BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

TOOL FOR AUTOMATED TESTING OF WEB SERVERS

NÁSTROJ PRO AUTOMATIZOVANÝ TEST WEBOVÝCH SERVERŮ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. MICHAL RAJECKÝ**
AUTOR PRÁCE

SUPERVISOR **Mgr. KAMIL MALINKA, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

140493

Institut:	Department of Intelligent Systems (UITS)
Student:	**Rajecký Michal, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Cybersecurity
Title:	**Tool for Automated Penetration Testing of Web Servers**
Category:	Security
Academic year:	2022/23

Assignment:

1. Learn about cybersecurity issues, ways to secure web servers, and common vulnerabilities. Learn penetration testing methodologies and the OWASP standard.
2. Study existing penetration testing tools. Focus on their applicability for automated operation.
3. Design a tool for automated security analysis of web servers. The tool will integrate selected penetration testing tools and support their automated operation. The tool will also include reporting on discovered vulnerabilities.
4. Implement the tool according to the developed design.
5. Verify the functionality and reliability of the resulting implementation in a simulated environment.
6. Describe possible extensions.

Literature:

- OWASP methodology: https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf
- The Web Application Hacker's Handbook, 2nd Edition. by Dafydd Stuttard, Marcus Pinto.
- The Hacker Playbook 3: Practical Guide to Penetration Testing Kim Peter
- Python Web Penetration Testing Cookbook

Requirements for the semestral defence:

- Items 1 to 3

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Malinka Kamil, Mgr., Ph.D.**
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	1.11.2022
Submission deadline:	17.5.2023
Approval date:	3.11.2022

Abstract

This thesis delves into the topic of cybersecurity, with an emphasis on the security of web servers. It covers the technologies that are employed to protect web servers from a variety of common security threats. Furthermore, this work explores the detection of these weaknesses using various penetration testing methodologies along with OWASTP Top 10. A framework for automatic testing of web servers is developed in the practical part of the thesis. This framework integrates features from several tools and provides support for user-defined modules. Lastly, its functionality is verified in a simulated environment.

Abstrakt

Táto práca sa zaoberá témou kybernetickej bezpečnosti s dôrazom na bezpečnosť webových serverov. Zahŕňa technológie, ktoré sa používajú na ochranu webových serverov pred častými bezpečnostnými hrozbami. Ďalej sa práca venuje spôsobu odhaľovania týchto bezpečnostných hrozieb pomocou rôznych metodík penetračného testovania a zoznamu OWASP TOP 10. V praktickej časti je vyvýjaný framework pre automatizované testovanie webových serverov. Integruje funkcionalitu vybraných nástrojov a poskytuje podporu pre uživateľom definované moduly. V závere práce je funkčnosť nástroja overená v simulovanom prostredí.

Keywords

security, threat, vulnerability, internet, OWASP, webový server

Klíčová slova

bezpečnosť, hrozba, zraniteľnosť, internet, OWASP, web server

Reference

RAJECKÝ, Michal. *Tool for Automated Testing of Web Servers.* Brno, 2023. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Kamil Malinka, Ph.D.

Tool for Automated Testing of Web Servers

Declaration

I hereby declare that this Masters's thesis was prepared as an original work by the author under the supervision of Mr. Kamil Malinka. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Michal Rajecký
May 15, 2023

Acknowledgements

I would like to express my gratitude to my supervisor Mr. Kamil Malinka for his approach, advice, and professional assistance.

Contents

Chapter 1

Introduction

Just as communication is vital for humans, the most significant advantage of computers resides in their ability to communicate with each other. They allow us to store and retrieve information at any time. In the early days of the Internet, it served merely as a repository for static documents that were publicly available. There was no division between those who should have access to certain information and those who should not. Therefore, there was little concern for cyber-attacks, as attackers could not inflict significant damage. However, this landscape has gradually changed. The Internet now hosts confidential documents accessible only to chosen users, while dynamic applications have emerged as well. With this technological advancement, new opportunities for security threats continue to surface.

The importance of securing web servers increases with the volume of confidential information they handle. It's no longer just banks, security agencies, or other large organizations that rely on data security. Even regular internet users trust web servers with their personal schedules, details about their health status, or even the management of their assets.

To mitigate the risk of attacks on web servers, it's necessary to eliminate security vulnerabilities that allow attackers to compromise the system. To address these security flaws, they must first be located. Various penetration testing tools serve this purpose, ranging from the most straightforward command-line tools to complex frameworks covering a wide range of functionality in penetration testing. Despite their extensive functionality, these frameworks come with disadvantages, such as high cost, steep learning curve, and low parameterizability. This thesis aims to design and implement a tool for automated penetration testing of web servers, which will offer extensibility, a high degree of parameterization, and be free to use.

In the beginning, Chapter 2 describes the functioning of web servers, ways of securing them, and the most common security threats they face. Chapter 3 further delves into penetration testing, its methodologies, and the OWASP TOP 10 list. Existing solutions are covered in Chapter 4. Chapter 5 outlines the design of the developed tool, including functional and non-functional requirements, while Chapter 6 details the process of transforming the design into implementation. Finally, Chapter 7 describes the process of verifying the tool's functionality in a simulated environment.

Chapter 2

Networking and web servers security

Nowadays, the Internet provides space for many interests, such as product sales, banking, personal communication, data storage or business management. People trust the Internet with sensitive data, such as credit cards, home addresses, personal appointments or health status. This places high demands on the security of data and communications on the Internet.

Web servers served only as repositories for static information when the Internet was starting. There was no access control system, and people could access that information without authorization because there was no reason to. The web browser served only as a tool for accessing documents. Attackers would not get access to sensitive data, as all the information on the servers was public. One of the few things an attacker could do is modify these documents to spread false information.

In contrast, nowadays, static documents are replaced by web applications that generate content dynamically and where users with different privileges are often logged in. Thus, much of the data is confidential. Security threats have become one of the leading topics in the Internet world, as not only banks or government institutions rely on a high level of security but also ordinary users who believe their data will remain safe and won't be misused by third parties. [52]

2.1 Internet protocol stack

Just as human language is an important part of our existence, computer programs' ability to communicate with each other is a decisive factor of their power. Regardless of the application type, the data transportation methods don't vary too much. Nowadays, it's a sure thing for almost all users, but it doesn't imply that the means of data exchange are untouchable. [21]

Before diving into the topic of web servers and web applications security mechanisms and their security threats, the communication mechanism of computers will be explained. Protocols used in the process of transmitting data between computers, along with software and hardware network devices, are divided into several layers, each of which is responsible for specific tasks related to data transmission. With this division of the communication process, network architectures become scalable and easier to comprehend and maintain. [22] The following section briefly describes the layers.

Application layer

The application layer, positioned at the topmost layer of the Internet Protocol Stack, serves as the interface between users' applications and the underlying network infrastructure. It is responsible for handling the requirements and functionalities of various applications. Data transmitted at the application layer is encapsulated within messages, which include the actual information being exchanged along with any necessary application layer headers or metadata. Examples of protocols operating at the application layer include HTTP[1], DNS[2], SMTP[3] or TELNET[4]. The application layer ensures efficient network communication through these protocols, enabling users to interact with various internet-based services. [31]

Transport layer

The messages of the application layer are transmitted by the transport layer using either TCP or UDP protocol. On the one hand, TCP (Transmission Control Protocol) is connection orientated protocol which focuses on the reliable delivery of application-layer data. TCP provides, among other mechanisms, error correction and congestion control. It guarantees that all data sent is received by the other endpoint, making it a suitable option for applications requiring error-free message delivery and web browsing. On the other hand, UDP (User Datagram Protocol) is a lightweight protocol that does not guarantee reliable delivery of application-layer messages and is incapable of correcting errors. This connectionless service allows data to be delivered quickly and in large quantities if necessary. Its primal use is in real-time applications, like video streaming or online gaming.

Network layer

The network layer, also known as the Internet layer, is responsible for routing and forwarding packets, so-called datagrams, between hosts. It receives, along with the destination IP address, also the transport-layer segments from the upper layer, which are subsequently divided into packets, adding additional information to each of them, such as TTL (Time to live), source IP address, destination IP address and many more. According to the information in the routing tables, the network layer also identifies the most efficient and secure path for each packet to reach its destination. It plays a critically important role in securing connections between computers worldwide thanks to the IP protocol, which uniquely identifies devices on various networks and even inside networks of networks.

Link layer

The main responsibility of the link layer is receiving data passed down from the network layer and encapsulating them into frames which are then sent to the next node along the route. After reaching the node, the frame is passed up to the network layer. Different link-layer protocols provide different mechanics for data transmission, one of which is reliable

[1]HTTP (Hypertext Transfer Protocol) is an application layer protocol for exchanging data over the Internet. It's an essential building block for communication for the World Wide Web.[31]

[2]DNS (Domain Name System) translates domain names to their IP addresses. When a user enters a URL into their web browser, it must be translated into the corresponding IP address before fetching the website. This is only one part of the DNS system.[31]

[3]SMTP (Simple Mail Transfer Protocol) is designed to transmit email messages over the internet.[31]

[4]TELNET is an application layer protocol for remote control of a computer over the Internet by providing a virtual terminal. It's worth noting that TELNET is not considered to be secure.[31]

delivery to the destination point over one single link. It differs from the reliable delivery of TCP protocol, which occurs between end systems. Examples of the link layer protocols are Ethernet, WiFi or FDDI[5], just to mention a few of them.

Physical layer

In the physical communication layer, nodes transmit individual raw bits of data through a dedicated physical transmission channel, such as coaxial cables. This layer focuses on the physical aspects of data transmission, including the electrical and mechanical characteristics of the communication medium, signal encoding, and modulation techniques. The physical layer establishes the foundation for higher layers of the network protocol stack to communicate effectively.

2.2 Cryptography

As an integral part of the Internet infrastructure, web servers are constantly exposed to numerous threats and attacks. These threats can compromise the integrity, confidentiality, and availability of the data stored and transmitted. The three aspects mentioned, together, form a framework known as the CIA Triad (Confidentiality, Integrity and Availability Triad). Integrity means ensuring that the data is protected from unauthorized changes during transmission. Confidentiality pertains to preventing unauthorized access to data. Availability presents the assurance that authorized users have reliable and timely access to the data they need. Additionally, two more principles are considered extensions to this framework. First, authenticity refers to the assurance that information is from verified and trusted sources. The second principle, non-repudiation, is a way to guarantee that the sender of a message cannot later deny having sent the message, and the recipient cannot deny having received it.[49]

Starting at its core, cryptography is a method of protecting information by its transformation into an unreadable format. This encoded data, referred to as ciphertext, should only be reverted into a readable format with the appropriate decryption key. [46]

There are primarily two types of cryptography: symmetric and asymmetric. Symmetric cryptography (or secret key cryptography) utilizes a single secret value, also known as the key, for both encrypting and decrypting. While efficient and fast, its drawback lies in the problem of secure distribution of the key between communicating parties. Asymmetric or public key cryptography solves this issue using two mathematically related but not identical keys. First of them are the public key, which can be distributed widely without any security risks, and the private key, which remains in the possession of its owner. [50]

2.3 Defnese mechanisms

This section provides a selection of commonly used security mechanisms for securing web servers. In most cases of web servers, multiple mechanisms are combined to achieve the highest level of security. [57] [12]

- **Cryptographic Hash Functions:** Cryptographic hash functions, like SHA-256 (Secure Hash Algorithm 256-bit), are mathematical algorithms that take an input and

[5]FDDI (Fiber Distributed Data Interface)is a high-speed LAN (Local area network) protocol developed in the 1980s, which uses fibre optic cable for data transmission. [31]

return a fixed-size string of bytes, typically a hash value. They are designed to be a one-way function, meaning it should be computationally infeasible to reverse the process to reveal the original input. The unique hash value produced can authenticate data integrity. Any alteration in the original data, no matter how small, results in a drastically different hash.

- **Digital Signatures:** Digital signatures are cryptographic techniques used for validating the authenticity and integrity of data. They are created by encrypting a unique string derived from the data, also known as a hash, using a private key. The receiver can decrypt the hash using the sender's public key and compare it with the hash of the received data. If the hashes match, it verifies that the data has not been altered in transit and confirms the sender's identity. This mechanism is critical to secure digital communications as it provides non-repudiation, ensuring that the sender cannot deny sending the message, and data integrity, ensuring that the data has not been tampered with during transmission.[8]

- **Digital Certificates and Public Key Infrastructure (PKI):** Digital certificates serve as electronic documents that verify the ownership of a public key. They contain information about the key, its owner, and the digital signature of a Certificate Authority (CA) that affirms the certificate's validity. The Public Key Infrastructure (PKI) system enables entities to exchange information over untrusted networks securely. It manages the creation, distribution, and cancellation of digital certificates, establishing trust between the server and the client. This system ensures that the identities of entities involved in a digital transaction are verified.

- **Secure Sockets Layer (SSL) and Transport Layer Security (TLS):** SSL and TLS are cryptographic protocols that provide secure communication over a network. They use symmetric and asymmetric encryption to ensure data confidentiality and integrity. The protocols initiate a handshake mechanism to establish a secure connection, which involves the exchange of digital certificates for authentication, followed by generating and exchanging a shared secret key for encryption. This process ensures that all data transmitted between the web server and the client is encrypted and secure, protecting it from potential disclosure or tampering.

- **HTTPS:** HTTPS (Hypertext Transfer Protocol Secure) is a secure version of HTTP, the protocol for transmitting data over the World Wide Web. HTTPS combines HTTP with SSL/TLS to provide encrypted communication and secure identification of network web servers. It ensures that all communication between the client and the server is encrypted, preventing potential interception and modification of the data. This secure protocol is essential for sensitive data transactions, such as personal information or credit card details.

- **Key Management:** Key management involves the procedures and methodologies for the secure generation, distribution, storage, and disposal of cryptographic keys. It ensures that encryption keys are changed or rotated regularly to reduce the likelihood of successful brute-force attacks. Further, it defines the protocols for key recovery in case a key is lost or compromised and for the secure deletion of no longer required keys. Despite the fact that this is not a software algorithm like in the case of the ones discussed before, this approach is vital for server security. Even with the best

software protection, negligence by an organization's employees can lead to a security threat.

These mechanisms, while not exhaustive, represent some of the most important and widely used techniques in securing web servers. By applying these measures, organizations can significantly enhance their defences against various cyber threats and protect and protect their data.

It is worth mentioning that implementing cryptography into web servers can be challenging. The increasing computational power of modern systems has led to older encryption algorithms becoming susceptible to brute-force attacks. Therefore, it's essential to regularly update cryptographic algorithms and protocols to newer, more secure versions to keep the same level of security. [42]

2.4 Common web server vulnerabilities

In case of improper or incomplete implementation of security measures, a web server may be left vulnerable to potential security breaches. Despite the exhaustive attempts by security professionals to ensure full protection of the system, vulnerabilities may still appear, potentially leading to a compromise of the system. For this reason, consistent security audits and penetration testing are highly recommended, a subject that is further elaborated in Chapter 3. Among the common vulnerabilities of web servers are the following: [53] [41]

- **Injection Attacks:** This vulnerability happens when an attacker introduces harmful data into a command or query, tricking the interpreter into executing unintended commands or accessing unauthorized data. SQL injection, a prevalent form of this attack, involves the insertion of malicious SQL statements into an input field for execution, potentially leading to data modification or disclosure. A closer look at SQL injection is provided in Section 3.5.7.

- **Cross-Site Scripting (XSS):** XSS vulnerabilities allow attackers to inject malicious scripts into web pages viewed by other users. Upon viewing these pages, the scripts execute within the user's browser, potentially enabling the attacker to steal sensitive information such as session cookies or personal data. This could result in identity theft, unauthorized data access, or malicious actions executed under the victim's identity. A more detailed explanation of XSS vulnerability offers in Section 3.5.8.

- **Cross-Site Request Forgery (CSRF):** CSRF is an attack method that manipulates victims into submitting harmful requests. It leverages the identity and privileges of the victim to perform undesired actions on their behalf, such as changing their email address or making unintended purchases. This can result in unauthorized data modification, loss of data integrity, and even possible financial losses.

- **Insecure Direct Object References (IDOR):** IDOR vulnerabilities occur when an application provides direct access to objects based on user-supplied input. By manipulating these references, attackers can gain unauthorized data access. For instance, altering a parameter value in a URL could allow the attacker access to other users' accounts or sensitive data files. An example could serve the following URL: `https://bankingsite.com/useraccount?id=123457`, which, if the application is not properly secured, could give an attacker access to various accounts.

- **Security Misconfigurations:** These can occur at any level of an application stack, including the network services, platform, web server, application server, and database. Misconfigurations can provide unauthorized access to sensitive data or functionality, often allowing attackers to fully compromise the system.
- **Unvalidated Redirects and Forwards:** This vulnerability appears when an application redirects or forwards users to other pages or websites without adequately validating the destination. It can lead to users being redirected to phishing or malware sites or enable URL-based attacks that trick users into performing harmful actions.
- **Server-Side Request Forgery (SSRF):** SSRF vulnerabilities occur when an attacker can make a server send a request to other resources. This allows the attacker to interact with and send requests to internal resources, typically protected behind firewalls, resulting in exposure of internal systems and potentially sensitive data.

Chapter 3

Penetration testing

With the proliferation of web applications, the area of security vulnerabilities also increased significantly. The more complex the systems and the greater the user interaction, the more potential holes can be found in a given system, posing a challenge to system security. Attacks giving access to sensitive data or unrestricted access to the system pose some of the greatest threats. Another category of cyberattack focuses on overwhelming server resources to such an extent that the server becomes unable to respond to legitimate users.

Statements like „This site is secure“ can be found on many websites. Often, they make this claim based on SSL encryption (now commonly referred to as TLS), which is just one part of the necessary security. SSL/TLS only secures the communication between the client and server, but vulnerabilities can also occur in other places. Examples include Broken authentication, where the server fails to prevent attacks on login mechanisms, Broken access control, where the server does not adequately protect sensitive data; or SQL injection, which allows an attacker to insert specially crafted input into the input field, just to name a few.

One of the biggest threats to web application security is in accepting user input. It is not uncommon for a user to be able, or even required to enter an input, which can be maliciously modified to trigger an unexpected response on the server side that could lead to gaining unauthorized access to the system or trusted data disclosure. For this reason, application developers must assume that each input is potentially insecure and must ensure that it is handled correctly to prevent a successful attack. [44]

Users can tamper with the sequence of requests sent, manipulate cookies, request parameters, or even HTTP headers. It is common for attackers to craft input to trigger unexpected behaviour on the server. While SSL/TLS is a valuable tool for secure data transmission, it alone cannot prevent the forenamed attacks.

In the days when web applications didn't rule the Internet, an organization's security was based on securing the network and implementing firewalls. But that's very different today. For an application to work, the firewall must enable the connection through HTTP or HTTPS protocols while often connected to databases and mainframes. Usually, it is placed behind network-level security measures. If the application had a security flaw, an attacker on the public Internet could attack the organization's core system by sending crafted data to the application overcoming the network's defence mechanisms. Applications serve as a gateway for all kinds of attacks. Therefore, the security perimeter must have shifted from networks and firewalls to applications. Another type of threat lies in legitimate users accessing vulnerable applications. An attacker could aim at a system user and perform unauthorized actions on their behalf once a vulnerability is exploited. Nowadays, many applications provide their users with a password recovery feature, which generates a recovery

email without performing any further user verification. Thus, in case of compromising the user's email, gaining access to the system is trivial. [52]

As Gary McGraw once said:

> *„If you fail a penetration test, you know you have a very bad problem indeed. If you pass a penetration test, you do not know that you don't have a very bad problem.“*

3.1 Penetration testing explained

Penetration testing is a process used to simulate the methods used by potential attackers to bypass an organization's security measures. The objective is not only to identify present vulnerabilities but also to execute the attack, trying to gain access to all the resources that an attacker can possibly get to after a successful attack, including sensitive data.

Before attempting to compromise the target system, collecting detailed knowledge about the target is essential. This includes identifying devices within the network, their corresponding IP addresses, operating systems, installed software along with their versions, open ports, certificate analyses or cypher suite analyses, among other tasks. [55]

3.1.1 White box and black box approach

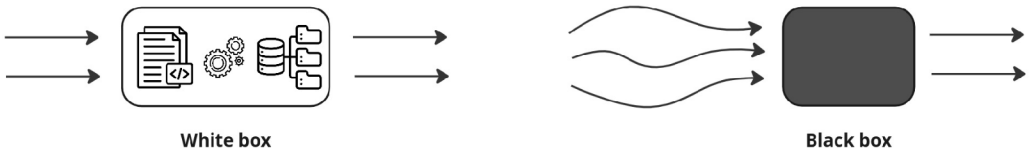
Generally speaking, to ensure the correct functioning of a software or system component, a tester needs to determine a set of conditions or variables, which will be tested and evaluated. It consists of input data, expected output, and execution conditions, all used to verify if the software meets the specified requirements. This model is also referred to as a „test case“. In the context of penetration testing, a test case refers to a specific scenario designed to probe and potentially exploit a particular aspect of the system's security.

In white-box penetration testing, the tester has complete knowledge of the system. The purpose is to simulate an attack from an insider on the network - someone with detailed knowledge of the system. Because white box testing requires a deep understanding of the codebase, developers or testers with a strong programming background often perform it. For this approach, a test case could involve a detailed examination of a certain part of the source code to look for vulnerabilities such as buffer overflows or injection flaws. It would also specify the section of code to be examined and the technique to be used.

This type of testing can be more thorough because it allows the tester to cover all possible paths and attack vectors. The benefits of white box testing include finding hidden errors and validating internal logic. However, it also has limitations, like possibly missing out on system-level or integration-level issues and being time-consuming due to the detailed analysis of each part of the code.

The second approach looks at the program as a function that accepts some inputs and provides output. The reason why this is called „black box“ testing can be found in the fact that the tester does not see the details of the implementation. In comparison to everyday human life, this is a common practice. Most people drive a car with just „black-box“ knowledge. Likewise, a person doesn't need to know the internal workings of an ATM to withdraw cash.

The test cases for black-box testing are designed without knowledge of the internal workings of the software or system. Instead, they are based on the system's requirements, functionality, and specifications. These test cases focus on inputs, outputs, and the expected



centring

Figure 3.1: Illustration of white-box testing, where the tester has access to the implementation of the software and other internal details, and a black-box testing, in which case the system is viewed just as a function that accepts inputs and returns outputs.

behaviour of the software from a user's perspective and could involve trying to access secured parts of the system without proper authentication or injecting malicious code through the user interface. Testers do not need specific programming knowledge to design or execute black box test cases; they only interact with the software's external interface. This implies the advantage that when the implementation changes, the test case does not change and is still usable. At the same time, the development of the implementation can proceed in parallel. [28]

The visual representation of the idea behind the two approaches can be found in Figure 3.1. [28]

3.1.2 Types of penetration tests

There are multiple options for approaching penetration testing, depending on the type of target, organizational needs, and other factors. The broad categories are described in the following text: [54]

- **Network Services Tests:** Network services tests, also known as infrastructure penetration testing, focus on identifying vulnerabilities in network devices such as routers, switches, and firewalls. The testing process involves assessing servers and hosts for vulnerabilities an attacker could exploit. This includes checking for misconfigurations, unnecessary services, outdated software versions, and other potential weak points within the network infrastructure.
- **Client-side Tests:** Client-side penetration tests focus on the applications and software installed on the client machine that interacts with the server-side applications. This could include web browsers, email clients, or any other locally installed applications. The primary aim is to identify vulnerabilities that could be exploited through mechanisms like phishing attacks, drive-by downloads, or other forms of social engineering.
- **Web Application Tests:** These tests are designed to identify security vulnerabilities in a web application. This includes reviewing the application for potential vulnerabilities as outlined in the OWASP Top 10 (described later in this chapter), such as SQL injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and security misconfigurations. These tests often involve automated scanning and manual exploitation to identify known and unknown vulnerabilities.
- **Wireless Network Tests:** Wireless network penetration testing involves evaluating the security of an organization's wireless networks. The tester will attempt to exploit

vulnerabilities related to the wireless protocols in use, weak or default passwords, and misconfigurations, as well as evaluate the effectiveness of wireless intrusion prevention systems (WIPS)[1]. [56]

3.1.3 Ethical and legal considerations

Penetration testing involves simulating cyber attacks on a computer system to identify vulnerabilities. While the intent is clean, the methods employed are indistinguishable from those used by malicious hackers. The question arises from the dual-use nature of penetration testing tools and techniques.

The most fundamental rule is that penetration testing should only be performed with explicit, informed consent from the system owner. Penetration testers must respect the scope of this consent, refraining from testing systems or using methods not covered in the agreement. Additionally, penetration testers may gain access to sensitive and confidential information during their work. They have an ethical duty to protect this data and not misuse them.

Except for the ethical question, legal frameworks must also be considered. They are not uniform around the globe as they vary from country to country. Still, in many jurisdictions, unauthorized access to computer systems is a criminal offence, possibly leading to legal consequences, regardless of the intent.

In addition to consent, penetration testers must also consider data protection and privacy laws. For example, the General Data Protection Regulation (GDPR) in the European Union imposes strict requirements on how personal data is handled, with severe penalties for non-compliance. Penetration testers who access personal data must ensure they comply with these requirements. [9]

3.2 Phases of penetration testing

There are various methods to approach the security of an organization's infrastructure. Typically, this process involves several phases, which begin with collecting information, continue with the attack, and conclude with the reporting phase. The phases can be generally outlined as follows[29][55]:

1. **Pre-engagement.** The pre-engagement phase is the initial stage of the penetration testing process and involves clarifying the client's requirements and expected outcomes. During this phase, it is important to determine the scope of the testing, what aspects of the organization are most critical, which areas require special attention, and if any sensitive devices or sites on the network should be excluded from the testing process. Additionally, other details such as the testing time window (whether it will be continuous or only at specific times of the day), the duration of the testing, the cost, and the pentest agreement, are serving as written authorization to perform the testing, need to be arranged.

2. **Information Gathering.** During the information-gathering phase, the focus is on collecting information about the target from publicly available sources. The goal

[1] A Wireless Intrusion Prevention System (WIPS) is a security solution that monitors a network's radio spectrum for malicious activities, such as unauthorized access or attacks and takes automated actions to protect the network.[56]

is to identify the security mechanisms implemented on the target, how it behaves, responds to certain requests, and so on. Although the activities of this phase may be less visible to the system (usually ignored as background noise), they are crucial to the success of the overall testing process. For more detailed information on the information gathering phase, please refer to the reconnaissance stage of the attacker kill chain in Section 3.3.

3. **Threat modeling.** The threat modelling phase involves creating attack scenarios using the information obtained during the previous phase. This involves identifying potential attack methods, determining which data and services are most likely to be targeted, and examining the target from the perspective of a potential attacker. The goal is to understand the target's architecture, potential security issues and points of interest for an attacker.

4. **Vulnerability analysis.** The vulnerability analysis phase deals with the probability of successful attacks. This phase often involves running vulnerability scans and conducting manual analysis to identify security weaknesses and determine which attacks are most likely successful. The goal is to identify vulnerabilities and select the most likely to have the biggest impact on the target and the highest probability of success.

5. **Exploitation.** The exploitation phase involves executing the attack. Once vulnerabilities have been identified, the next step is to exploit them to compromise the system and get access to the target's resources. Only those vulnerabilities should be targeted, which turned out to be present and truly exploitable. Blindly attempting to use all available tools is ineffective and may not yield valuable results. Therefore, the importance of obtaining accurate target information is again emphasized.

6. **Post exploitation.** The post-exploitation phase is a critical part of the overall testing process. Hence this phase provides valuable information, as it allows the tester to assess the potential damage that a successful attack could cause. Once a system has been compromised, the attacker will typically seek to expand their privileges, gain access to other systems, obtain passwords, or access sensitive organizational data, among other actions. Therefore, it is important to thoroughly evaluate the consequences of a successful attack during this phase.

7. **Reporting.**The reporting phase is the final and most important step in the penetration testing process. The testers provide a detailed report to the customer summarizing their findings. This report should contain what the testers were able to achieve, how they accomplished it, and, most importantly, what the organization needs to do to improve its security. By presenting the information from an attacker's perspective, the report provides valuable insights into how the organization can better secure the systems and prevent future attacks. Writing a good report is not an easy task. It should include technical and non-technical sections, such as an executive summary describing the testing process, scope, and key findings in language accessible to non-technical workers. The technical section should provide detailed information on specific vulnerabilities and recommendations for patching them.

In penetration testing, several methodologies exist, each focusing on slightly different objectives and procedures. Therefore, each contributes differently to the overall picture of a company's security. The above text offers a general division of the process into several

phases, providing a structural approach. Meanwhile, the following section is dedicated to another concept primarily focused on successfully attacking the target system without a specific stage dedicated to reporting.

3.3 Attacker kill chain

The „attacker kill chain“ is a concept developed by Mike Cloppert, the former director at FireEye company[2]. It can be described as a framework consisting of several stages of a cybersecurity attack, which must be followed to achieve the attackers' goals. After organizations understand and adapt this concept, they can implement more powerful threat detection and response strategies. Traditionally, the attacker kill chain consists of the following stages: [29] [45]

1. **Reconnaissance.** The basic idea of the first stage is best described by the adage: *„Reconnaissance time is never wasted time“*, well known among many military organizations, saying that before an attempt to attack the target, it's critically important to collect as much information as possible. Some sources say penetration testers should spend over 70% of the overall effort gathering information. This can be done in 2 ways: either passive or active. The **passive reconnaissance** is based on gathering information through non-violent means, such as social engineering, searching the Internet for publicly accessible data (employee names, for instance), exploring social media platforms or performing domain name system lookups. Since the goal is to better understand the target's infrastructure without using intrusive methods, passive reconnaissance is hardly distinguishable, if at all, from the ordinary behaviour of the users. It is important to keep in mind that data collected this way may be incomplete or outdated. The latter technique, the **active reconnaissance**, involves active probing of the target system or network through direct interaction. It involves host-discovery scans and port scans, vulnerability scanning, security headers analysis, or brute-forcing[3] directories and files. Active reconnaissance is quite noisy and can be easily detected by the target system and trigger alarms or alerts. If done without prior mutual consent, it can lead to legal consequences.

2. **Weaponization.** The goal of weaponization is to create a functional tool (the weapon) for exploiting a weakness in the target's system or network. This may involve crafting malware, developing means to target sensitive data and packaging it into a format suitable for the actual deployment against the target.

3. **Delivery.** The next step is to deliver the attacker's payload once a vulnerability or weakness has been discovered and exploited. This can be achieved by several methods, such as email phishing or social engineering, among others, trying to bypass security measures implemented by the target and trick the users into downloading and executing the payload.

4. **Exploitation.** exploitation, also known as the „compromise phase“, is the moment of the attack kill chain when the exploit is successfully applied. Techniques used in this phase include remote code execution, command injection or buffer overflow. It can be done in a single step, for example, by taking advantage of known system vulnerability

[2]FireEye is an intelligence-led security company. https://fireeye.dev/docs/about/fireeye/
[3]Brute-forcing is a trial-and-error method of guessing, usually the username and password[12]

thanks to which the attacker gains access, or it can take several steps, such as finding a hidden login page by brute-forcing files on a web server, gaining access by using wordlist created from the names of the organization's employee, etc.

5. **Achieve phase.** At this point, knowing the attackers' objectives is important. Other goals, except aiming for the target machine's root privileges, could include flooding the system by DDoS attack[4], sensitive data disclosure or horizontal privilege escalation, meaning that the attacker is trying to compromise as many accounts as possible (on the other hand, „vertical escalation" simply means to improve the access privileges to the highest possible level).

6. **Persistence.** After compromising the target and potentially gaining elevated privileges, it may be important to maintain persistent access to the target system. This can be done by installing backdoors, rootkits or other forms of malware, modifying system settings or creating new user accounts. This is the easiest phase for defenders to protect their systems or network.

To successfully achieve the goal in penetration testing, it is not enough to have multiple tools at their disposal; the integral part of the work of penetration testers is also understanding principles and methodologies, thanks to which become their work more precise and efficient.[10]

3.4 OWASP Top 10

This section covers OWASP guidelines for penetration testing, including OWASP Top 10. For that, the concept of CWE and CVE will be used. Therefore, their definition and the relationship will be briefly explained first. Common Weakness Enumeration (CWE) is a community-developed list that describes weaknesses in both software and hardware, which could lead to vulnerabilities exploited by attackers. The CWE records usually include a unique identifier (for example, CWE-862, which is Missing Authorization), describing the weakness and how it can be exploited. It is widely used in cybersecurity to identify and mitigate potential security holes in computer systems. On the other hand, Common Vulnerabilities Exposures (CVE) is a list of publicly known vulnerabilities present in certain software versions. A CVE contains its unique identifier, description, severity, and potential consequences.

There are several ways of looking at computer vulnerabilities and their potential consequences, one of them being OWASP Top 10. It's a list representing the most critical security threats that web applications currently face, according to OWASP. In 2021, a new version of OWASP Top 10 was released, merging some of the previous categories and introducing new ones. This text will only discuss the most recent version.

In contrast to the previous version of the OWASP Top 10, where vulnerability categories were selected based on the frequency of their occurrence and consequential manual edit by IT experts, the latest version released in 2021 uses a new approach. The new approach considers two aspects: how easy it is for someone to exploit the vulnerability (exploit score) and the amount of damage the vulnerability can cause (impact score). CWEs are

[4] A Distributed Denial of Service (DDoS) uses many compromised machines to flood the target system, preventing legitimate users from access. The main goal is to disrupt the normal operation of the system or network.

grouped and ranked according to their level of risk in combination with the severity score derived from the Common Vulnerability Scoring System[5].

The OWASP Top 10 consists of eight categories based on hard data and two resulting from data and experience from security experts. The first of the two approaches works with approximately 400 CWEs, categorized as either root-cause type (the CWE is the primary cause leading to a security issue) or symptom type (describing a direct impact and consequences).

On the other hand, security analysts may discover new vulnerabilities that require developing and implementing testing mechanisms, which can take considerable time, possibly even years. To ensure that the Top 10 remains up-to-date with modern trends and real risks of today, the latter two from the ten categories are based on points derived from questionnaires answered by security professionals. These experts can provide first-hand information that hard data may not yet capture. [2] [3]. The following section will introduce several CWEs that are, among many others, relevant to the OWASP Top 10.

1. **Broken Access Control.** The access control system ensures that users perform actions and access data that they are authorized to. Failure of this mechanism may result in access to unauthorized data and damages caused by actions performed outside the scope of authorization. Vulnerabilities falling under this category include granting access to anyone instead of specific roles and users and the ability to bypass control by modifying URLs, API requests or HTML pages, among others. Overall, in 94% of tested applications, a flawed access control system was detected. This category includes 34 Common Weakness Enumerations, such as CWE-35 Path Traversal [6], CWE-352 Cross-Site Request Forgery (CSRF) [7], and CWE-200 Exposure of Sensitive Information to an Unauthorized Actor [8], among others.

2. **Cryptographic Failures.** In the second place is a category, which is not the primary cause but rather a symptom of cryptographic errors. This often leads to the exposure of sensitive and confidential data. As mentioned in section 2.2, cryptography is an important mechanism to protect data. However, not all types of data require the most secure algorithms. The first thing to consider is the level of protection needed for specific types of data (for instance, passwords and credit cards are expected to have extra protection). The measures against cryptographic failure include identifying which data is sensitive according to laws, regulations and the organization itself and encrypting them, discarding this data as soon as they are not needed, ensuring the implementation of modern algorithms and disabling caching for responses that contains sensitive data. The most significant CWEs are CWE-259: Use of Hard-coded Password and CWE-327: Broken or Risky Crypto Algorithm. The first refers to using hard-coded passwords in products, which, if discovered, can lead to massive attacks on all organizations using the product as each installation contains the same

[5]CVSS (Common Vulnerability Scoring System) is a framework which assigns a score to security vulnerabilities in software based on several factors, for example, the potential impact on the system, the complexity of the attack or availability of mitigations[11].

[6]CWE-35: Path Traversal: '.../...//' is a weakness based on improper handling of pathnames. The absence of neutralizing of the slash sequences within the pathname can lead to the resolution to a location that is outside of the directory[1]

[7]CWE-352 Cross-Site Request Forgery is the incapability of a web application to sufficiently check whether a well-formed request was provided by the user who submitted the request.

[8]CWE-200 is a security flaw based on providing the unauthorized actor with data they are not authorized to have access to[37].

password [38]. The CWE-327 relates to cryptography and the use of old or insecure algorithms, which can lead to the exposure of critical data. Even algorithms once considered absolutely secure are becoming less secure due to the rapid advancement of computational power [39].

3. **Injection.**The danger of this category meets the reality when an application fails to adequately validate user input, uses dynamic queries in the interpreter, or directly uses or concatenates user input. The most well-known vulnerabilities include SQL injection, NoSQL injection, OS command injection [9], or ORM injection [10]. Ways to prevent these threats include automated testing of all input parameters, headers, URLs and cookies to identify and patch security holes, among others. To increase protection from this threat, developers can opt for a secure API without an interpreter or offer a parameterizable interface. Alternatively, using the SQL LIMIT command is possible to prevent the potential massive leakage of sensitive data. This category covers, for example, CWE-79: Cross-site Scripting (XSS), where the product fails at neutralizing an input controlled by a user, which is then saved to the web page and is accessible to other users. There are three main kinds of XSS; the first is stored XSS, where malicious code is injected and permanently stored on the server, serving all users who access the page. Secondarily, reflected XSS, where malicious code is injected into the URL and then reflected the user through a web application's response, and the last DOM-based XSS, where the payload is into the page's Document Object Model (DOM) instead of the server response, allowing the attacker to manipulate the web page's behaviour and interact with the user's data. Section 3.5.8 provides a closer look at this vulnerability. Another CWE covered under this category is CWE-89: SQL Injection, again a mishandling of user input that allows the user to modify an SQL statement with subsequent execution in the database. Section 3.5.7 discusses this vulnerability in more detail.

4. **Insecure Design.** This category was created in 2021 to address errors arising during product design. There is a difference between non-secure design and non-secure implementation. Even a secure product design can be followed by non-secure implementation, thus potentially introducing security holes which could be exploited. Conversely, a non-secure design will not be fixed by a perfectly secure implementation. Therefore, when designing software, it is important to agree with the client on data security requirements and determine the sensitivity and significance of the data sets. It should also be taken into consideration how exposed the software will be. Secure design is a way of developing software that considers potential security threats and tests the code to prevent known attack methods. It involves integrating threat modelling into development activities. To prevent insecure design, you can establish a secure development lifecycle with AppSec professionals, use secure design patterns and threat modelling or integrate security language and controls into user stories. This category includes security weaknesses such as CWE-522: Insufficiently Protected Credentials, which refers to situations where the product stores or uses credentials in an insecure manner that can be easily acquired by attackers[20]. Another example is CWE-209: Generation of Error Message Containing Sensitive Information,

[9]OS command injection makes the vulnerable application unintentionally execute an attacker's command as a result of improper input validation (from parameters or input fields) [51].

[10]ORM injection is a vulnerability where an attacker can manipulate ORM queries to access a database and therefore being able to read, modify or delete data [53].

which exposes sensitive information to potential attackers. Not only does this vulnerability reveal sensitive data about the environment or other users, but the error message contents could also help attackers to compromise the system, for example, by unintentionally guiding them to a successful SQL injection attack[19].

5. **Security Misconfiguration.** As the trend of highly-configurable software continues to grow, the fifth most common threat is Security misconfiguration, which was detected in 90% of the tested applications. To prevent this threat, it's important to properly configure permissions, disable unnecessary features and default accounts, and ensure that security settings are set to secure values in all parts of the application. Keeping the software up to date is, of course, also an integral part of prevention. One of the key CWE is CWE-16: Configuration, where weaknesses are typically introduced during the configuration of the software[16]. The second weakness, among many others, is CWE-611 Improper Restriction of XML External Entity Reference, where XML documents containing entities with URIs can lead to a resolution to documents outside of the designated scope. One example is Document Type Definition [11] which enables the definition of XML entities. It is possible to define an entity by providing a URI which contains a path to some resource (for instance `file:///etc/passwd`), which could allow access to files which are not reachable directly[17].

6. **Vulnerable and Outdated Components.** This category was in second place in the community survey. The product may be vulnerable to this threat if its components' versions are unknown, the software is outdated or vulnerable (including the operating system, database management system, etc.), and the security scans don't occur regularly. Prevention strategy includes removing unused dependencies and unnecessary features, regular scanning to identify vulnerable components, using only official sources, securing the correct configuration of updated components, and monitoring if the libraries are maintained and regularly patched. Only 3 CWEs are mapped to this category, one of which is CWE-1104: Use of Unmaintained Third-Party Components, where the product uses libraries and components that lack regular maintenance on the part of the developers. In that case, it is difficult to address and fix the vulnerabilities and other bugs[14].

7. **Identification and Authentication Failures.** This category, previously known as Broken Authentication, covers threats related to confirming the user's identity and authentication. The vulnerability may be present if the application allows repetitive guessing of credentials, permits default or weak passwords if it uses insecure password-recover mechanisms (like the knowledge-based answer), exposes the session identifier in the URL, or if the application uses plaintext or weak hashing algorithms for storing passwords. Part of the solution might be implementing multi-factor authentication, preventing product deployment with default passwords, and aligning user's password requirements with National Institute of Standards and Technology (NIST) guidelines [12], or ensuring that credentials-recovery mechanisms are resistant against account enumeration. Important CWEs are CWE-287: Improper Authentication, in which case the product fails to adequately verify the claimed identity of

[11] Document Type Definition (DTD) is a set of rules defining an XML document's structure, elements and attributes. It is also used to validate XML documents [26].

[12] NIST 800-63b is a set of guidelines for digital identity management that includes recommendations for creating and managing secure passwords [13].

the user[18], and CWE-297: Improper Validation of Certificate with Host Mismatch, when the product fails to verify that a certificate received from a host is associated with the sender, potentially allowing an attacker with a valid certificate to act like the trusted host[**?**].

8. **Software and Data Integrity Failures.** Software and Data Integrity Failures, the new category for 2021, occur when code or the infrastructure does not prevent integrity violations, for example, when an application relies on untrusted plugins, libraries, or modules. Insecure CI/CD pipelines [13] can introduce unauthorized access, malicious code, or system compromise. At the same time, the absence of sufficient verification of auto-updates might lead to the distribution of fake updates. To protect from this threat, OWASP recommends using digital signatures[14] to ensure the integrity of transmitted data, that libraries are being downloaded from trusted repositories, verify components for known vulnerabilities and implement a review process for code changes. This category maps CWE-494: Download of Code Without Integrity Check, which allows an attacker to execute a malicious program thanks to insufficient checks. Another CWE is CWE-829: Inclusion of Functionality from Untrusted Control Sphere, when a product imports executable functionality from an untrusted source (which can include web widgets, libraries, or other third-party sources). This can lead to malicious functionality being included, potentially resulting in severe consequences.

9. **Security Logging and Monitoring Failures.** This category refers to the importance of detecting, escalating and responding to security incidents and active breaches, which can be achieved by implementing proper logging and monitoring mechanisms. However, testing these techniques is not easy; the process usually involves interviews and asking if attacks were detected properly. Failures described by this category are likely to happen whenever errors and warning messages contain incomplete or unclear information, when logs are stored only locally, or when monitoring for suspicious activity doesn't take place at all. Countermeasures against this threat include logging all the unsuccessful login attempts with the broad context to be able to detect potentially malicious activity, storing logged data for an extended period of time to allow later analysis, ensuring proper encryption of logs to prevent attacks on logging systems, and implementing effective means of monitoring and alerting to be able to respond quickly on incidents, among others. This category covers, for example, CWE-778 Insufficient Logging when an organization logs a security-critical event with no or insufficient information needed for further analysis or does not log the events at all. One of the reasons is the additional cost an organization might have to pay. That way, it is challenging to detect malicious behaviour and therefore increase the chances of a successful attack[5].

10. **Server-Side Request Forgery (SSRF).** By exploiting a lack of input validation or insecure handling of user-controlled data, it is possible to modify a request, making the server execute the attacker's command. The result of the attack varies depending

[13] CI/CD pipelines, or Continuous Integration/Continuous Deployment pipelines, are processes used in software development to automate the building, testing, and deployment of software applications[23].

[14] A digital signature is a cryptographic method based on asymmetric cryptography (explained in Section 2.2) used to verify the authenticity and integrity of digital data. It creates a unique digital code that can only be generated by the sender and verified by the recipient using a public key.

on the specific vulnerability and what the attacker does with the access. Still, a usual case includes performing internal scans (open ports, for example) or reading files. As a preventive measure, it is possible to intervene at the level of several layers. Details on the operation of the individual layers are given in Section 2.1. At the application level, it is recommended to check user data, not send raw responses to clients, disable HTTP redirections or allow only explicitly defined URLs with specific schemas, ports, and destinations. At the network layer, it is possible to use „deny by default“ firewall policies, to log all accepted and blocked traffic on firewalls, or to separate remote resource access into different networks. The only CWE in this category is CWE-918: Server-Side Request Forgery (SSRF)[15], which is a category by itself, to raise awareness and draw attention to this threat.

To summarise this section, OWASP Top 10 is a useful framework for understanding the most significant web application security weaknesses. It categorizes these weaknesses based on hard data and expert experience, thereby staying up-to-date in a rapidly evolving cybersecurity field. However, it is crucial to remember that while the Top 10 provides a strong foundation, it is not exhaustive. For good security, it's important to stay updated on new threats and always be alert.

3.5 Reconnaissance in details

Since this thesis is centred on developing a tool for penetration testing with a focus on the data-gathering phase, the following text will provide an overview of the data collected for penetration testing purposes.

3.5.1 Open ports discovery

Identifying open ports and running services is key in the early stages of penetration testing. Open ports are gateways into a system and are usually linked to specific services, like HTTP on port 80 or HTTPS on port 443[15]. Each open port could be a vulnerability and a possible attack route.

Running services are active programs on a system, like a web or a database server, which, if outdated, can have known vulnerabilities. Knowing the open ports and active services significantly helps penetration testers plan their next steps and can be considered a critical part of the process.

3.5.2 DNS lookups

In the digital world, servers and computers communicate using IP addresses, not human-friendly names. When users visit a webpage, they typically enter a URI into the browser. This starts when a DNS server takes the domain name from the user's input and translates it into an IP address. This IP address sends the user's request to its final destination.

Because DNS servers play such a central role in internet use, they are a key topic in cybersecurity. Their job of turning domain names into IP addresses can create potential security issues. This makes understanding DNS servers and their vulnerabilities a critical part of the process .[31]

[15] In some cases, services run on non-standard ports, in which case it can be challenging to detect them.[52]

3.5.3 Word list creation

Websites are rich sources of words and phrases tied to their specific areas of interest. These words may appear in subdomain names or form parts of usernames and passwords. As such, generating a custom word list from a specific web page's content can benefit cybersecurity tasks. For instance, these custom word lists can aid in enumerating subdomains or brute-forcing usernames and passwords. This approach allows for targeted and efficient testing based on the unique content and context of the target website. [47]

3.5.4 Web source discovery

Discovering directories on the target server is an important part of penetration testing because it can reveal a lot of information about the server's structure, configuration, and potentially sensitive data.

Web servers often host numerous files and directories not directly linked to the site's main pages. Some of these directories might contain backup files, configuration files, or old versions of scripts, which could have vulnerabilities or sensitive information. By knowing the directory structure, a penetration tester can better understand the application's functionality and potential areas of vulnerability. [12]

3.5.5 Cipher suits analysis

A cypher suite is a set of algorithms that help secure a network connection that uses Transport Layer Security (TLS) or its predecessor Secure Socket Layer (SSL). The suite typically contains one cryptographic algorithm for each of the following tasks:

1. Key exchange: Determines how both parties - the client and the server - will agree on the secret key for that session (ex.: RSA, DHE, ECDHE).
2. Digital signature: Provides authentication, integrity and non-repudiation to the key exchange process (ex.: RSA, ECDSA).
3. Cipher: Determines how the data will be encrypted (ex.: AES, ChaCha20).
4. MAC (Message Authentication Code): Provides integrity and authentication to the encrypted data (ex: SHA256, Poly1305).

Cypher suite analysis examines the specific cypher suites a server configures for use. This helps determine if the server is configured to use strong cypher suites, if the server still supports outdated or weak cypher suites that could compromise the data's confidentiality and integrity and if the server is configured to prefer more secure cypher suites over less secure ones when negotiating encrypted connections. [44]

3.5.6 Analyzing software versions

It is common for many web frameworks to have security flaws discovered in a previous version, which were patched afterwards. However, if an outdated framework runs on the target machine, it is possible to use known misconfigurations or vulnerabilities. This is one of the reasons why it is essential to find out the framework and its version.

One of the options how to find out the framework is by inspecting the HTTP header sent by the server. This value is determined by the type of the server. However, the pentester

needs to keep in mind that the field can be modified or even disabled on the server side. That way, the attacker is given false information.

Another option is analyzing Cookies in the HTTP response header, where tempering with the content is less common but still possible.

Next, analyzing the source of the actual web page is an option where various indicators of the framework could be present, such as comments, framework-specific paths and variables, and so on.

From the point of view of the owner of a website, it is recommended to modify HTTP headers and cookies, remove unneeded lines of comments, delete unnecessary folders and files, and so on, in order to hide information about the technology and to make it less likely for an attack to happen. [54]

3.5.7 SQL injection

SQL Injection is a type of security vulnerability that allows an attacker to interfere with the queries an application makes to its database. Typically, it involves an attacker inserting malicious SQL code within a query, which can trick the system into executing unintended commands or accessing data without proper authorization. [41]

Here are the typical steps that an attacker might take when exploiting a SQL injection vulnerability:

1. Identification: The attacker identifies a point of input within the application (such as a user form) included in an SQL query.

2. Injection: The attacker provides input that includes malicious SQL statements. This is often done through trial and error, with the attacker observing the application's responses to refine their approach.

3. Execution: If the application does not properly sanitize the input (i.e., it does not ensure the input is safe before including it in an SQL query), the database may execute the attacker's injected SQL commands.

4. Exploitation: Depending on the nature of the SQL injection, the attacker may be able to extract sensitive information from the database, modify database data, execute administrative operations on the database (such as shutdown), or even execute commands on the operating system.

Finding an SQL injection vulnerability in the target system could lead to precious information, increasing the chance of a successful attack.

3.5.8 Testing HTTP Methods

HTTP (Hyper Text Transfer Protocol) in RFC 2616 [16] defines eight methods which serve web programmers when building a web application. They are HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT. If the server is misconfigured, some of them could be used by an attacker to temper files stored on the server or even obtain sensitive information.

Some HTTP methods mentioned above carry a certain level of risk as the attacker can use them. If the web developer decides to use them, a certain level of cautiousness needs

[16] https://www.rfc-editor.org/rfc/rfc2616.html

to be applied. Starting with `PUT`, this method servers for uploading files on the web server, which the attacker can use to get malware on the server. This could lead to remote code execution, among others. On the other hand, `DELETE` allows users to remove files on the server. An attacker could use the HTTP method `CONNECT` to use the server as a proxy. The Cross Site Tracing attack can be performed with the method `TRACE` allowed, which is meant to serve for debugging purposes. [25]

Cross-site scripting

Cross-site scripting, also known as XSS, was first defined in 2000 by a Microsoft security engineer and still represents a considerable threat in the world of web applications. Cross-site scripting attack is based on injecting the attacker's code into a web application so that legitimate users execute it unknowingly, which can lead to sensitive information leakage, executing unauthorized operations, and so on. [40]

There are several types of cross-site scripting attacks: `Reflected XSS`, `Stored XSS`, and `DOM-based XSS` [40].

The `Reflected XSS`, or non-persistent XSS, involves modifying a legitimate HTTP request to a vulnerable web server in such a way that it injects a malicious script into the request. This could be, for example, within the parameters of a URL. The attack is typically propagated through social engineering techniques, convincing the victim to open a crafted URL. Once opened, the victim's browser sends the injected request to the server, which, if vulnerable, reflects it to the victim's browser. As the browser processes the returned code, it executes the malicious script. This script, often written in JavaScript, can be used for various malicious activities such as installing keyloggers, stealing cookies, or defacing the web page. [**?**]

In contrast, the `Stored XSS` attack, also known as a persistent attack, injects a malicious script directly into the server. If the server fails to validate the input properly, the malicious script is stored and executed each time a user accesses the associated URL. The malicious code persists on the server until it is removed.

The third category, `DOM-based XSS`, is neither reflected nor stored. The underlying issue is that the server uses DOM objects, such as `document.location` or `document.URL` that are not fully controllable and can be manipulated by the attacker. Unlike other types of XSS attacks, detection techniques often overlook DOM-based XSS as they typically focus on injecting the HTTP request/response, which is irrelevant to this type of attack. Consequently, the detection often fails in such cases. [30]

Chapter 4

Existing solutions

The following text explores existing tools used in penetration testing, focusing on those being used, especially in the reconnaissance phase of the process, and with a secondary emphasis on their automation potential. The penetration testing process employs a wide range of tools across different categories, from passively collecting publicly available information to more aggressive forms of target analysis, all the way to deploying the actual attack, which involves compromising the system itself. The selection of the most widespread tools will be described in this section, which at the same time will serve as a reference point for the design and implementation of the tool being developed in the practical part of this thesis.

Nmap

Nmap (Network mapper) is an open-source tool for collecting network-related data. Despite the fact that its primary function is to scan large networks, it can also be effectively used against individual targets. It can provide the user with precious information regarding the „up“ hosts, operating systems and software versions, open ports and much more. In the context of the tool developed within this thesis, Nmap is a viable option for several purposes. [34]

Obtaining information about which targets are online is the initial step in security testing, and for this, Nmap offers a variety of options and techniques. While the host discovery process is sometimes called a „ping scan,“ Nmap can go beyond simple ICMP scans and utilize other techniques, such as TCP SYN or TCP ACK packets.[33]

Moreover, Nmap offers many other useful functionalities such as firewall bypassing, backdoor detection, checking for known vulnerabilities, scripts to analyze specific services and much more. [35]

Masscan

Masscan is an open-source network scanning tool that's highly revered in the field of penetration testing. As one of its advantages can be considered its high speed. Using an asynchronous transmission technique, Masscan is capable of transmitting millions of packets per second, which allows it to scan a multitude of ports on multiple IP addresses concurrently.[43]

Thanks to its scalability and adaptability, Masscan can be used effectively in small-scale and large-scale operations, making it possible to map active hosts, track the distribution of malicious botnets across the internet or analyze certain services. Since Masscan is a

community-driven tool, users can modify it according to their requirements. This transparency and adaptability make it a valuable part of a job in the cybersecurity domain. [48]

There is a claim about Masscan's capability to scan the internet in under 6 minutes, made by its creator, Robert David Graham. According to Graham, under ideal conditions and with sufficiently powerful hardware, Masscan can send out around 10 million packets per second, fast enough to scan the entire internet's IP addresses in a few minutes. [24] [7] [6]

However, in practice, the time to scan the entire internet would likely be much longer due to a variety of factors, including network congestion, packet loss, the need to wait for responses, and the potential for such rapid scanning to be classified as a denial-of-service attack and subsequently blocked.

CeWL

Cewl is an open-source custom word list generator crucial in penetration testing. It works by spidering a target website's URLs to extract unique words, forming a list, which can be used in password-guessing attacks such as a dictionary or brute force attacks. In each case, the result contains words relevant to the specific field, target's language and topic, thereby increasing the chance of success in password attacks. [4] Cewl provides the user with minimal or maximal word length, contributing to overall versatility. It also offers options to consider or disregard email addresses and numbers during extraction. Cewl's capabilities make it a valuable tool, increasing the effectiveness of password-cracking methods in cybersecurity. [29]

Dnsrecon

DNSRecon is a robust DNS enumeration tool for information gathering and network mapping. It is designed for querying DNS records associated with a target network. The records queried can include SOA, NS, A, AAAA, MX, TXT, and SRV records, providing a detailed view of a network's DNS setup[1]

DNSRecon's functionality is wider than just enumerating DNS records. It also allows for zone transfers to identify hosts in a network, checks for wildcard resolution, attempts reverse lookups on netblocks, and even performs brute-force attacks against subdomains and hostnames.

DNSRecon can extract a significant amount of data about a target, which can then be utilized to identify potential weak spots in a network's configuration. Its versatility in DNS enumeration makes it a vital resource in the reconnaissance phase of penetration testing. [41]

Shcheck

SHCheck (a shorter version of Shell Checker) is a penetration testing tool utilized primarily for checking HTTP headers and cookies. Its main function is to assess the security of

[1]The Domain Name System (DNS) is a decentralized system that translates human-friendly domain names, like „www.example.com“, into their corresponding IP addresses, which computers use to locate each other on the internet. DNS records are entries in a DNS database that provide information about a domain, including its IP address (A or AAAA record), mail servers (MX record), and many others. Among other sources, more details on DNS can be found in [32].

HTTP headers in web applications. By analyzing the HTTP response headers, SHCheck can provide information about potential security vulnerabilities in a web application's setup.

SHCheck's capabilities extend to examining security mechanisms like HTTP Strict Transport Security (HSTS), X-Content-Type-Options, and X-XSS-Protection. Additionally, it can analyze cookies to check for flags such as HttpOnly and Secure, which can have a major impact on the security of a web application.

Overall, SHCheck provides a simple method to assess the security of HTTP headers and cookies. [27]

Whatweb

WhatWeb is a flexible and comprehensive web scanner specializing in identifying web technologies. It is used to collect information about a target website's technology. This includes information about the type of web server, scripting languages, content management systems, and even specific plugins being used.

WhatWeb has an advanced plugin system, offering over 1800 of them. Each plugin is designed to recognize and report specific technologies. Therefore WhatWeb can identify a wide range of systems and software. Additionally, WhatWeb supports an aggressive mode which uses additional techniques, such as error-based detection, for a more in-depth analysis. [44]

Sslscan

SSLScan is used for evaluating a network service's SSL/TLS configuration. It is widely used in penetration testing and cybersecurity, as it can identify vulnerabilities within an SSL/TLS setup.

The core functionality of SSLScan includes determining the SSL protocols (e.g., SSLv2, SSLv3, TLSv1, etc.) supported by a server, as well as the specific cipher suites it can use. It also identifies server preferences, certificate information, and potential vulnerabilities associated with weak cypher suites or protocol versions.

Since communication is a fundamental part of a computer's characteristics, it is essential to keep it safe. By discovering misconfigurations and weaknesses in the target systems' SSL/TLS setup, SSLScan can reduce the risk of data breaches. [36]

Issues associated with complex penetration testing frameworks

In this context, the category of complex penetration testing systems offering extensive functionality can not be missed. While these systems provide undeniable advantages, they also come with certain drawbacks that can be decisive in specific situations:

- **Cost:** Although some of these tools offer free or community versions, the full-featured versions typically require a paid license. This can pose a barrier for individuals or smaller organizations with limited budgets.
- **Complexity:** These tools encompass many features and capabilities, resulting in high complexity. Consequently, learning to use them effectively can demand a significant time investment, particularly for beginners.
- **Limited Parameterization:** Some tools may have restricted options for configuring scans or filtering results, which could be limiting in certain testing scenarios.

- **Insufficient Extensibility:** These tools may provide a given functionality but lack the possibility to extend it further. This limitation can constrain in certain cases where customization or expansion is required.

Nessus, developed by Tenable, Inc., is a renowned vulnerability assessment tool with a vast set of features, including vulnerability scanning, configuration audits, and asset profiling. Nessus provides over 100,000 plugins, each designed to detect a specific vulnerability or a set of vulnerabilities. Despite its extensive capabilities, Nessus comes with certain limitations. Its complexity might pose a steep learning curve for beginners. Additionally, while a powerful tool, the full-featured version of Nessus can be costly, which might become the reason why an organization chooses a different solution.

OpenVAS, an open-source vulnerability scanner and manager, stands out for its affordability and comprehensive vulnerability detection. It has a routinely updated database and offers a user-friendly web interface and customization possibilities for scans. Drawbacks include limited professional support, confined to the paid version, and a potentially complex and long-lasting setup process. Additionally, when managing large networks, its performance may be outpaced by commercial tools.

As the last penetration testing solution on this list, **Burp Suite** will be mentioned, a web application security testing platform from PortSwigger. It offers many tools, including a proxy server, web spider, scanner, and more. Among other advantages, it provides an intuitive interface and extensive customization options. However, the drawbacks include a potentially steep learning curve for beginners, costliness for the feature-rich version, and limitations in non-web application penetration testing areas.

Chapter 5

Technical specifications and design

As described in Chapter 3, penetration testing can take many forms, but data collection plays a key role in each of them. This thesis aims to provide penetration testers with a software solution for data collection on targets, which will be modular, parametrizable, and free to use.

This chapter presents the requirements and design of the tool. Firstly, the general requirements that the tool must meet are described, followed by a breakdown of both functional and non-functional requirements. Finally, the tool's design is laid out with accompanying figures.

The minimum requirement for the design of this automated penetration testing tool is the integration and automation of chosen tools (as outlined later in this chapter). The tool will be capable of providing reports on information about the targets and found vulnerabilities while also listing other potential points of interest for penetration testers. Additionally, it will offer options for expanding functionality through modules.

The final goal is to get as much data as possible on all targets. Penetration testers will use this information to make the process as accurate as possible.

5.1 Functional requirements

This section details the main functional requirements for the proposed penetration testing framework. These are the essential tasks that the framework needs to carry out to support penetration testing effectively. They're designed to make the framework user-friendly, efficient, and valuable to penetration testers.

- **Target Scanning**: The tool should be able to scan specified targets, which could be IP addresses or domains. It evaluates whether the target is up, performs host discovery and identifies open ports and running services.
- **Generating custom word lists**: Ability to generate custom word lists, which can be later used in dictionary attacks on folders, directories or user accounts.
- **SSL/TLS configurations**: Analysis of the servers' security headers and SSL/TLS configurations.
- **DNS lookup**: The tool provides the user with a reverse DNS lookup option.
- **Reporting**: The tool should generate comprehensive reports detailing the penetration test findings. These reports should be clear and understandable.

- **Module Support**: The framework should support the integration of additional tools or modules to extend its functionality.
- **Configuration and Customization**: The framework should provide options for customization, allowing testers to adjust its behaviour based on their specific testing requirements.
- **Automation**: The framework should support automation for regular, scheduled penetration tests.

Use-cases and scenarios

Next, real-world situations or 'use cases' will be examined. These use cases will better explain the application of the concepts discussed above.

- **Gathering information on targets.** If the user knows a target address or possesses a list of addresses, they will initiate a scan on these targets:
  1. The user selects desired modules in the configuration file, determining which scan will be executed.
  2. In case of multiple targets, a text file containing the list of the addresses is created.
  3. Through command line options, the user decides whether to save the program's output to a file.
  4. Once the tool starts, it analyzes the chosen targets, prints the information via standard output, or saves it to a file.
  5. The results are formatted to be easily readable.
- **Host discovery.** Let's consider a situation where a penetration tester has access to a single computer within an organization's network or has been granted network access from their device. The objective is to gather information on as many potential targets as possible:
  1. The user enables the „host discovery“ feature in the tool to scan the subnet.
  2. The user then selects the modules to be executed on all discovered targets.
  3. After running the program, the results are displayed through standard output or saved to a file.
- **Adding a user-defined module.** If the process of penetration testing demands new functionality through an already integrated tool, a tool which is new to the framework, or based on features provided by the Python programming language:
  1. User decides to add new functionality.
  2. Thanks to its extensibility, users can develop a new module by creating a parser for the chosen tool's output and formatting the results according to their preferences.
  3. A reference to the newly developed module is added to the mapping file, enabling the framework to recognize and execute it.
  4. This newly created module can be switched on or off for each future run.

5.2 Non-functional requirements

The focus of this section turns to non-functional requirements, which also represent an essential part of the overall quality of the framework.

- **Extensibility:** It will be possible to add functionality easily without changing the existing architecture. The design presumes that new tools may be developed, and their functionality could be desirable in this framework. It also considers that each penetration tester has different preferences and favours different tools.
- **Modularity:** The framework's functionality comes from several components (modules) that can be freely enabled or disabled at startup. The user has the option to add modules according to their requirements for functionality. Modifying the existing modules without affecting the rest of the architecture is also possible.
- **Portability:** Given the widespread use of the Linux operating system in penetration testing, the program will be designed specifically for this operating system.
- **Fault Isolation:** As this involves integrating several tools under one roof, it is desirable that a fault in one of them does not cause the entire framework to fail. Each of the selected tools is well- maintained and widely used, which could lead to updates that might cause, for example, a change in the output format and, therefore, incompatibility with the parser of a given module. Modifying a specific module is sufficient for any repair without the need for intervention in the entire system.

5.3 Design

From a high-level view, the tool's core will consist of a set of Python files responsible for the framework's functionality and the integration of all modules. The input to this „Python core“ will be the target or a list of targets, along with a configuration file specifying which modules should be executed. Subsequently, the Python core will perform port scanning on the specified targets and run the selected modules on the running services. Each module covers the specific functionality of a Unix tool, which is encapsulated in a Docker image and executed by the framework. The outputs of the tool's execution in the Docker images are processed and presented to the user in a comprehensive format on the standard output or/and written to a file. This is also shown in Figure 5.1

Modularity

The tool will be highly extensible thanks to the ability to add custom modules. Choosing a new combination of parameters for an already integrated tool or adding a new one will be possible. As for the second option, a docker image must first be created. Adding modules for both integrated and new tools is the same. A new parser needs to be constructed so the output can be processed correctly, along with a simple function which allows the program to use the module. Then all that remains is to add the new test option to the configuration file.

The tool will consist of multiple modules, each addressing a specific functionality. Each module will be composed of the following components, as also shown in Figure 5.2

1. **Record**, which servers as a mapping file linking all the components together

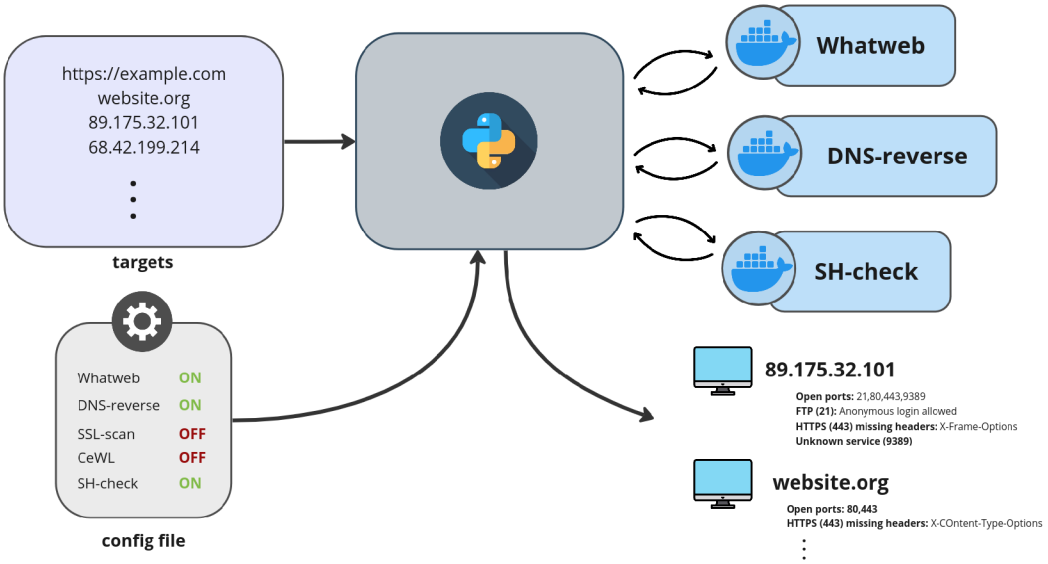


Figure 5.1: High-level overview of the architecture of the tool. The main Python program accepts user input in the form of targets and also reads the scan configuration from the corresponding file. Based on these instructions, it runs the relevant Docker images and then outputs the target scan results on the standard output or into a file.

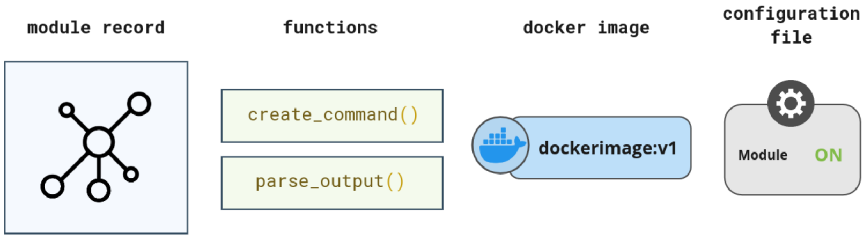


Figure 5.2: Overwiev of a module. In general, it consists of four parts. First is a module record, which serves as a mapping file containing information about which docker image should be run, which functions to use, for which service it is designed, etc. Second, are the functions which create the command for the docker image and parse its output. Next is the docker image itself. Finishing with the last part of a module, an entry in the configuration file, which switches on/off the module.

2. **Unix tool** packaged in a docker image.
3. **Functions** needed for the module to be operational
4. **Parser** to interpret the tool's output.

Chapter 6

Implementation

This chapter details the process of converting the design into program implementation. The selected technologies for the tool's implementation will be discussed, followed by a description of the module architecture and its connections. Next, details regarding the possibility of extending functionality with user modules will be provided, and finally, the challenges associated with developing this framework will be addressed.

6.1 Technologies used

Docker was chosen as it provides an efficient and lightweight way to create isolated and reproducible environments, so-called containers. These containers can run anywhere Docker is installed, supporting portability. With Docker, individual tools within the penetration testing framework can be isolated in separate containers, reducing the risk that a failure or security issue in one tool will affect others. This approach aligns well with the principle of fault isolation.

As for the solution of a programming language technology, Python3 was chosen for this project. Its simplicity and readability make it a good choice for many users. Python3 has a rich ecosystem of libraries and modules, many of which are network-focused and security-focused and can be easily incorporated into a penetration testing framework. The support for Docker was also a key factor. Furthermore, Python3, being the latest version of Python, includes many improvements and features not found in Python2 and is the version that currently receives updates and bug fixes.

6.2 General overview

The developed framework is based on the Python programming language, specifically version Python3.10, which is one of the minimum requirements for the framework to be operational.

A distinguishing feature of the tool is its extensibility with user-defined modules. Hence its architecture is fundamentally modular. It is comprised of a collection of Python files collectively responsible for the entire operation of the framework. The following describes the key libraries and functions that the framework employs:

- **argparse:** One of the initial steps in launching the framework involves processing command-line arguments facilitated by the Python library argparse. This library is user-friendly and offers a wide range of argument processing options.

- **importlib:** To run selected modules, dynamic importing is employed. Each run may theoretically contain a different path to the required function within the module, facilitated by the importlib library.
- **appdirs:** The appears library is used to identify the folder where a specific operating system stores configuration files so that the framework's configuration files can also be placed there.
- **Python function getattr():** In addition to dynamically specifying Python modules, calls to dynamically specified functions are also performed. The specific function used to create the Docker command and the function used to process the raw Docker output is unknown at the beginning of the framework's operation. The Python function getattr() is employed for this purpose.
- **configparser:** The configparser library, which provides an intuitive format for configuration files and their simple processing, forms the basis for the configuration files.
- **docker:** The Docker library runs the Docker image with the appropriate parameters.
- **xmltodict, json, termcolor :** The Docker image run provides the output of the specific tool, which needs to be processed and formatted. Several libraries are utilized here. The libraries xmltodict, and json offer easy handling of these data formats. The term colour library is used for formatting the parsed information.

Those mentioned above are only selected important modules; the framework uses many others, most commonly used for basic tasks such as file and system operations.

6.3 Modules

The following text will describe the functionality covered by the modules. As previously mentioned, the tool's capabilities can be expanded. Thus, the selection of integrated tools isn't neither definitive nor limiting. Each user is free to extend the tool's functionality in the required direction. The modules described in this chapter represent what a penetration testing tool should cover.

6.3.1 Defining module

The whole tool is composed of several modules. Visual representation of modules is shown in Figure 6.1 and will be explained in the following text. The tool is designed to easily add another module according to the user's specific needs. In general, each module consists of:

1. A record in `dipmodules.py`
2. A docker image of the chosen tool
3. Necessary Python functions
4. An entry in the configuration file

Docker image is not mandatory; it can have the value of None. Each of these parts will be explained in the following text:

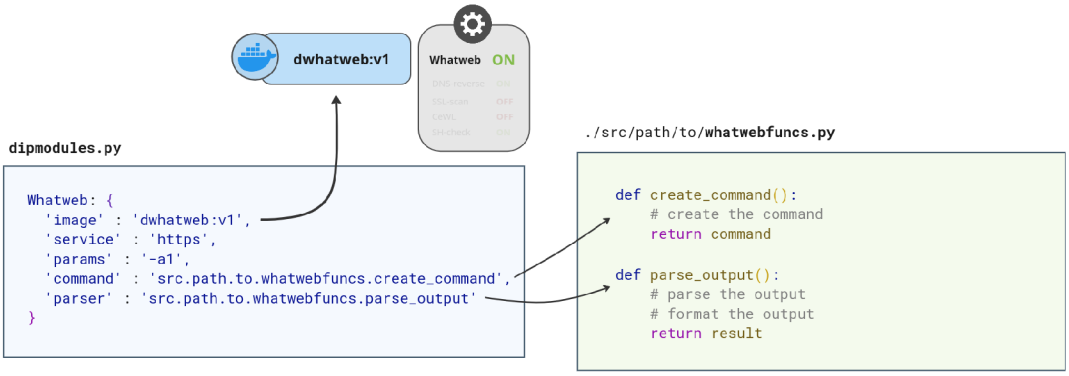


Figure 6.1: Illustration of a module consisting of several parts. There is a record in the python file `dipmodules.py`, a corresponding docker image (which can be shared among several modules), functions for the command creation and output parsing and as the last component, a switch button in the configuration file.

6.3.2 A module record

As shown in Figure 6.1, each record in `dipmodules.py` contains several keys with their corresponding value. Not all of them are required. Minimal keys are those that the framework counts during its run. An error will be raised if any of the minimal keys are missing. The minimal keys are:

- **`'image'`**: A docker image which will run its tool. Note that the entry point of the specified image must be set to run that tool. A special value which 'image' can be set to is None. In that case, no image will be run. It may be useful in some cases when building a module, especially when the functionality of the module depends solely on Python features.
- **`'service'`**: For which service it is designed. If it's desired to run a module on multiple services, creating a separate module for that is necessary. Service cannot be an array. Ex. 'https' or 'domain'
- **`'params'`**: A string specifying the parameters for the tool (ex.: `-p443 -script ssl-cert` for nmap). This is quite benevolent option since the only one accessing this value is again the user in 'command' function and 'additional' function, as described below. This means that the parameter can be hard-coded into those functions instead, leaving an empty string in the record in `dipmodules.py`. However, in terms of good practice, it is recommended not to do so.
- **`'command'`**: Path of a python function that creates the docker run command. This function can be defined in any Python file in any (accessible) folder. The framework will decode the path and run the specified function. Three arguments will be passed and must be accepted:
  1. `type:string` - target address
  2. `type:int` - port
  3. `type:string` - 'params' value from the corresponding module record (which stayed untouched by the framework, as explained in the previous point)

- `return:string` - a command which will be supplied to the docker container for its run, which would be equivalent to

```
docker run imgname <returned_string>
```

- **`'parser'`**: Path of a python function which processes the output of the docker image. The guidelines for specifying this key are the same as above. It takes one argument:

  1. `type: <container>` - raw output from Docker; to get the text representation of the output, the first action must be the following:

```
data = ""
for output line.logs(stream=True):
    data += line.decode("utf-8")
```

  - `return:string` - a string which will be printed out as a final result of the module (including extracting the desired info and formatting).

There are two more keys, which are recognized by the framework, providing the user with flexibility in creating new modules:

- **`'additional'`**: Path of a python function according to rules described above. This function provides a way for the user to perform any actions which wouldn't be otherwise possible due to the strict rules of the framework. If used, the user can define any behaviour here. It takes six arguments and returns nothing:

  1. `type:string` - a command which was created in the function specified in 'command' key
  2. `type:string` - target
  3. `type:port` - port
  4. `type:dict` - the module with all the „key-value“ pairs
  5. `type:function` - for printing to stdout or a file, according to -q and -o options. It takes two parameters, the first of which is the 6th argument (see the line below). The second argument is a text you wish to print as a result of your module.
  6. `type:dict` - first argument for the function (see the line above) (it contains the info if the `-quiet` option was specified and if `-output file` was specified)

- **`'_abort_regular_run'`**: If present, the execution of the docker image won't occur. The value doesn't matter since it is not evaluated.

It is possible to add completely new keys if desired (and use it through the 4th argument of the „additional“ function. It won't cause any conflicts.

A docker image

The technology of docker images provides an isolated environment for each integrated tool, isolating possible incompatibilities and errors. Every Docker image included in the framework is built from a Dockerfile with the following structure:

```
FROM ubuntu

RUN apt update && apt install -y \
    dnsrecon

ENTRYPOINT ["dnsrecon"]

CMD ["--help"]
```

Docker images are based on the ubuntu docker image, which is well maintained and up-to-date, accessible from the official Docker Hub[1]. When creating a user-defined module, Dockerfile doesn't necessarily need to follow described structure. However, it could lead to unexpected errors.

As mentioned above, creating a docker image is not critical for a module, as it can use an existing one or none.

Necessary python functions

To tell the framework how to run the tool and what to do with the output, two functions need to be defined, as mentioned previously. Third, the optional function (specified with `'additional'` key) offers users an option to define custom behaviour.

Entry in the configuration file

The configuration file is based on a structure defined by `configparser` module in Python, and it instructs the framework for which modules should be run. Each entry should contain the exact name of the module and `switched_on` option. The name must match the one specified in the record in `dipmodules.py` file. Otherwise, the framework wouldn't be able to map the module correctly. If `switched_on` option has a value of 1, the corresponding module will be executed. Other options are not taken into consideration as the framework evaluates only `swtiched_on`

```
[NmapSSL]
switched_on = 0

[Sslscan]
switched_on = 1

[Cewl]
switched_on = 1
```

[1] Docker Hub - https://hub.docker.com/_/ubuntu

6.3.3 Modules integrated into the framework

Each module has its corresponding entry in the configuration file. This entry allows the user to tell the framework whether or not the module will be activated. Barring a single exception, every module executes its associated tool via a Docker image, created during the tool's installation phase. The outputs generated by these modules are formatted to provide good readability and intuitiveness. The results of these modules' operations are presented as a part of the testing described in Chapter 7.

- **nmap_S:** Module designed for host discovery. Its function is to scan the current subnet utilizing the `-sn` parameter of the Nmap tool, a scan type also known as a 'no-port-scan'. This module uses ICMP requests to get the status—up or down of a given computer on the network.
- **nmap, masscan:** These two basic modules discover open ports and their operating services. Users can specify any desired ports or ranges they wish the framework includes in scans. Users can select either one or both modules to identify open ports. When both modules are run, the open port lists from both tools are merged, possibly discovering a bigger number of services. Despite the fact that these two modules utilize Nmap and mass can for one specific purpose, other modules can use them with different parameters for different purposes.
- **shcheck_basic:** This module checks the HTTPS protocol using the shcheck tool. This tool examines security headers, which can protect against common web attacks such as XSS, Clickjacking, and code injection attacks. The output includes the list of missing headers along with a brief description of the corresponding risks.
- **whatweb, Whatweb_http:** These modules use the WhatWeb tool to identify technologies used by websites. One is for HTTPS services, and the other is for HTTP. The output includes the target's IP address, origin country, X-Frame-Options, and non-standard headers, among other information.
- **Dnsrecon, Dnsrecon_reverse:** This module is for the „domain“ service. It uses DNSRecon, a tool that performs various DNS queries such as standard record queries, zone transfers, reverse lookups, and more. It shows TXT records, which might contain interesting information, and PTR records from reverse lookups.
- **cewl, cewl_http:** This module creates custom word lists based on a webpage's content using the CeWL tool. It uses a custom key in the `dipmodules.py` record for specifying a path to the file where the list should be saved.
- **nmapSSL:** This module uses the nmap tool with the ssl-cert script to gather information about a target's SSL certificate.
- **sslscan:** This module uses the sslscan tool to analyze a target's SSL/TLS setup. The output is a list of enabled TLS/SSL versions and used cypher suites sorted by security level. It also displays information about possible threats like TLS Renegotiation and the Heartbleed vulnerability.
- **ftpAnonLogin:** The only module not using any Docker image and is entirely based on Python, therefore using `'additional'` key for specifying the extra function. It uses `ftplib` library to try and connect to the FTP service and perform an anonymous login. If the login succeeds, the user is notified.

6.4 Challenges

The following section will highlight some of the challenges associated with software development.

Dealing with parsing

Output parsing was notably significant among the challenges encountered during the development of this tool. Creating a universal parser for the integrated tools was impractical due to the diversity of the outputs. Even a change in the parameters of the same tool could, in some cases, alter the format or content of the output. Moreover, such a universal parser would conflict with the tool's principle of simple extensibility. For these reasons, it was determined that each module must include a parser for the specific tool or, more precisely, for a particular combination of the tool's parameters. If one parser would be suitable for multiple modules, reusing it multiple times would be no issue.

Offering modularity

Another challenge was to provide the user with extensibility in its broadest, simplest, and cleanest form. As previously mentioned, the cybersecurity and penetration testing field evolves rapidly, with security vulnerabilities appearing and disappearing daily and the tools used in this sphere following a similar trajectory. Therefore, every user will likely have slightly different requirements and needs related to penentration-testing tools. Support for extensibility is thus one of the fundamental features of this tool. The goal was to facilitate the use of user-defined modules, the creation of which would require the user to expend only the minimal necessary effort while also offering the broadest possible options. The solution was a combination of several elements. The first is constructing the tool's architecture so that the core of the framework (a file of Python functions) manages the coordination and interconnection of modules. Integrated modules are written in the same format a user would define their new ones. The second is including the 'additional' key in the module description, the value of which is a path pointing to a user-defined function.

Chapter 7

Testing

The following text describes the process and evaluation of the functional verification of the framework. The virtual machines used will be discussed, along with the chosen approach, output images, and finally, a review of how closely the framework aligned with the design.

7.1 Approach

As mentioned in Chapter 3.1.3, in the case of most of the implemented modules, testing public applications is, for several reasons, not possible. For testing purposes, the technology of virtual machines was chosen, specifically two virtualization software: VMware by VMware, Inc. and VirtualBox by Oracle Corporation. Both create a virtual environment, allowing the operating system to operate as if running directly from the hardware while running on a software-emulated virtual machine. Thanks to this, it provides isolation, thus making it suitable for testing the framework.

This is also suitable for another type of testing, where the installation process's functionality and the tool's operation in a new environment will be evaluated.

Four publicly available virtual machines were selected from the `vulnhub.com`[1] website. The first one is the bWAPP project[2], which provides a web application with several vulnerabilities. The project used to be under active development, with the last version being v2.2 in 2014. The second chosen is the Broken Web Applications Project developed by OWASP[3]. It also provides several versions, and for the purpose of this work, version 1.2 was chosen. The last two virtual machines are ZorZ[4] and RPIMER[5]. For clarity in the following text, these tools will be referred to as „bWAPP VM“, „BWA VM“, „ZozR VM“ and „RPIMER VM“.

The tests were also conducted among other individuals in the IT field. Specifically, it involved a group of 5 people using Linux and MacOS operating systems, who were provided with this tool to verify its functionality.

[1] vulnhub.com
[2] bWAPP - https://www.vulnhub.com/entry/bwapp-bee-box-v16,53/
[3] https://www.vulnhub.com/entry/owasp-broken-web-applications-project-12,46/
[4] ZorZ - https://www.vulnhub.com/entry/tophatsec-zorz,117/
[5] RPIMER - https://www.vulnhub.com/entry/primer-101,136/

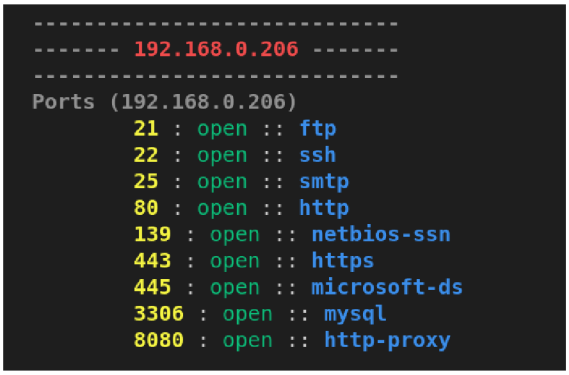


Figure 7.1: Resut of a combination of modules `masscan` and `Nmap_S`, providing information about open ports on bWAPP VM.

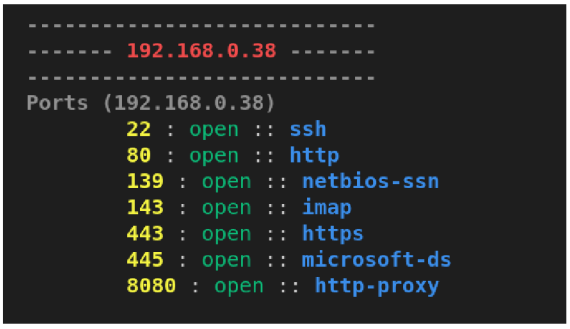


Figure 7.2: Output of a single module `Nmap_S`, showing open ports and running services on BWA VM.

7.2 Process

The tool was executed against the selected virtual machines and the outputs are presented below.

The port scanning can be done using either nmap, masscan or both. Using both tools increases the chances of discovering a greater number of ports. However, it takes longer. The results can be seen in Figure 7.1 for bWAPP VM and Figure 7.2 for BWA VM.

Execution of `sslscan` module is shown in Figures 7.3 and 7.4

`Whatweb` module outputs information about the web itself, and the example can be seen in Figures 7.5 and 7.6. The second mentioned is the output of scanning Zorz VM.

The output upon discovering an anonymous login vulnerability of the FTP service is illustrated in Figure 7.7

Next, the option of using the `DISC` scan was tested, which obtains information about targets marked as „up“ by sending ICMP requests. This is accomplished using the „Nmap_s“ module. However, the DISC mode was supplemented with the -c1 option, which instructs the framework to run all enabled modules against each of the discovered targets. The complete output can be seen in Figure 7.8. When the framework encounters an issue, it correctly informs the user and immediately continues with the next module instead of stopping its progress. For better orientation, the list of online virtual machines, along with the corresponding IP addresses, is for this test as follows:

- **bWAPP VM** - 192.168.0.206
- **BWA VM** - 192.168.0.38
- **ZorZ VM** - 192.168.0.136
- **RPIMER VM** - 192.168.0.101

Figure 7.3: The output of the SSL scan module runs against bWAPP VM, which provides a clear division of TLS/SSL protocols and supported cypher suites.

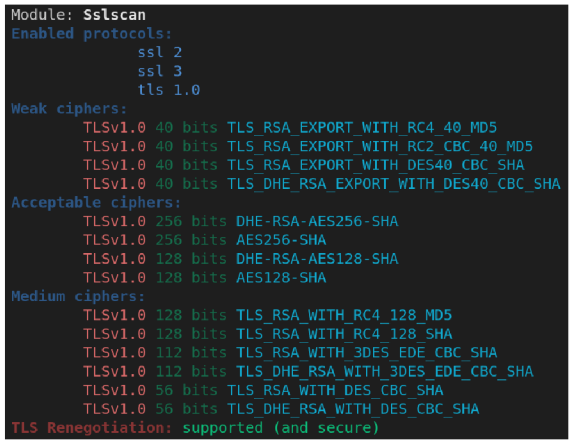


Figure 7.4: Sslscan module run against BWA VM, showing many supported cyphers, among which there are also weak cyphers.

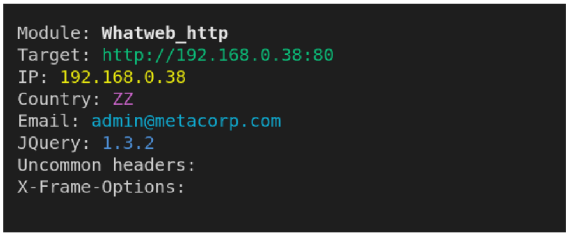


Figure 7.5: Outputu of `Whatweb_http` module provided after scanning bWAPP VM.

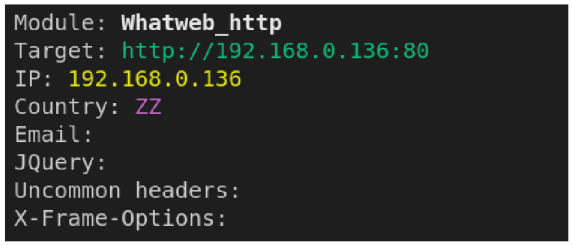


Figure 7.6: Result of `Whatweb_http` module from scanning Zorz VM.

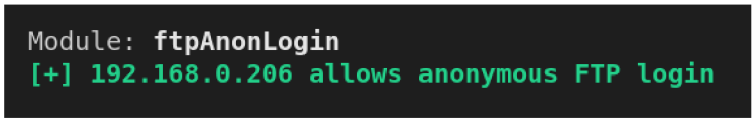


Figure 7.7: Alerting the user to the presence of anonymous FTP login vulnerability.

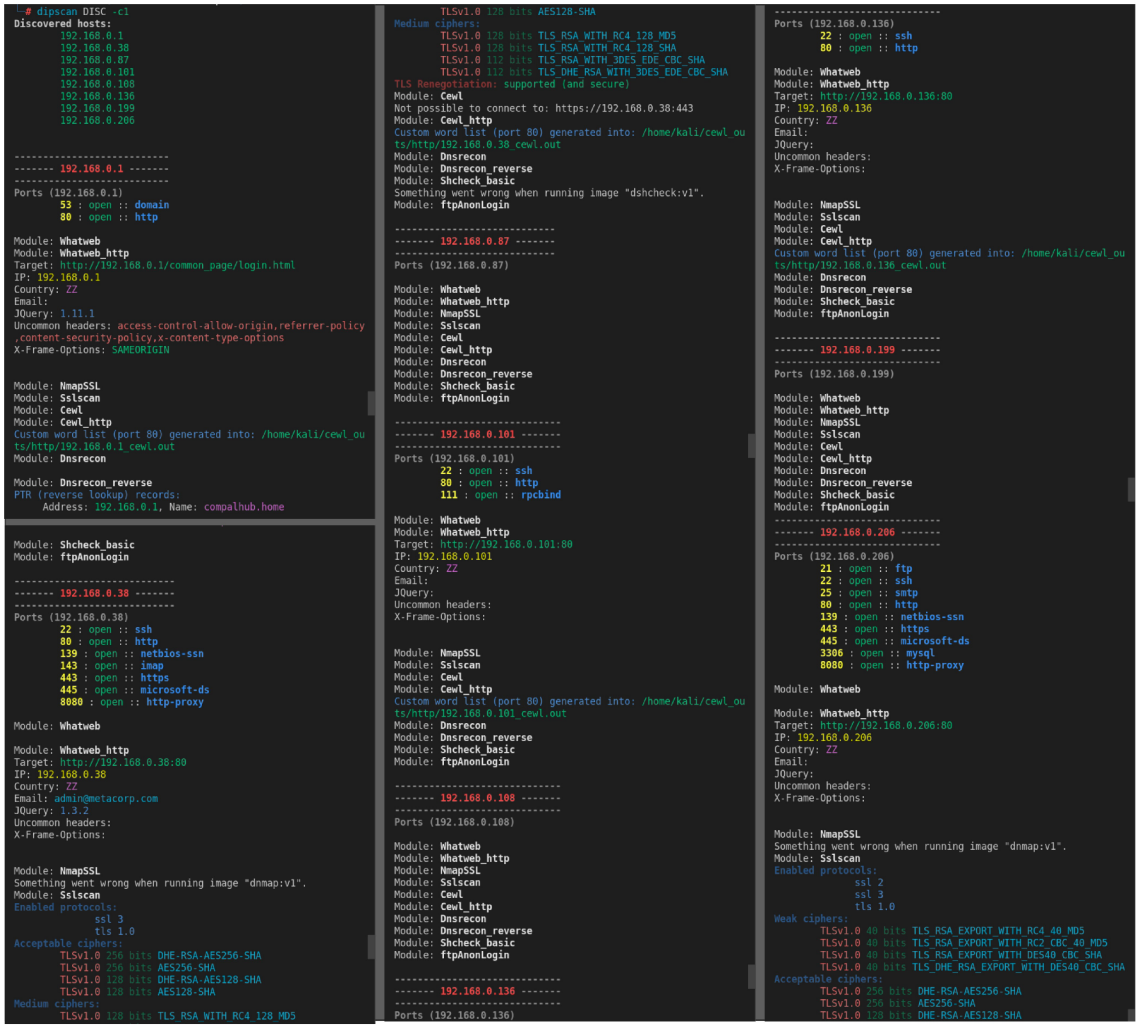


Figure 7.8: Ouput of `dispcan DISC -c1`, which first discovers all the visible hosts (`DISC`) and subsequently runs all the enabled modules against each of them (`-c1`. Four virtual machines were running among other IPs (as the gateway or mobile phone). Due to space reasons, the last few lines were not included in the image.

7.2.1 Adding a user-defined module

To verify the functionality of adding user-defined modules, the feature of retrieving the content of `robots.txt`[6] was chosen. The task is to add a module without manipulating the framework's source code.

Before starting, it is necessary to determine the location of the configuration file and the mapping file for the modules since this location can be different for each user. For this purpose, `CONF` mode will be used, which task is to print the location of the module file and configuration file with the user's system:

```
$ sudo dipscan CONF

 The two files with the description and settings of the module are the following:

 Details on modules:  /root/.config/dipconf/dipmodules.py
 Turn on/off modules: /root/.config/dipconf/run.cfg
```

First, a new record was added to `dipmodules.py`:

```
'robots': {
    'image': None,
    'service': 'HTTP',
    'params': '',
    'additional': '/home/kali/new mod/func.robots_additional',
    'command': '/home/kali/new mod/func.cmd',
    'parser': '/home/kali/new mod/func.parse',
}
```

The image is set to `None` since the module will operate using Python. It is designed for HTTP protocol, and no parameters are needed. As this module will not be using any Docker image, it is not necessary to construct a command for Docker or a parser for its output. However, based on the conditions implied in the tool's description, it is necessary to have these paths defined. Therefore, the only functional key remaining is „additional“, which points to a function that will handle everything the module is supposed to provide. As stated before, the function under the key „additional“ must always take six arguments, some of which will not be utilized in this case.

Therefore, the file `/home/kali/newmod/func.py` contains three functions, two of which (`cmd()` and `parse()`) return an empty string, since their result is not used. All the functionality is covered by `robots_additional()`. Thanks to `requests` module in Python, it tries to connect to the destination and obtain `robots.txt` file. In case of success, the content is printed out, while in case of the absence of the text file, a corresponding message is provided. The core of the function is shown below:

```
import requests

###

url = "http://" + target + "/robots.txt"
```

[6]The „robots.txt“ file is a standard used by websites to communicate with web crawlers and other web robots, providing instructions about which areas of the website should not be processed or scanned.[52]

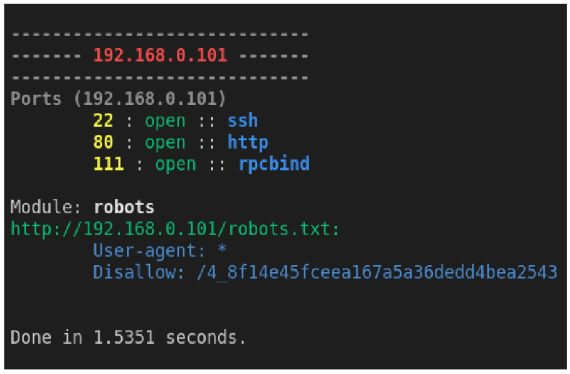


Figure 7.9: The output of the user-defined module for detecting `robots.txt` file. In this case, the file was found on PRIMER VM and its content was printed on standard output.

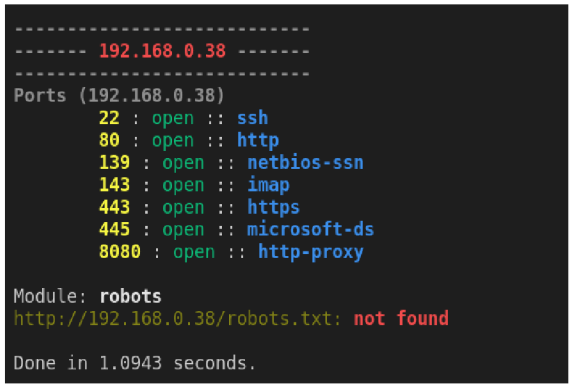


Figure 7.10: Warning message indicating that the user-defined module for detecting `robots.txt` files did not find it at the target machine, which is BWA VM.

```
response = requests.get(URL)
result = ''
if response.status_code == 200:
    # print robots.txt
else:
    # print the message
```

An example of the output of both cases (found and not found) is shown in Figures 7.9 and 7.10.

The functionality of this module is intentionally kept simple to provide a clear and concise illustration of how to create such a module. The primary aim is to demonstrate the process of adding user-defined modules rather than significantly expanding the framework's functionality.

7.3 Results

The tests fulfilled the functional requirements stated in chapter 5 and successfully satisfied the use cases. The extensibility was also verified, both on users' computers and in the virtual environment of a freshly installed Linux distribution. Testing of other individuals showed that, upon meeting dependency requirements, the tool is operational on both Linux and MacOS. In the case of all five people involved in the testing, all modules were successfully launched. Of these, four people also verified the extensibility by creating a new user module, which was, in all 4 cases, successfully incorporated into the framework's functionality.

Overall, the framework fulfils the specified functionality.

Chapter 8

Conclusion

This work focused on cybersecurity with an emphasis on web server security. Initially, methods of securing web servers were described along with common vulnerabilities, followed by an explanation of ways to detect these security flaws. It was accomplished using OWASP methodology and various penetration testing techniques. In the practical part, a tool for automated testing of web servers was developed, which combines the functionality of modules and offers the ability to add user-defined modules. Its functionality was verified in a simulated environment. Possible extensions include expanding functionality through additional modules, adding the option to follow redirections, or adding a proxy feature to provide users with further valuable information.

Bibliography

[1] *CWE-35: Path Traversal* [https://cwe.mitre.org/data/definitions/35.html]. Accessed: April 30, 2023. Page Last Updated: April 27, 2023.

[2] *OWASP Top Ten Project* [https://owasp.org/www-project-top-ten/].

[3] *OWASP TOP10* [https://www.owasptopten.org/].

[4] *Web Penetration Testing with Kali Linux - Third Edition*. 3rd editionth ed. Packt Publishing, 2018. ISBN 1-78862-337-1.

[5] *CWE-778: Insufficient Logging* [https://cwe.mitre.org/data/definitions/778.html]. 2023. Accessed: May 2, 2023.

[6] *How to scan the Internet in 5 minutes* [https://thechief.io/c/editorial/how-to-scan-the-internet-in-5-minutes/]. TheChief, 2023.

[7] *Masscan: TCP port scanner, spews SYN packets asynchronously, scanning entire Internet in under 5 minutes | Hacker News* [https://news.ycombinator.com/item?id=28682986]. 2023.

[8] Alencar, M. S. *Cryptography and network security*. 1st ed.th ed. New York, New York: River Publishers, 2022. River Publishers Series in Security and Digital Forensics Ser. ISBN 9781000792935.

[9] Barker, J. *Confident Cyber Security : How to Get Started in Cyber Security and Futureproof Your Career*. London ;: Kogan Page, Limited, 2020. Confident series. ISBN 1789663415.

[10] Barnett, R. and Grossman, J. *Web Application Defender's Cookbook: Battling Hackers and Protecting Users*. Wiley, 2013. ISBN 9781118417058. Available at: https://books.google.cz/books?id=flC9dFFLWIsC.

[11] Blokdyk, G. *CVSS: A Complete Guide - 2021 Edition*. Charleston, South Carolina: 5STARCooks, 2021. ISBN 978-1867384744.

[12] Buchanan, C., Ip, T., Mabbitt, A., May, B. and Mound, D. *Python Web Penetration Testing Cookbook*. Packt Publishing, 2015. ISBN 1784392936.

[13] Burr, W. E., Dodson, D. F. and Newton, E. M. *Digital Identity Guidelines: NIST Special Publication 800-63B*. National Institute of Standards and Technology, 2017.

[14] Corporation, M. *CWE-1104: Use of Unmaintained Third Party Components* [https://cwe.mitre.org/data/definitions/1104.html]. 2021. [Online; accessed 1-May-2023].

[15] Corporation, M. *CWE-918: Server-Side Request Forgery (SSRF)* [https://cwe.mitre.org/data/definitions/918.html]. 2021. [Online; accessed 2-May-2023].

[16] Corporation, M. *CWE-16: Configuration* [https://cwe.mitre.org/data/definitions/16.html]. 2023. [Accessed: May 1, 2023].

[17] Corporation, M. *CWE-611: Improper Restriction of XML External Entity Reference ('XXE')* [https://cwe.mitre.org/data/definitions/611.html]. 2023. [Accessed: May 1, 2023].

[18] Corporation, M. *CWE-287: Improper Authentication* [https://cwe.mitre.org/data/definitions/287.html]. 2023 (last updated). [Online; accessed 01-May-2023].

[19] Corporation, T. M. *CWE-209: Generation of Error Message Containing Sensitive Information* [https://cwe.mitre.org/data/definitions/209.html]. 2010. [Online; accessed 1-May-2023].

[20] Corporation, T. M. *CWE-522: Insufficiently Protected Credentials* [https://cwe.mitre.org/data/definitions/522.html]. 2010. [Online; accessed 1-May-2023].

[21] Erickson, J. *Hacking: The Art of Exploitation, 2nd Edition*. No Starch Press, 2008. No Starch Press Series. ISBN 9781593271442. Available at: https://books.google.cz/books?id=0FW3DMNhl1EC.

[22] Fall, K. and Stevens, W. *TCP/IP Illustrated, Volume 1: The Protocols*. Pearson Education, 2011. Addison-Wesley Professional Computing Series. ISBN 9780132808187. Available at: https://books.google.cz/books?id=a23OAn5i8R0C.

[23] Forsgren, N., Humble, J. and Kim, G. *Accelerate: The Science of Lean Software and DevOps Building and Scaling High Performing Technology Organizations*. 1stth ed. IT Revolution Press, 2018. ISBN 1942788339.

[24] Graham, R. D. *Masscan: TCP port scanner, spews SYN packets asynchronously, scanning entire Internet in under 5 minutes* [https://github.com/robertdavidgraham/masscan]. GitHub, 2023.

[25] Grossman, J. *CROSS-SITE TRACING (XST) THE NEW TECHNIQUES AND EMERGING THREATS TO BYPASS CURRENT WEB SECURITY MEASURES USING TRACE AND XSS* [online]. Santa Clara, CA, USA: www.whitehatsec.com, 2003 [cit. 2023-01-03]. Available at: https://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf.

[26] Harold, E. R. *XML in a Nutshell*. 2nd ed. O'Reilly Media, Inc., 2002. ISBN 0596002920.

[27] Hoffman, A. *Web Application Security: Exploitation and Countermeasures for Modern Web Applications*. O'Reilly Media, Inc., 2020. ISBN 978-1-492-05756-9.

[28] Jorgensen, P. *Software Testing: A Craftsman's Approach, Fourth Edition*. Taylor & Francis, 2013. An Auerbach book. ISBN 9781466560680. Available at: https://books.google.cz/books?id=6WlmAQAAQBAJ.

[29] Kennedy, D., O'Gorman, J., Kearns, D. and Aharoni, M. *Metasploit: The Penetration Tester's Guide*. No Starch Press, 2011. ISBN 9781593274023.

[30] Klein, A. DOM Based Cross Site Scripting or XSS of the Third Kind. *Annals of Mathematics*. 2005. Available at: http://www.webappsec.org/projects/articles/071105.shtml#r5.

[31] Kurose, J. F. *Computer networking : a top-down approach*. 6th ed., International.th ed. Boston ; London: Pearson, 2013. ISBN 978-0-273-76896-8.

[32] Liu, C. and Albitz, P. *DNS and BIND*. O'Reilly Media, Inc., 2006. ISBN 978-0-596-10057-5.

[33] Lyon, G. Nmap Reference Guide. In: Fyodor, ed. *Nmap Network Scanning: The Official Nmap Project Guide*. 3rd Editionth ed. Insecure.Com LLC, 2021.

[34] Lyon, G. *Nmap: Free Security Scanner, Port Scanner, Network Exploration Tool* [https://nmap.org/]. Accessed: 2023.

[35] Marsh, N. *Nmap Cookbook: The Fat-free Guide to Network Scanning*. CreateSpace Independent Publishing Platform, 2010. YBP ORDER. ISBN 9781449902520. Available at: https://books.google.cz/books?id=U4H3QwAACAAJ.

[36] McNab, C. *Network Security Assessment: Know Your Network*. O'Reilly Media, 2004. Know your network. ISBN 9780596006112. Available at: https://books.google.cz/books?id=JNSbAgAAQBAJ.

[37] MITRE. *CWE-200: Exposure of Sensitive Information to an Unauthorized Actor* [https://cwe.mitre.org/data/definitions/200.html]. 2021. [Online; accessed 30-April-2023].

[38] MITRE Corporation. *CWE-259: Use of Hard-coded Password* [https://cwe.mitre.org/data/definitions/259.html]. 2011. [Online; accessed April 30, 2023].

[39] MITRE Corporation. *CWE-327: Broken or Risky Crypto Algorithm* [https://cwe.mitre.org/data/definitions/327.html]. 2011. [Online; accessed April 30, 2023].

[40] Nagarjun, P. and Ahamad, S. S. Cross-site Scripting Research: A Review. *International Journal of Advanced Computer Science and Applications*. The Science and Information Organization. 2020, vol. 11, no. 4. DOI: 10.14569/IJACSA.2020.0110481. Available at: http://dx.doi.org/10.14569/IJACSA.2020.0110481.

[41] Ortega, J. M. *Mastering Python for Networking and Security.* September 2018. ISBN 9781788992510.

[42] Paar, C. and Pelzl, J. *Understanding Cryptography: A Textbook for Students and Practitioners.* Springer Berlin Heidelberg, 2011. ISBN 9783642041006.

[43] Paul Troncone, C. A. *Cybersecurity Ops with Bash : Attack Defend and Analyze from the Command Line.* Sebastopol, CA,: O'Reilly Media, 2019. ISBN 1788839234.

[44] Prasad, P. *Mastering Modern Web Penetration Testing.* Packt Publishing, 2016. ISBN 9781785284588.

[45] Scambray, J., Liu, V. and Sima, C. *Hacking Exposed Web Applications, Third Edition.* McGraw Hill LLC, 2010. ISBN 9780071740425. Available at: https://books.google.cz/books?id=tleXSTnyCXcC.

[46] Schneier, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, 2015. ISBN 9781119183471.

[47] Seitz, J. *Black Hat Python: Python Programming for Hackers and Pentesters.* No Starch Press, 2014. ISBN 9781593275907.

[48] Singh Chauhan, A. and Singh., A. *Practical Network Scanning: Capture Network Vulnerabilities Using Standard Tools Such As Nmap and Nessus.* Birmingham: Packt Publishing, Limited, 2018. ISBN 1788839234.

[49] Stallings, W. *Network Security Essentials: Applications and Standards.* Prentice Hall, 2007. William Stallings books on computer and data communications technology. ISBN 9780132380331.

[50] Stallings, W. *Cryptography and Network Security: Principles and Practice, 5/e.* Pearson Education, 2011. ISBN 9788131761663.

[51] Stein, L. D. *Web Security: A Step-by-Step Reference Guide.* Addison-Wesley Professional, 2021.

[52] Stuttard, D., Pinto, M. and Safari, a. O. M. C. *The Web Application Hacker's Handbook, 2nd Edition.* Wiley, 2011. Available at: https://books.google.sk/books?id=COhOzQEACAAJ.

[53] Sullivan, B. and Liu, V. *Web Application Security: A Beginner's Guide.* McGraw-Hill Osborne Media, 2011.

[54] The Open Web Application Security Project. *OWASP Testing Guide v4.0.* The Open Web Application Security Project, 2014.

[55] Weidman, G. *Penetration Testing: A Hands-On Introduction to Hacking.* No Starch Press, 2014. ISBN 9781593275648. Available at: https://books.google.cz/books?id=T_LlAwAAQBAJ.

[56] Wright, J. and Cache, J. *Hacking Exposed Wireless, Third Edition: Wireless Security Secrets & Solutions.* McGraw Hill LLC, 2015. Hacking Exposed. ISBN 9780071827621.

[57] Young, S. and AITEL, D. *The Hacker's Handbook: The Strategy Behind Breaking Into and Defending Networks*. CRC PRESS COMPANY, 2004. ISBN 9780849308888.

Appendix A

Content of the attached storage media

In this section, the contents of the attached storage media are described. Its directory structure with a depth of 2 is as follows:

```
"/"
 |--- /imagesFromDocker
 |      |--- /cewl
 |      |--- /dnsrecon
 |      |--- /gobuster
 |      |--- /masscan
 |      |--- /nmap
 |      |--- /shcheck
 |      |--- /sslscan
 |      |--- /whatweb
 |--- __main__.py
 |--- README.md
 |--- requirements.txt
 |--- setup.py
 |--- /src
 |      |--- argParser.py
 |      |--- classes.py
 |      |--- /cores
 |      |--- dckrChiefExecutive.py
 |      |--- __init__.py
 |      |--- /parsers
 |      |--- portFuncs.py
 |      |--- p.py
 |      |--- scanCoordination.py
 |      |--- /secondary
```

README.md

The README.md file contains instructions for the installation of the dipscan tool, including dependencies. Furthermore, it provides a description of the individual structures of the architecture and a guide to creating your own module. Finally, it mentions some common errors that may occur when working with the tool.

/imagesFromDocker

This folder contains Dockerfile files for creating Docker images. During the installation of the tool, all of these are processed and installed. The format of the individual Dockerfiles is very similar, usually differing only in which tool the respective Docker image covers.

src/cores

The src/cores folder contains Python files with functions necessary for the operation of the modules. These are the functions that the pointers in the module descriptions ('core', 'command') point to.

src/parsers

This folder contains Python files with functions for parsing and formatting the outputs of Docker images. These are pointed to by the 'parser' field in the module description. It would also be possible to eliminate the src/parser file and insert the functions into the file containing the functions mentioned in the previous point, but for clarity purposes, it is split. When defining a user module, all functions can be located in one file.

src/secondary

The src/secondary folder contains auxiliary functions for the correct operation of the framework as well as default configuration files, which are, during the installation process, copied to the default location for configuration files of the given system.