Information Security
P3 Network-Security Challenge v1.0

Winter term 2022/2023

Introduction

The most widespread use of networking today is when connecting to the internet to browse the worldwide web. The web browser you are using is connecting to web servers using the HTTP (Hypertext Transfer Protocol). When developing web applications, most of the effort is usually spent fulfilling the basic functionality requirements. Often, security is left as an afterthought. However, since the majority of web services are handling actual user data (be it usernames, passwords, addresses or payment information, just to name a few), attackers often have valuable targets.

Since the beginning of the worldwide-web in the 90's, we have seen numerous attacks, and even (and maybe especially) in the last few years, large data breaches are a constant part of the news-cycle. Having a look at the OWASP Top 10[1], an attempt to categorize the main vulnerabilities in web applications, we see that the number two vulnerability in their list is called **A02:2021 – Cryptographic Failures**, tying us into our first assignment where we had a look at using cryptography in an incorrect fashion. We also see **A06:2021 – Vulnerable and Outdated Components**, bridging the gap to the second assignment, as this is often concerning components such as web servers themselves, most of which are written in C/C++. However, this category also includes different parts of the ecosystem such as Javascript libraries or Wordpress plugins.

The goal of this task is to get familiar with some of the main types of vulnerabilities and attacks that are part of the OWASP Top 10. We have prepared small hacklets in several different categories. These hacklets are small self-contained web-apps built using `Flask`[2], a popular `python` web-framework.

There are 11 subtasks for this assignment. The number of points awarded for each task is stated in this document. If you have any questions, please contact the responsible teaching assistants: `amel.smajic@student.tugraz.at, b.jost@student.tugraz.at`, or ask a question in the IAIK Discord[3]: `#infosec`.

Framework. All of the subtasks use a similar framework, with exceptions noted in the respective subtasks. A subtask lives in it's own folder, usually containing the following files and folders:

- `attack_script`: This is the folder where your attack script (usually called `attack.py`) is located. This is the only part that you actually should change to solve the task. All other parts of the challenge will be restored to their upstream version on the test system. For some tasks, you also have a victim (in a file called `victim.py`), that represents the behavior of another user visiting the webpage. The behavior of this victim will be explained in more detail in the respective subtasks.
- `db`: This is a directory representing the database that is used by the web application. Like in a real-world deployment, you cannot directly interface with this database (it is not exposed to the public) and must use the web application's features to interact with it.
- `www`: This is the main directory for the web application. Here you will usually find a `main.py` file, containing the main `Flask` web application. It also may contain additional files like static HTML files, HTML templates, Javascript or CSS files.

[1] https://owasp.org/www-project-top-ten/
[2] https://flask.palletsprojects.com/
[3] `https://discord.com/invite/bBbrVSJ`

The challenges are written and have to be solved in Python (3.8+). The following packages must be installed, use, e.g., `pip install <package_name> --user`

- requests
- pycryptodomex
- selenium
  - This might also need `apt install chromium-chromedriver` to install the Chromium Web-Driver for it.

You can use these during your exploits.

docker-compose. We have set up the web application as a composition of Docker containers. The provided `docker-compose.yml` file is orchestrating their interaction. To build and run the web application use the provided `Makefile`, which uses the appropriate `docker-compose` commands. You might have to install docker-compose first (`apt install docker-compose`).

Testsystem Setup. On the test-system, you will not be allowed access to the files in `db` or `www`, in fact, they wont even exist. You are only allowed to interact with the web service over the network, using the provided url.

Successful Exploit. For a successful exploit, recover the information specified in the challenge description and either write it to the standard output of your script or to a file called `solution.txt`. For all exploits, you get a maximum of 10 seconds on the test system. If a victim is part of the exploit, the victim will begin its operation after 1 second has passed, and will continuously repeat his action until the timeout is reached.

1 SQL Injection

The following three tasks will focus on databases and how a text field can be used to inject malicious code in order to either gain access to a service by bypassing the login, or leaking data from the database. SQL injection attacks are a type of injection attack (**A03:2021 – Injection**), in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.[4]
Should those SQL statements combined with the input from the user not be parsed correctly, the user is able to inject any arbitrary SQL code into the statement, thus manipulating the outcome to his advantage. The simplest and most commonly used example is the following:

```
1 statement = "SELECT * FROM users WHERE name = '" + userName + "'"
```

The variable userName would contain the input from a text field, where the username is expected to be inserted. The userName however can be manipulated in any way, such that username could also be "' OR '1'='1' –". The two dashes at the end work as a comment thus invalidating any other command being defined by the programmer. Our query for the server would then look the following:

```
1 statement = "SELECT * FROM users WHERE name = '' OR '1'='1' -- '"
```

As this code always returns true in the evaluation, any authentication would be allowed as the user - seemingly - entered a correct password.

1.1 Basic SQL Injection (1 Points)

Folder: `SQLi1`
Webservice URL: `http://localhost:8080/`

Challenge. "I forgot my password, can you log me in?" - admin
The user can insert a username and a password and register as user to the service. Afterwards the user is allowed to login. Can you login as admin?

Hints. Do not try and brute-force the password. Look at the example above for a simple sql injection, maybe it is a good start?

1.2 Intermediate SQL Injection (1.5 Point)

Folder: `SQLi2`
Webservice URL: `http://localhost:8081/`

Challenge. "My database can not be leaked" - admin
There are 2 flags hidden in this challenge, make sure to find both of them! (Print both of them on separate lines in on stdout and/or solution.txt)
We fixed the simple error from last time! But there might still be ways to leak information from the database.

Hints.

- `UNION SELECT` might be a good starting point to look at.
- Due to some technicalities, one of the flags here has a different format than usual flags. We therefore provide you with a second regex. You can just print/write this flag as you recovered it, you do not need to "fix" the format of the flag.

[4] `https://owasp.org/www-community/attacks/SQL_Injection`

1.3 Advanced SQL Injection (1.5 Points)

Folder: `SQLi3`
Webservice URL: `http://localhost:8082/`

Challenge. "My files can not be leaked!" - admin
Can you get the flag? (/flag/flag.txt)
Users can now also upload files. Could it be possible to load the flag into the database to view it?

Hints. Sometimes SQL dialects also have methods to interact with more things than just normal tables.

2 Files and Serialization

2.1 Untrusted Data (1.5 Point)

Folder: `File1`
Webservice URL: `http://localhost:8083/`

The OWASP category **A08:2021 – Software and Data Integrity Failures** contains a number of different sub-categories, amongst them the *Insecure Deserialization of Data*, which covers the fact that sometimes web applications hand out data to users and then at a later point read that data back in. For portability, this data is often serialized into a standardized format. However, many serialization formats are much more powerful than developers might assume and can not just serialize data...

Challenge. Users can upload their own content in the form of an .svg file. The developers thought that introducing a new custom field in the image format would provide more security than harm. Try to find the flaw in the upload function which handles the images and get a hold of the contents of flag.txt. Print the flag when you have found it!

2.2 Path Traversal (1 Point)

Folder: `File2`
Webservice URL: `http://localhost:8084/`

The OWASP category **A03:2021 – Injection**, in addition to SQL injection and Cross-side scripting, also covers the much more innocent looking *External Control of File Name or Path*. This is a set of vulnerabilities based around the idea that users in control of choosing paths or filenames of objects stored on a web server. While looking innocent at first, this simple vulnerability can lead to a whole host of problems, ranging from data breaches to remote code execution.

Challenge. You are given access to the new admin panel of your company where you can edit or delete existing user data. Additionally there is the option to download the data as a csv file. The developers said that there are still some difficulties with implementing this feature securely. Is there a possibility to download more files than intended?

Hints. Look at the file structure created by the service and the means of accessing a file.

3 Cross-site Scripting

The next two tasks are about cross-site scripting, which is a different kind of injection attack (**A03:2021 – Injection**). Cross-site scripting enables an attacker to inject code into a website which will then be executed by a different user. This can allow the attacker to, for example, get a session-cookie of another user with which the attacker can impersonate the victim as long as the session-cookie is valid or force the user to take actions without his intent.

3.1 Basic Cross-site Scripting (1 Point)

Folder: `XSS1`
Webservice URL: `http://localhost:8085/`

Challenge. Messages can be posted to the website by a user. Every message that is posted can be seen by each user after logging in. Try to get the flag of the victim.

Victim. The victim is a normal user that makes an account, logs into the website and then visits the main page after logging in. The flag is stored in his session cookies. For local testing just start the victim directly after your exploit. On the testsystem, the victim will start performing his actions 1 second after your attack script has been started.

Hints.

- Take a look at `XMLHttpRequest`s for the attack.
- In your code, you might want to wait a bit for the victim to do its thing (for example using, `time.sleep(3)`).

3.2 Advanced Cross-site Scripting (2 Points)

Folder: `XSS2`
Webservice URL: `http://localhost:8086/`

Challenge. For this challenge messages can be sent to the admin. Only admins have access to the flag, can you still get a hold of it?

Victim. The victim is an admin user that, logs into the website and then visits the main page after logging in. The flag is not directly known by the admin. For local testing just start the victim directly after your exploit. On the testsystem, the victim will start performing his actions 1 second after your attack script has been started.

Hints.

- Take a look at `XMLHttpRequest`s for the attack.
- In your code, you might want to wait a bit for the victim to do its thing (for example using, `time.sleep(3)`).
- While the victim does not have direct knowledge of the flag, can you still get him to help you out?

4 Access Control and User Authentication

A01:2021 – Broken Access Control is the OWASP category with the most detected vulnerabilities. Some of the most common vulnerabilities in this category include missing access controls for APIs, privilege escalation, parameter tampering in URLs, violation of the principle of least privilege and many more. These vulnerabilities arise most often due to the complexity of large APIs, where care has to be taken over the whole footprint of the API. Even if most calls to an API have been crafted with care, missing out on correct access control for a single route can already spell disaster.

4.1 Basic API (1 Point)

Folder: `API1`
Webservice URL: `http://localhost:8087/`

Challenge. The flag is only allowed to be read by administrator accounts. Try to use the API to get access to it even though you are not an admin.

Hints. Have a look at all of the routes that the API exposes and what features they provide.

4.2 Advanded API (2 Point)

Folder: `API2`
Webservice URL: `http://localhost:8088/`

Challenge. Again the flag is only allowed to be read by administrator accounts. Although this time the API looks a bit different. Users have the ability to post messages to a message board. Additionally, posts that are made by accounts that aren't older than 1 day are flagged with a *Yes* in the *Beginner* column. Lastly, the website is already filled with messages of potential victims.

Hints. For this challenge you will have to restart the web server each time you attempt an attack. It seems Broken Access Control isn't the only problem this website has. Take a look at the generation of the session ID. Keep in mind, on each startup of the web server, all users and messages are deleted and generated anew.

5 Authentication

5.1 JSON Web Tokens (1.5 Points)

Folder: `JWT`
Webservice URL: `http://localhost:8089/`

JWT (JSON Web Token) is an open standard[5] to share security-related information. Typically, a JWT is an encoded (base64) JSON object including a set of claims related to a user (e.g. email address, a flag indicating if the user is an admin etc.). JWTs are signed using a cryptographic algorithm to ensure that the claims cannot be altered after the token is issued. However, it is easy to forget checks in which enable attacks like forging signatures or bypassing missing checks.

Challenge. In this task, we provide a flask application using JWTs. You can find the implementation of the library in `custom_jwt_lib/myjwt.py`. Can you find a flaw and promote yourself to an admin and read the flag?

Hint. Take a close look at potential flaws the library implementation.

[5] `https://jwt.io/`

6 Template Injection

OWASP category A03:2021 – Injection. In addition to well known injections such as SQL-injection and Cross-side-scripting there also exists a set of vulnerabilities based around templates. Templating engines combine templates and a data model to form a result document. In case of web development this output document is usually a html page with dynamic content. Problems may occur when developers are not careful with the data given to the rendering engine. Malicious data can lead all the way up to remote code execution and a compromised system.

6.1 SSTI (2 Points)

Folder: `SSTI`
Webservice URL: `http://localhost:8090/`

Challenge. Url parameters on websites can be dangerous if they are not handled in a safe way by the server. Take a look at the deals provided by the website and try to take advantage of the poor parameter handling in terms of a template injection. The goal is to read the flag.txt file and print its content.

Hint. There are some blocked strings to prevent code execution, try to find a way around it.

7 Additional Resources.

Here are some links to additional resources:

- OWASP Top 10: `https://owasp.org/Top10/`
- OWASP Cheatsheet Series: `https://cheatsheetseries.owasp.org/`
  - e.g. SQL Injection Prevention `https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html`
  - e.g., XSS prevention `https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html`
- Flask documentation: `https://flask.palletsprojects.com/en/2.0.x/`
- Requests documentation: `https://docs.python-requests.org/en/latest/`
- Introduction to JWT: `https://jwt.io/introduction`
- BurpSuite, an example of a web security testing toolkit: `https://portswigger.net/`

8 Assignment interview

In order to help you prepare for the assignment interview, we have a small overview of possible questions. First off: Both team members should be able to answer, regardless of who actually solved the challenge.

You will need to explain how your attack works in detail. More precisely, think about the following questions:

- Which security vulnerabilities did you find? How can they be exploited, in general?
- Explain your exploit in detail. In particular:
  - What is the general idea of your exploit? How can the security vulnerabilities be exploited?
  - What are the steps to reach your goal?
  - Explain the ingredients to your exploit. How did you find the vulnerabilities?
- Which security mechanisms would have prevented this exploit? What programming error could have been prevented?

9 Version History

v1.1

- Clarified that the one of the flags of SQLi2 has a different format.
- Updated skeleton files, correcting minor mistakes like a wrong shebang.

v1.0

Initial release of P3 assignment sheet.