Computer Security
Coursework Exercise CW1
Web Server and Application Security

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 site that was programmed by "Alice". Then you will help "Bob" protect his machine from future attacks. 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.

This coursework is worth 50 points in total and accounts for 12.5% of the final mark for this course.

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. There are three virtual machines, **alice**, **bob**, and **mallet**. These are accessible from all DICE machines in `/group/teaching/cs/`, in a compressed *virtual appliance* format. The directory is an AFS mount so you may need to type the full path into a command line or into the folder GUI to see the files. Tab completion will not work and the containing folders will appear empty.

We have provided two versions of the **mallet** attacker VM. The simpler VM is called **mallet** and can be found in *mallet.ova*. This is the version that is guarenteed to run on the DICE lab computers. We have also provided a more advanced **Kali Linux** installation which is labeled *kali-large.ova*. Both versions will work identically for this coursework and both have the key tools like nmap and OpenVAS installed. The **Kali Linux** install is newer and nicer to work with from a usability perspective especially if you are working from home. But both will provide the same answers to the coursework. For the remainder of the document we will be refering to the **mallet** VM. You can safely use the **Kali Linux** VM anywhere we say **mallet**.

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 **bob** there is a user account named *bob* which is printed italic except in user inputs.

Kali Linux Mallet's virtual machine is **mallet** (or **Kali Linux**[1] , with some modifications). The default password for *mallet* is `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 one of the files `alice_v1.2.ova`, `bob_v1.0.ova`, or `kali.ova`. The only property in the configuration screen you should

[1] `https://kali.org`

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/alice_1/alice_v1.2-disk1.vmdk

to

/tmp/sXXXXXX/VirtualBox VMs/alice_1/alice_v1.2-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

This coursework consists of two halves, each containing two parts. In the first half, you will exploit vulnerabilities in a website running on **alice**. In the second half, you will identify and patch vulnerabilities on **bob**.

The first two parts will require running the both the **alice** and **mallet** virtual machines at the same time. You will not be able to log in to **alice**, and can only interact with the machine through the web page it is hosting. This can be accessed from **mallet** at `http://alice/`.

For the final two parts, you will need to run the virtual machines **bob** and **mallet** simultaneously. You can log in as *bob* using the password `bob`.

Part A: Session stealing (10 points)

To find and exploit a vulnerability on a website it is helpful to know the structure of the site. That is, to know what the different pages are and how they communicate with one another. 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`. From Mallet's virtual machine **mallet**, access Alice's homepage at `http://alice/` and do the following:

1. Explore the entire website by clicking on all the links and buttons to see what they do.
2. For each page transition (link or button) on the website, provide the information below. Some page transitions have different behavior based on the state the page is in. So you may need to provide more than one record for each link to fully describe the behavior of the site. It may help to draw the behavior of the page as a state transition diagram.
    - Page State: Does this link or button's behaviors depend on earlier events or actions. (For example, buttons behave differently if the user is logged out vs. logged in.)
    - Link Text: The text on the link or button. (For example: "Home".)
    - Data: What data is being transferred between the pages and how it is being transferred (GET or POST).
    - Effect: Describe any effect clicking the link has. (For example: causing a message to appear on the list of messages.)

Example record:

- Page State: Initial start page with no content and not logged in.
- Link Text: "Home"
- Data: No data is sent
- Effect: Page reloads with no change.

You should turn in either: a) a state transition diagram which contains all the information described above, or b) a set of text records containing all the information described above with at least one blank line between the records.

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 Alice's web page is programmed in PHP this is a reasonable assumption; PHP code is server-side code and thus never delivered to the client's machine.

Our first attack point is the transmission of a new message. This is done using the URL and the GET 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.

3. By logging in and out several times find out how the session id is computed. Describe the algorithm used to compute sid when a user logs in (do not over think this).

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

4. Steal another session:
   - Open two Firefox windows
   - On the first window log into the web page `http://alice/` as *bob* with the password `bob123`.
   - On one window log into Alice's web page as *alice* with password `alice123`
   - Now go to the second window and forge a URL with data to publish a message as *bob*.
   - Describe all details of your method. What are the preconditions and limitations of this attack?

Note: you can also do the above attack using multiple VMs. If you log into Alice's web page from **bob** you should be able to steal the session from **mallet** but you need to run all three VMs at the same time to do it, which is difficult on the DICE computers.

Turn In: Answers to 2, 3, and 4 above.

Part B: SQL Injection (15 points)

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 confidentiality: Describe a sequence of login credentials, that will give you the list of all user names on this forum. Write down your input strings, give a short explanation what happens inside the system, and provide a list of all the users on this 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.)

Turn In: Answers to 1, 2, and 3 above.

Part C: Scanning (10 points)

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.

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

   ```
   nmap -O –version-all -sV bob
   ```

2. List all the open port numbers and the MAC address.
3. Nmap offers a variety of different scanning methods. In particular: TCP SYN scan, TCP ACK scan and the XMAS scan. Explain the differences and the advantages of using each of these three scanning approaches.

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. The commands to use OpenVAS are different for the **mallet** and **Kali Linux** VMs, please use the instructions for the VM you are currently using.

Mallet: 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.

Perform a full scan of the system **bob**: 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.

Kali Linux: To start the daemon, run `sudo openvas-start` from a terminal. This command will take up to 15 minutes to finish. Once it is loaded start the user interface by navigating a web browser to `https://127.0.0.1:9392/`. Login using the username *admin* and the password `admin`.

Perform a full scan of the system **bob**. This test may take a while and cause virtual box to temporarily stop responding.

4. Report back the 4 most dangerous security risks found and explain why they are the most dangerous. Answers must contain a clear reason, not simply a quote of the text or "because it said 10".

Turn In: Answers for 2, 3, and 4.

Part D: Firewalls (15 points)

Warning do this part AFTER completing part C.

Bob uses his computer for two things:

- Using SSH to connect to his computer remotely (ssh).
- As a server for his website (http). (Although he hasn't found the time to set it up yet.)

The earlier port scan showed that **bob** has more ports open and responding than are strictly necessary to do the two things above. Help Bob setup his iptables firewall to block malicious attacks. The firewall should only allow http and ssh connections. Bob is also concerned that other people will try and brute force guess his ssh password so you need to find a way to rate limit connections to ssh so that no IP can send more than 60 connection attempts each minute. When you are done, the website `http://bob/` should be accessible from the browser on both **mallet** and **bob**. It should be possible to ssh into **bob** from **mallet** using *bob*'s username and password. Finally, an nmap scan of **bob** from **mallet** should show only two open ports (http and ssh).

The iptables command requires root privileges, so you will need to execute the command as `sudo iptables`.

You are welcome to use any online resource to complete this part of the exercise. The tutorial at the link below is recommended as is reading the man page for iptables.

```
https://help.ubuntu.com/community/IptablesHowTo
```

Turn In: The list of iptables commands you used to correctly setup the firewall and the output of the command: `sudo iptables-save`.

The following commands are also useful.

To see the current iptables rules use:

```
sudo iptables -L
```

To reset the rules to their default empty state use:

```
sudo iptables -f INPUT
```

To ssh into **alice** from **mallet** use:

```
ssh alice@alice
```

Submission Instructions

You should turn in one plain text file with sections for each part of the assignment. This is so that it is possible to see which part of the file answers which question. Do not submit a Word document or a PDF file - only submit a plain text file.

IMPORTANT: You must create your text file on your local computer, NOT the VM. The VMs we are using are isolated from the host system for everybody's safety. There is no easy way to copy files between the VM and the local computer. The submit command will not work from within the VM. You must use the local command line on a DICE computer to use submit.

Your text file should be named cw1.txt. Organize your file into sections for each part of the exercise and clearly label each answer with the number associated with it. An example submission file should look like this:

```
####
# Part A
####

Question 1
Answer


Question 2
Answer

   ...

####
# Part B
####
...
```

Use the submit program to submit the cw1.txt text file. You may submit the assignment as many times as you like before the deadline. To submit from the command line on a DICE machine use the following command:

```
submit cs 1 cw1.txt
```

If you choose to submit an image file for Part A question 2, then name the file cw1-b2.jpg. This is the only question for which we will accept an image. Separately submit the image file using the following command:

```
submit cs 1 cw1-b2.jpg
```

You need to submit by the deadline of **October 28st at 16:00**.

If for any reason you are unable to use the submit command you may also submit to the ITO 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.

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 `http://www.inf.ed.ac.uk/admin/ITO/support/index.html` to make a request.

Good Scholarly Practice:

Please remember the University requirement as regards all assessed work for credit. Details about this can be found at:

`http://www.ed.ac.uk/academic-services/students/undergraduate/discipline/academic-misconduct`

and at:

`http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

Furthermore, you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on a public repository then you must set access permissions appropriately (generally permitting access only to yourself, or your group in the case of group practicals).

[2]

[2]Exercise originally created by Daniel Franzen and David Aspinall. Modified by Kami Vaniea and Thomas Kerber.