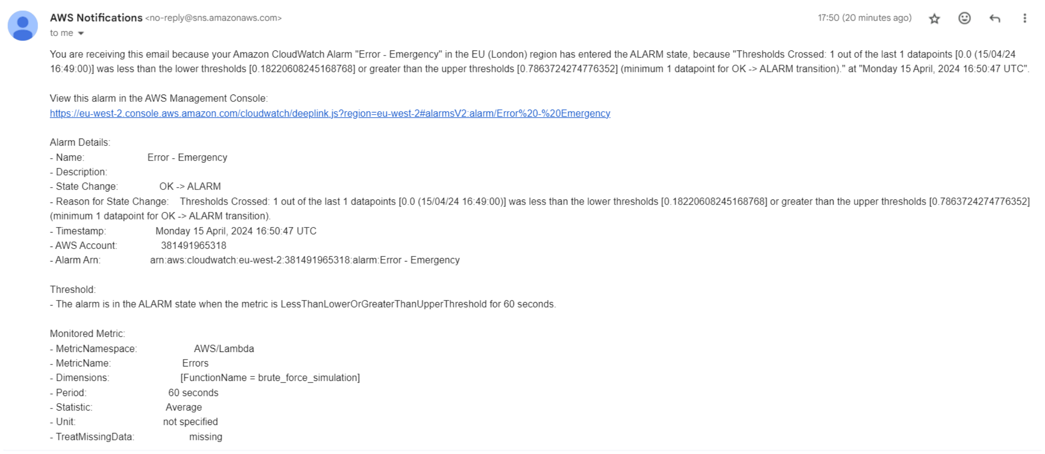

Graph
FailedLoginCount
FailedLoginCount >= 1 for 1 datapoints within 5 minutes
In alarm
Alarm state change triggering datapoint
No unit
2
1
0
13:00
13:15
13:30
13:45
14:00
14:15
14:30
14:45
15:00
15:15
15:30
15:45
FailedLoginCount
Click timeline to see the state change at the selected time.

Figure 9. Alarm generated indicating the failed login attempt.



AWS Notifications <no-reply@sns.amazonaws.com>
to me

17:50 (20 minutes ago)

You are receiving this email because your Amazon CloudWatch Alarm "Error - Emergency" in the EU (London) region has entered the ALARM state, because "Thresholds Crossed: 1 out of the last 1 datapoints [0.0 (15/04/24 16:49:00)] was less than the lower thresholds [0.18220608245168768] or greater than the upper thresholds [0.7863724274776352] (minimum 1 datapoint for OK -> ALARM transition)." at "Monday 15 April, 2024 16:50:47 UTC".

View this alarm in the AWS Management Console:
https://eu-west-2.console.aws.amazon.com/cloudwatch/deeplink.js?region=eu-west-2#alarmsV2:alarm/Error%20-%20Emergency

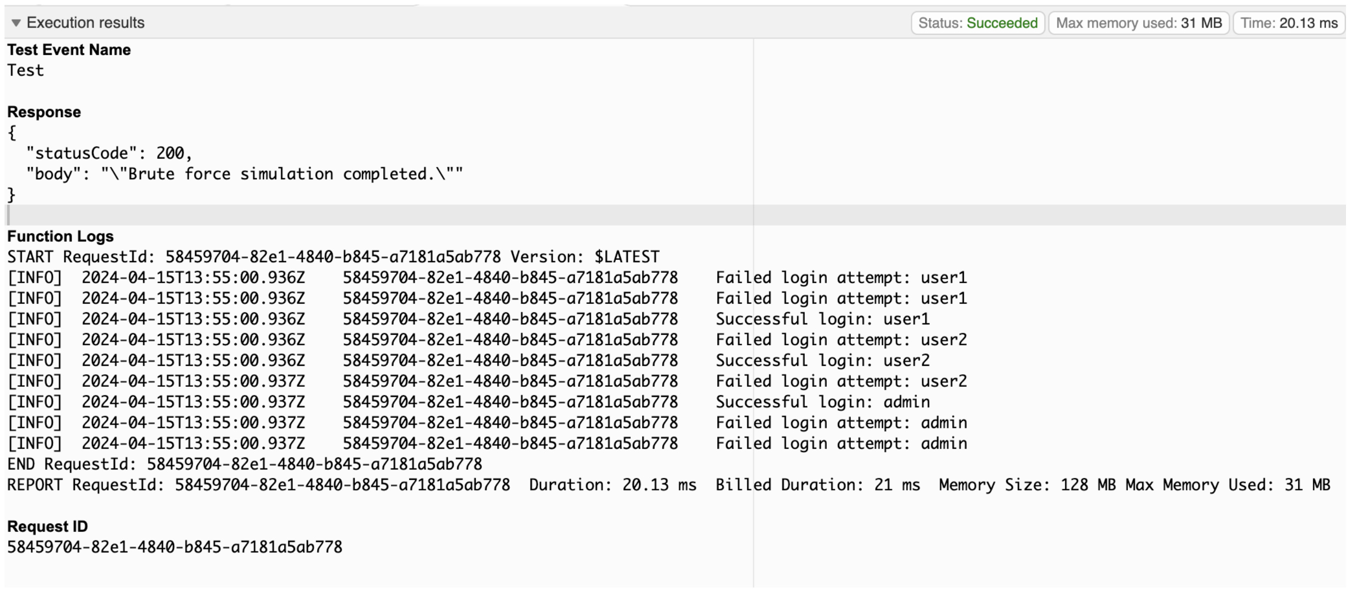
```
Alarm Details:
- Name:                      Error - Emergency
- Description:
- State Change:              OK -> ALARM
- Reason for State Change:   Thresholds Crossed: 1 out of the last 1 datapoints [0.0 (15/04/24 16:49:00)] was less than the lower thresholds [0.18220608245168768] or greater than the upper thresholds [0.7863724274776352]
(minimum 1 datapoint for OK -> ALARM transition).
- Timestamp:                 Monday 15 April, 2024 16:50:47 UTC
- AWS Account:               381491965318
- Alarm Arn:                 arn:aws:cloudwatch:eu-west-2:381491965318:alarm:Error - Emergency

Threshold:
- The alarm is in the ALARM state when the metric is LessThanLowerOrGreaterThanUpperThreshold for 60 seconds.

Monitored Metric:
- MetricNamespace:           AWS/Lambda
- MetricName:                Errors
- Dimensions:                [FunctionName = brute_force_simulation]
- Period:                    60 seconds
- Statistic:                 Average
- Unit:                      not specified
- TreatMissingData:          missing
```

Figure 10. Email notification.

The above figure shows the email notification sent when the threshold was exceeded. AWS IoT core was utilized to register the device securely and attach policies and certificates. Amazon GuardDuty was activated to detect suspicious activities related to the brute force attack, emphasizing the security of this environment in protecting the IoT infrastructure. The execution result is showcased in Figure 11 which demonstrates the entire outcome of the simulation attack on the IoT device.



▾ Execution results

Status: Succeeded | Max memory used: 31 MB | Time: 20.13 ms

Test Event Name

```
Test
```

Response

```
{
  "statusCode": 200,
  "body": "\"Brute force simulation completed.\""
}
```

Function Logs

```
START RequestId: 58459704-82e1-4840-b845-a7181a5ab778 Version: $LATEST
[INFO]  2024-04-15T13:55:00.936Z    58459704-82e1-4840-b845-a7181a5ab778    Failed login attempt: user1
[INFO]  2024-04-15T13:55:00.936Z    58459704-82e1-4840-b845-a7181a5ab778    Failed login attempt: user1
[INFO]  2024-04-15T13:55:00.936Z    58459704-82e1-4840-b845-a7181a5ab778    Successful login: user1
[INFO]  2024-04-15T13:55:00.936Z    58459704-82e1-4840-b845-a7181a5ab778    Failed login attempt: user2
[INFO]  2024-04-15T13:55:00.936Z    58459704-82e1-4840-b845-a7181a5ab778    Successful login: user2
[INFO]  2024-04-15T13:55:00.937Z    58459704-82e1-4840-b845-a7181a5ab778    Failed login attempt: user2
[INFO]  2024-04-15T13:55:00.937Z    58459704-82e1-4840-b845-a7181a5ab778    Successful login: admin
[INFO]  2024-04-15T13:55:00.937Z    58459704-82e1-4840-b845-a7181a5ab778    Failed login attempt: admin
[INFO]  2024-04-15T13:55:00.937Z    58459704-82e1-4840-b845-a7181a5ab778    Failed login attempt: admin
END RequestId: 58459704-82e1-4840-b845-a7181a5ab778
REPORT RequestId: 58459704-82e1-4840-b845-a7181a5ab778  Duration: 20.13 ms  Billed Duration: 21 ms  Memory Size: 128 MB Max Memory Used: 31 MB
```

Request ID

```
58459704-82e1-4840-b845-a7181a5ab778
```

Figure 11. Execution result.

6. Future Prospects

The future for both Docker vulnerabilities analysis, whether static or dynamic, and the enhancement of IoT security with AWS IoT Core, CloudWatch, and GuardDuty altogether is bright. Here are a few ways things could go:

- Increasing use of AI and ML: Artificial intelligence and machine learning technologies have been used for the betterment of security in the IoT, comprising devices using neural networks and decision trees. Accuracy in the detection of IoT security threats can be enhanced through these kinds of technologies. Additionally, such technologies are capable of reducing the rate of false alarms and automating security processes in order to detect and respond fast enough [22].
- Advanced tool development: Further development of new tools and techniques to analyze Docker vulnerabilities is in progress. According to some recent research, deep

learning algorithms can be used for the detection and classification of vulnerabilities in Docker images. This will enhance the accuracy and efficiency of vulnerability detectiongreatly [23].

- Dynamic analysis improvements: Increasing the application of dynamic analysis techniques, such as fuzz testing and symbolic execution-based vulnerability detection in runtime containerized apps, will enable one to detect threats that static analysis might miss [22].
- Edge computing integration: There is growing interest in how the integration between IoT security and edge computing really works. Edge computing allows for offloading processing or data analysis from the center, closer to IoT devices, hence reducing latency and improving reaction time. Research shows that edge computing can also be used for security monitoring, analysis, and faster threat detection and response modes [22].

7. Conclusions

In our research, we investigated the security issues of Docker systems and IoT platforms, and we employed Trivy and Falco for static and "runtime" security checking, which allowed us to make a product prototype visualizing the potential vulnerabilities in the digital ecosystem, providing a mechanism to stop attacks from escalating by blocking or terminating tainted Docker containers before they can infect the system.

Our results highlight the pressing need to implement advanced Intrusion Detection Systems (IDSs) to counter the growing number of complex cyber attacks aiming at Docker platforms, which will become even more prominent in the future with the advent of IoT environments. Categorizing and overcoming these threats requires more stringent and robust security requirements for both Docker and IoT platforms. These security policies, in combination with advanced IDSs, could ensure the robust security posture of the platforms. This, in turn, could better serve the emerging applications in the Internet of Things environment.

But, given the pervasiveness of the IoT in everyday living, this research also strongly suggests that further studies on identified security vulnerabilities are crucial. The sheer depth and scale of the IoT necessitate a continued research effort into the security of IoT devices, including in security by design to be incorporated into IoT devices, standardization of protocols, privacy-enhancing technologies and their application in IoT devices, and last but not least, regulatory policies and government interventions.

Our brute force attack simulation also showed this on a practical level: It demonstrated how essential it is to have good practices in place to prevent unauthorized access on all IoT systems, not just those involving the new generation of more powerful tools. The tools that exist today are fine, but they will need to be extended and refined to stay a step ahead of evolving threats. By investing now in research, policy making, and education, we can reduce the risks posed by IoT devices and build a safer and more secure digital future.

To conclude, we have seen that security improvements offered by using Docker and AWS tools provide us with a path forward toward a more secure digital landscape. Opportunities for further work might lay in continuing to create a secure infrastructure or else find novel ways to keep up against advancements in bad actors.

Author Contributions: Conceptualization, V.A., T.C. and C.C.; Methodology, V.A., T.C. and C.C.; Validation, V.A., T.C. and C.C.; Investigation, V.A., T.C. and C.C.; Writing—Original draft, V.A., T.C. and C.C.; Writing—review and editing, V.A., T.C. and C.C.; Supervision, A.T.K., V.C. and K.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is generated solely by authors efforts.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ferrara, P.; Mandal, A.K.; Cortesi, A.; Spoto, F. Static Analysis for Discovering IoT Vulnerabilities—International Journal on Software Tools for Technology Transfer, SpringerLink. 2020. Available online: https://link.springer.com/article/10.1007/s10009-020-00592-x (accessed on 30 January 2024).
2. Veijanen, J. Implementation of Security Best Practices on AWS Cloud: Case: Vulnerability Scanning of EC2 Instances and Networks. Urn.fi. 2020. Available online: http://www.theseus.fi/handle/10024/343321 (accessed on 30 January 2024).
3. Docker. Docker Security. Docker Documentation. 23 April 2021. Available online: https://docs.docker.com/engine/security/ (accessed on 20 January 2024).
4. Azrour, M.; Mabrouki, J.; Guezzaz, A.; Kanwal, A. Internet of Things Security: Challenges and Key Issues, Security and Communication Networks. 2021. Available online: https://onlinelibrary.wiley.com/doi/10.1155/2021/5533843 (accessed on 28 January 2024).
5. What Is AWS IoT?—AWS IoT. Available online: https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html (accessed on 12 February 2024).
6. Using Amazon CloudWatch Alarms—Amazon CloudWatch. Available online: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/AlarmThatSendsEmail.html (accessed on 13 February 2024).
7. What Is Amazon GuardDuty?—Amazon GuardDuty. Available online: https://docs.aws.amazon.com/guardduty/latest/ug/what-is-guardduty.html (accessed on 13 February 2024).
8. Karie, N.M.; Sahri, N.M.; Yang, W.; Valli, C.; Kebande, V.R. A review of security standards and frameworks for IOT-based Smart Environments. *IEEE Access* **2021**, *9*, 121975–121995. [CrossRef]
9. Singh, J.; Pasquier, T.; Bacon, J.; Ko, H.; Eyers, D. Twenty Security Considerations for Cloud-Supported Internet of Things. *IEEE Internet Things J.* **2016**, *3*, 269–284. [CrossRef]
10. Wu, Y. Basic Intrusion Technology of Industrial Internet of ...—Iopscience. 2021. Available online: https://iopscience.iop.org/article/10.1088/1742-6596/1738/1/012094 (accessed on 28 January 2024).
11. Malhotra, P.; Singh, Y.; Anand, P.; Bangotra, D.K.; Singh, P.K.; Hong, W.C. Internet of Things: Evolution, Concerns and Security Challenges. *Sensors*. **2021**, *21*, 1809. [CrossRef]
12. Aleisa, N.; Renaud, K. A Systematic Literature Review—arxiv. Available online: https://arxiv.org/pdf/1611.03340 (accessed on 28 January 2024).
13. Ziegeldorf, J.H.; Morchon, O.G.; Wehrle, K. Privacy in the internet of things: Threats and challenges. *Secur. Commun. Netw.* **2013**, *7*, 2728–2742. [CrossRef]
14. Karale, A. The challenges of IOT addressing security, ethics, privacy, and laws, Internet of Things. *Internet Things* **2021**, *15*, 100420.
15. Idrissi, I.; Azizi, M.; Moussaoui, O. IoT security with Deep Learning-based Intrusion Detection Systems: A systematic literature review. In Proceedings of the 2020 Fourth International Conference on Intelligent Computing in Data Sciences (ICDS), Fez, Morocco, 21–23 October 2020; pp. 1–10. [CrossRef]
16. Overview—Trivy. Available online: https://aquasecurity.github.io/trivy/v0.49/docs/ (accessed on 20 January 2024).
17. The Falco Project. Falco. Available online: https://falco.org/docs/ (accessed on 21 January 2024).
18. Dewo, K.T.; Yasin, V.; Budiman, T.; Sianipar, A.Z.; Yulianto, A.B. IT Infrastructure Dashboard Monitoring Application Development Using Grafana And Promotheus, a Case Study at Astra Polytechnic School. In Proceedings of the 2023 International Conference of Computer Science and Information Technology, Cairo, Egypt, 8–14 December 2023. [CrossRef]
19. MQTT—AWS IoT. Available online: https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html (accessed on 13 January 2024).
20. Building Lambda Functions with Python—AWS Lambda. Available online: https://docs.aws.amazon.com/lambda/latest/dg/lambda-python.html (accessed on 13 January 2024).
21. Amazon Web Services. What Is Amazon S3?—Amazon Simple Storage Service. 2023. Available online: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html (accessed on 13 January 2024).
22. Li, S.; Yang, X.; Zhang, Y. An Improved IoT Security Detection Method Based on Machine Learning. *J. Phys. Conf. Ser.* 2022, 012002.
23. Kim, S.; Kim, S.; Lee, S.; Lee, J. DeepLearning-Based Vulnerability Detection for Docker Images. *IEEE Trans. Comput.* **2024**, *73*, 605–616.