Ciências ULisboa | Informática

Sistemas de Software Seguros (SSS)

Mestrado em Segurança Informática

Segurança de Software (SS)

Mestrado em Engenharia Informática

Mestrado em Informática

2024/2025

Project

1. Objectives

This assignment aims to give the students an insight into vulnerabilities found in software packages and how they can be attacked. The assignment uses two software packages, `vulnApp` and `sss-db`, written in C/Python and PHP. Most of the software needed for the project is provided in a VirtualBox image with Xubuntu Linux 18.04.1 (the user/password is `ss/ss`, and the user has `sudo` privileges to go to root).

The project is divided into two parts:

- The first part requires manual analysis of the code in `vulnApp` and, in some cases, exploiting the vulnerabilities in `sss-db`. It might be necessary to use the ZAP proxy to perform a few of the attacks.
- The second part uses several tools to perform the analysis automatically. The assignment resorts to the static analysis tool Flawfinder and the fuzzer AFL. You will also do some experiments with ChatGPT.

Most software needed for the project is already installed in the VirtualBox image. The resources section at the end of this document gives more information about the VirtualBox image.

First part

2. vulnApp

vulnApp includes three applications, called `map_path`, `mime`, and `mysig`. All these applications are based on large open-source projects written in C language. One or more vulnerabilities can be found in each of these applications. This first part of the assignment aims to find these vulnerabilities by auditing the software manually.

> *Source code: the source code for vulnApp is available in* `/home/ss/apps/vulnApp`.
> *Note that these applications **might not even compile**!*

The first task is to understand how the applications work and how they can be attacked. Answer the following:

01 Characterize the attack surface of each application. Justify your answer by relating it to the code.

Consider only the C language part of the application and do the following:

02 The `map_path` application has several vulnerabilities, including several *buffer overflows*. How many buffer overflow vulnerabilities do you find, and where (line number) do they occur? Please describe how the vulnerabilities can be exploited, i.e., that they can be attacked successfully (this requires providing details on how data supplied from outside the program reaches and exploits the vulnerabilities).

03 Describe an input to the program that exploits one of your identified vulnerabilities.

04 The `mime` application also has several *buffer overflow* vulnerabilities, this time in the heap and related to pointer manipulation. Please find one of the vulnerabilities and explain why it is exploitable.

05 How should the program be modified to prevent this vulnerability? Explain how your patch is effective at preventing the exploitation of the flaw.

06 The `mysig` application may have an *integer overflow* flaw that allows for a buffer overflow. Could you confirm if this problem exists? Explain how the adversary could exploit the bug. (NOTE: the application may have other bugs, but these bugs **should NOT be considered**)

07 Explain how you would modify the application to prevent the flaw's exploitation.

3. SSS-DB

SSS-db is a web application designed specifically for the course. It was written in PHP and runs on an Apache web server with a MySQL/MariaDB database management system. This part of the project aims to find and exploit vulnerabilities in this application. There is no further documentation on the application; you have to run it and read the source code to understand what it does (but you don't have to delve into the details of PHP; you only have to have a generic idea about its syntax).

Source code: the source code of sss-db is available in `/opt/lampp/htdocs/sss-db`

Running sss-db: the application is simply a set of files made accessible by the Apache webserver, which is already running. Therefore, you merely have to run the web browser (Firefox) and open http://localhost/sss-db/

The application's entry page is file `index.php`, which includes the files `header.php` and `footer.php`, as well as other files `*.inc`. There is an option to reset the database whenever you want (the third option in the Core Controls menu entry). There is also an option that changes the current security level and, therefore, the defenses that the program employs to protect itself from your attacks. Carry out the tasks below with a security level of 0.

Do the following tasks:

08 The Show User Info option of Operations is vulnerable to a SQL injection attack (file `user-info.php`). Perform this attack and get a listing of all the users and passwords of the application. Look at the file `user-info.php` and **explain why** this happens. Notice that the syntax of the SQL queries must follow MySQL's conventions.

09 The Show Log option of the Core Controls menu entry allows a *stored XSS* attack. **Explain why** and describe how the attack could be performed (the attack can show a popup window saying "hello").

10 The option Text File Viewer of the Operations menu allows the display of a specific document about hacking (file `text-file-viewer.php`). This option is vulnerable to a *reflected XSS* attack even though the user does not write any text, and they can only choose a document from a fixed list. Explore this vulnerability to show, for example, a popup window saying "hello." (HINT: maybe you can use the ZAP proxy tool to change one of the requests to the web application. See the end of the document for further information on ZAP).

11 **Explain why** this vulnerability exists by looking into `text-file-viewer.php`. How could you protect yourself from this attack? (HINT: You may get some inspiration by looking at the part of the code that is executed at a higher security level.)

12 The DNS Lookup option of the Operations menu entry is vulnerable to a *command injection* vulnerability (file `dns-lookup.php`). Explore this vulnerability to get the credentials of access to the application's database (host, database name, username and password). (HINT: usually, an application contains a configuration file with these credentials. Also, maybe you can use the ZAP proxy tool to see the application's responses. See the end of the document for further information on ZAP).

13 **Explain why** this vulnerability exists by looking into `dns-lookup.php`. How could you protect yourself from this attack? (HINT: look at the part of the code executed with a higher security level).

Second part – Use of tools

4. Flawfinder

`Flawfinder` is a simple static analysis tool that can be used to check programs written in C and C++. You can experiment with the various options of this tool, but for the report, you should only analyze the output from the commands specified in the following questions. Use `Flawfinder` to analyze the `mysig` application from vulnApp.

14 At the folder of the tool `(/home/ss/apps/flawfinder)`, execute the next command

```
./flawfinder -m 2 ../vulnApp/mysig/*
```

- For each error reported by the tool, indicate if it is actually a vulnerability. Justify your answer in the report.

To help you organize your answer, the following table should also be included in the document. The table should have a line per reported error, indicating if it is a False Positive (or false alarms) or a True Positive (or correct alarms).

15 Execute the same command but add the -F option (does not output False Positives)

```
./flawfinder -m 2 -F ../vulnApp/mysig/*
```

- Compare the results with the analysis you did in the previous question. Is the tool precise in its results? Justify your analysis and answer in the report.

6. AFL

16 Use the AFL fuzzer tool to discover vulnerabilities in an extensive library made vulnerable by injecting a few bugs. While compiling the code, several applications are generated that call the library (see below). We will test the `tiffcp` application for this exercise. Carry out the following steps:

1. Take **three of the input files** generated by AFL that should cause a crash of the application `tiffcp`. (i) Confirm these crashes arise when running the application with the input files. (ii) Determine where the crash is occurring in the application code. (iii) Explain what coding error/vulnerability is causing the application to crash.

NOTE the following:

(a) the command that you use to test could be (you may need to provide full paths):

```
tiffcp -M AFL_crash_file test.out
```

(b) since the name of the files created by AFL has a particular format (for example, "`id:000000,sig:11,src:000078,op:int32,pos:44,val:+1`") that might not be well understood by `tiffcp`, you may need to rename them to something more straightforward (for example, `crash_file_1.tiff`).

(c) you may want to use gdb (or some other debugger) to help you understand where the program crashed.

To generate the input files that cause the crashes, you need to do the following:

i) Compile everything (Note: there are a few commands that are pretty big, so I have formatted them to the left of the page):

```
cd ~ss/apps/AFL
make clean all

cd ../libtiff_zlib/
./configure --static --prefix="/home/ss/apps/libtiff/work"
make clean
make CFLAGS="$CFLAGS -fPIC"
make install

cd ../libtiff_libjpeg-turbo
cmake . -DCMAKE_INSTALL_PREFIX="/home/ss/apps/libtiff/work "
-DENABLE_STATIC=on -DENABLE_SHARED=off
make clean
make
make install
```

```
            cd ../libtiff_jbigkit/
            make clean
            make lib
            cp libjbig/*.a ../libtiff/work/lib/
            cp libjbig/*.h ../libtiff/work/include/

            cd ../libtiff
CC=/home/ss/apps/AFL/afl-gcc               CXX=/home/ss/apps/AFL/afl-g++
AS=/home/ss/apps/AFL/afl-as ./autogen.sh

CC=/home/ss/apps/AFL/afl-gcc               CXX=/home/ss/apps/AFL/afl-g++
AS=/home/ss/apps/AFL/afl-as     ./configure     --disable-shared     --
prefix="/home/ss/apps/libtiff/work"
            make clean
            make
            make install
```

ii) The executables were saved in `/home/ss/apps/libtiff/work/bin`. Now we need to setup the OS to run AFL. You need to run the following command as root, and then you should go back to user ss

```
echo core >/proc/sys/kernel/core_pattern
```

iii) Now you can run the fuzzer

```
cd ~ss/apps

/home/ss/apps/AFL/afl-fuzz  -m  100M  -i  /home/ss/apps/libtiff/corpus  -o
/home/ss/apps/libtiff/out_AFL -- /home/ss/apps/libtiff/work/bin/tiffcp -M
@@ tmp.out
```

When the screen indicates that there are "`uniq crashes`", it means that the tool has generated inputs that cause the crash of the application. You can find these inputs at:

```
/home/ss/apps/libtiff/out_AFL/crashes
```

In any case, you should let the fuzzer for one hour to increase the probability that AFL finds different vulnerabilities.

7. Large Language Models

Large Language Models, such as the ones that power sites like ChatGPT, can process code from multiple programming languages. In this question, we will determine how well they perform at two tasks: (i) identify vulnerabilities in the code and (ii) propose corrections to vulnerabilities that may exist in the code. To make the tests, we will be using the Python codes in the directory: `/home/ss/apps/vulnApp/LLM`

`COLE.py` – this code may be vulnerable to a path traversal flaw

`LEAKS.py` – this code may be vulnerable to an OS command injection flaw

17 Analyze manually the two codes and determine if they are vulnerable or secure to the abovementioned flaws. Your answer should justify your conclusions carefully.

18 Imagine that you are the developer of the two codes and would like ChatGPT to help you determine if they are vulnerable. It would be best if you designed the most appropriate question for ChatGPT so that it provides the most helpful response (this is called *prompt engineering*). In your report, you must explain the steps you have taken to create the most effective question for ChatGPT.

Consider the following: Did ChatGPT provide the correct response? Is the response helpful to check if there is a vulnerability in the code? Does the response indicate the line number where the flaw is located? Is the response very generic, or does it explain with a good level of detail how the flaw could be exploited? In your question, should you provide the whole code or just the code around the lines that may be vulnerable? Should you place the initial part of the file containing the includes together with the vulnerable lines?

19 Ask ChatGPT to provide you with a fix/patch to the codes that were considered vulnerable. Again, you must explain in detail the steps you have taken to obtain the best response from ChatGPT. You must also discuss the reactions you have received, namely, whether they were appropriate.

NOTE: Sometimes, when ChatGPT is not provided with the right question, it produces a wrong response. Therefore, I will evaluate your efforts to develop the most effective questions/prompts. For example, you should discuss why the ChatGPT response is incorrect. In addition, you should describe how you modified your question to seek a better response.

8. Delivery

The project's output is a report answering all the questions and issues raised in the sections above and justifying all the answers. The report is submitted on the course Moodle page; if there is some difficulty with this method, it is emailed to the professor. The file type must be a pdf. Each group must also deliver a **printed copy** of the report to the professor or place it in the professor's mailbox at C6.

Deadline: **9 December 2024 (there will be no extensions)**

9. Bibliography

Course bibliography.

10. Resources

IMAGE

The image is the one we have been using in the classes, but it can be obtained from:

https://cloud.admin.di.fc.ul.pt/index.php/s/AEfK8g8oQDiiyPJ

Since the image is around 4 Gbytes, downloading takes some time. Therefore, it is advisable that either:

- use a pen to have a local copy of the image
- alternatively, try to run the project on your machine because it can take some time to set up everything on a computer.

SETUP THE IMAGE IN VIRTUAL BOX

The VirtualBox player is available at:

`https://www.virtualbox.org/wiki/Downloads`

To setup the image in the VirtualBox player do the following steps:

- Startup VirtualBox software
- Use `File->Import appliance` to setup the image
- Startup the machine

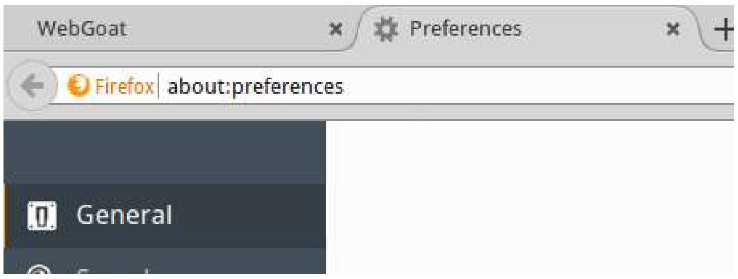
ZAP PROXY

To understand better what is happening in the communication between the browser and the web application and also to help the execution of a few of the attacks, we will use a web proxy developed by OWASP called ZAP[1]. To launch ZAP, you need to create a new terminal and run the command:

```
~/apps/ZAP/ZAP/zap.sh
```

After doing this, you should have a new window with the ZAP.

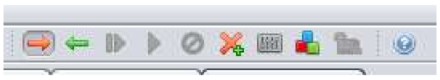
Next, you need to configure the browser to start using ZAP as a proxy. On Firefox, you can select the button `"open menu" -> "Settings"` or indicate `"about:preferences"` in the URL field:



At the end of the `"General"` page, select Network Settings the `"Settings"`. On the new window, choose `"Manual proxy configuration:"` and indicate as HTTP Proxy `"localhost"` and as Port `"8088"`. Additionally, select `"Also use this proxy for HTTPS"` and remove all information from the "No Proxy for:" field.

[1] There is some information on the tool on the ZAP project page where you will find links to the manual (just search for "ZAP OWASP" in your favorite search engine)

To intercept the requests performed with the browser with ZAP, set the button "`Set break on all requests`" (the green right arrow). The intercept request operation is already set if the right arrow is orange.



After the break on a request, you can modify the message's contents, and then you can send it to the web application by using one of the two buttons to submit the request (the two blue arrows next to the orange/green right arrows).

When you finish using ZAP, you need to change again the Network Settings configuration of Firefox to "`Use system proxy settings`".