Computer Security

Coursework Exercise CW2

Web Server and Application Security

School of Informatics
University of Edinburgh
`http://www.inf.ed.ac.uk/teaching/courses/cs`

This is an **individual assessed coursework exercise**. It will be awarded a mark out of 25. It is one of two assessed exercises in the Computer Security course. Each exercise is worth 12.5% of the final result for the course. You can expect to spend about 10 hours on this exercise, plus time for the required reading. The deadline for completing this coursework is **4pm, Tues 25th March 2014**. The final page summarises submission instructions.

In this coursework exercise we will guide you through an attack against a vulnerable machine. You will take the role of "Mallet" attacking a web side that was programmed by "Alice". The successful run through this attack will require some reading about tools and attacks. The questions are intended to encourage you to experiment with the given machines and tools, to give a better understanding of security mechanisms and exploits.

Preparation

The Virtual Machines. For this exercise we are using Virtual Machines (VMs) inside VirtualBox. This provides a convenient way to experiment with systems that might be vulnerable and to test them against known attacks. The VMs we are using come preconfigured and preinstalled with useful tools. The victim is called Alice, and the attacker is called Mallet; they each have a machine. The two virtual machines we are using in this exercise are accessible from all DICE machines in `/group/teaching/cs/`, in a compressed *virtual appliance* format (about 900Mb each).

Notations for names. We will use names like "alice" in several ways. In the examples we use we introduce one protagonist, which serves as the victim; her name is Alice, written with a capital A. The victim Alice has a computer; the name of her system is **alice** and is printed bold except in user inputs. Finally on the machine **alice** there is a user account named *alice* which is printed italic except in user inputs. The same notations apply for Mallet.

Setup The machines come as *.ova files, which include the virtual hard drive as well as all necessary settings for VirtualBox. VirtualBox can be downloaded at `https://www.virtualbox.org/` and is already installed on all DICE computers. Once VirtualBox is running, select *File↪Import Appliance* and select the file `alice_v1.1.ova` or `mallet_v1.1.ova`. The only property in the configuration screen you might want to change is the location of the virtual hard drive. On DICE computers a home directory account does not have enough quota to store the hard drive, so you will need to store the virtual disk locally on the workstation. The disks take up about 2.3G each once expanded. Go to the last line *Hard Disk Controller↪Virtual Disk Image*. Change the path from, e.g.,

`/afs/inf.ed.ac.uk/user/sXX/sXXXXXX/VirtualBox VMs/mallet_1/mallet_v1.1-disk1.vmdk`

to

`/tmp/sXXXXXX/VirtualBox VMs/mallet_1/mallet_v1.1-disk1.vmdk`.

(you may need to create the directory `/tmp/sXXXXXX` first). The import operation can take several minutes and need about 3G of disk space per machine.
IMPORTANT: On DICE machines the virtual disk has to be stored on the local disk. That means that you will not be able to access the virtual machines on any other DICE computer than the one you did this setup on. If you want to use your virtual machines from another computer you will have to log in to the machine using `ssh -X <computername>`, or copy the disk files to another machine with `scp`; so make sure you remember the name of the computer you used!

Exercise

We will go through an attack against the computer **alice**. To begin, you will need to execute both virtual machines **alice** and **mallet** at the same time. Once **mallet** is started, login with the user *mallet* and password `mallet`. You will find a standard Gnome desktop environment. On **alice** you can login as *alice* with the password `alice`. However, Part A only requires **alice** to be running, a login is not necessary yet.

Part A: Scanning

The first thing we want to do is to have a look at the victim's system and its vulnerabilities. To do this attackers can use freely available software tools. One of these tools is **Nmap**, the *Network Mapper*, which is installed on **mallet**. More information for the command can be obtained on the man-page. Some of the functionalities of Nmap require root privilege. In this case it helps to execute the command as `sudo nmap <parameters>` and provide *mallet*'s password `mallet`.

1. From **mallet** run a scan of **alice** with the tool nmap. Make sure to include the parameters `-O` and `--version-all -sV` to get as much information about the system **alice** as possible. Report back all information you got about the system.

2. One piece of information that can be provided by nmap is a list of the open ports and the corresponding protocols. For each port you discover on **alice** provide a short summary about the protocol serving this port, how an attacker could connect to this port and what the security risk of an attack would be.

3. Nmap offers a variety of different scanning methods for example TCP SYN scan, TCP ACK scan and the XMAS scan. Explain the differences and the advantages of each method.

After this first gathering of information we proceed with a more automatic method. The program **OpenVAS** is a collection of tools to automatically scan for vulnerabilities. It consists of a daemon and a user interface. To start the daemon type `sudo openvasd` into the command-line. Once it is loaded start the user interface by typing `openvas-client` into another command-line-window. All options are already adjusted properly.
Note: Do not connect to the openVAS-server without creating the task as described below or the client will freeze.

4. Perform a full scan of the system **alice**: Hit the *Scan assistant* button "?" at the upper left corner, provide reasonable descriptions for the task and the scope and hit *Execute*. In the next window you will be asked to provide the password for the server, which is again user *mallet* and password `mallet`. This test may take a while and cause virtual box to temporarily stop responding.
   Report back the 4 most dangerous security risks found and explain why they are the most dangerous.

In the previous results we have seen that port 80 is open at **alice** and OpenVAS reports that there might be a unverified input vulnerability in the web pages served at this port. The web pages are generated by a simple PHP application to communicate through short snippets of text posted on a private message board. This service is available by typing `http:\\alice` into the URL line of a browser on **mallet**.

5. Look around on this web page and give at least one possible example for each of the following terms: Owner, Assets, Threat agents, Threats, Risk, Vulnerabilities and Countermeasures.

6. Which of the security properties that you have seen in lectures have been considered and how? Is there a property that has been neglected?

Part B: Session stealing

To find and exploit a vulnerability on a web page it is helpful to know the structure of the page. To do this, Mallet has already created a user account for herself on the web application. The username is *mallet* and the password is `mallet123`.

1. Create a map of the structure of this web page: try to provide answers to the following question in a suitable diagram. The specific type and form of the diagram is up to you.
   - What different kinds of screens does the application show?
   - How are they related, what are the transitions between them?
   - What data do the transitions carry and which method is used [e.g. GET, POST, cookies]?
   - Mark two possible vulnerabilities. (other than those considered in the following)

Having collected some information data, we are now going to try *black-box* attacks against **alice**. That means we are not assuming any knowledge about the code itself. Since this application is programmed in PHP this is a reasonable assumption; PHP code is server-side code and thus never delivered to the user's machine.

Our first attack point is the transmission of a new message. This is done using the URL and the POST method of HTML. We can see that the information included into the URL after pressing the submit button on the massage board includes the message in plain-text, the screen that should be displayed and a parameter that is called 'sid'. Not included are for example the name of the user, which is probably stored in a database at the server. It is not hard to guess that the 'sid' parameter is the session id, which links the request to the user in the database on the server. Note that it is not verifiable whether the URL sent to the server (including the data) was generated by the HTML form, or simply typed in directly by the user.

2. By logging in and out several times find out how the session id is computed, discuss the practicality of this method and propose a safer solution.

Sending data to a server using the session id of another user is called "session stealing".

3. Steal another session: On the machine alice log into the web page `http:\\alice` as *alice* with the password `alice123`. Now get back to **mallet** and forge a URL with data to publish a message as *alice*. Describe all details of your method. What are the preconditions and limitations of this attack?

Part C: SQL Injection

Another category of attack is the so called *white-box attack*. For this type of attack we assume we know at least a part of the source code. For this exercise we assume that Alice carelessly gave away the SQL query containing the lookup for the password. It is:

```
SELECT id FROM users WHERE username='$uname' AND password = '$pwd'
```

The variables $uname and $pwd contain the user input from the two input fields in the login form. Alice does not check them further but just passes them to the SQL command. This gives us the chance to insert code into this command. SQL code entered into the input fields will be inserted into this SQL command and executed. This attack is called SQL injection. A usual injection string consists of the following parts:

- characters to close the opened environments (e.g. ' or closing brackets)
- the malicious code

- characters to prevent syntax error resulting from the injection: Either reopen all necessary environments or hide following characters as comment using the sequence " `--` " (note the blanks).

As an example entering the username `dont know' OR '1'='1' --` will result in the query being

```
SELECT id FROM users WHERE username='dont know' OR '1'='1' -- ' AND password = '$pwd'
```

which will be true for every user in the users-table.

1. Perform the described SQL injection. Describe the result and give a possible explanation.
2. Break privacy: Describe a sequence of login credentials, that will give you the list of all user names on this forum. Write down your input strings and give a short explanation what happens inside the system.
3. Break availability: Now that we can inject arbitrary SQL commands we can tamper with data that should not be accessible to any user, for example with the user database. Change the password for the user account *alice*. Write down the inputs you used and explain in detail why this works.
   (Hint: Look at `http://www.w3schools.com/sql/sql_update.asp` if you need help with the SQL commands.)

Part D: Fortification

In a similar way as we have seen above the user can inject HTML code into a page that displays user generated messages. For example the message

```
blubb</font><br><br><hr>
<font color="#336600">2014-02-27 12:34:56</font>
<font color="#ae1b2a">mallet</font> says: <br>
<font color="#555f6e">I owe you!
```

will be displayed as if mallet had posted "I owe you!". A prevention against this is *sanitising* the input of the user, before it is displayed. For messages that could potentially contain HTML code, we replace every "`<`" with "`<`" to escape the tags included in the message.

1. Write a function `escape_html($input)` in PHP or pseudo-code that takes a string as input and returns the sanitised string as explained above.

We have also seen that the session management is vulnerable. Thus Alice has decided to change the authentication method provided by the server she is using. In the file `/var/www/index.php` line 6 change the value of `$auth_type` from `get` to `basic`.

2. After the adjustment try to access the message board on Alice's web side from **mallet** and observe how the server's behaviour has changed. Use the Firefox add-on TamperData to listen to the communication between the browser and the server. How does the browser send the authentication information now?
   Note: You can still use the user *alice* with password `alice123`.
3. The new relation between usernames and passwords is now stored in a file `passwords` on Alice's computer. Try to download this file to **mallet**. Report how you were able to download the file and a possible countermeasure.
4. Use the provided program `john` on **mallet** to decipher the passwords contained in this file. Report the new password for the user *mallet*.

Submission Instructions

You should submit your answers in person on hardcopy, either a printed document or a solution in clearly legible handwriting marked with your matriculation number at the top of each page.

Please submit the ITO by the deadline of **4pm, Tues 25th March 2014**.

You're reminded that **late coursework** is not allowed without "good reason", see

`http://www.inf.ed.ac.uk/teaching/years/ug3/CourseGuide/coursework.html`

for more details about this, and the procedure to follow if you must submit late. In particular, if you have a good reason to submit late, please use the ITO support form to make a request.

Daniel Franzen and David Aspinall
6th March 2014