Secure Computer Systems I: Lab 3

Ren Li Tianyao Ma Samuel Pettersson

March 4, 2014

Host 1: 192.168.233.20

Host 2: 192.168.233.110

The second host to which root access was gained is 192.168.233.110. The instructions said that access to host 192.168.233.20 was required to access this machine.

Running nmap showed that two services were running on the target machine: Microsoft Windows RPC on port 135 and Microsoft Terminal Service on port 3389, the latter being used for serving remote desktop clients.

With no obvious vulnerabilities being present in the RPC service, attention was turned to the remote desktop service. The RDP client of our choice was (initially) rdesktop, which is available in the software package repository and was installed by issuing the command sudo apt-get install rdesktop.

Connecting to the other machine was as simple as running the command rdesktop 192.168.233.110. Upon doing so, we were presented with the Modern UI login screen. Authentication as the administrator appeared to require a smart card, whereas password authentication was an option for other users.

After some tinkering with a remote desktop connection on host 192.168.233.20, we found out that password authentication as the administrator on 192.168.233.110 was possible by starting a remote desktop session from 192.168.233.20 to 192.168.233.110. One could suspect that the administrator password for the two machines were the same, but despite having root access to 192.168.233.20, we were unaware of what the password to the administrator account was.

The immediate thought was to dump the password hashes on host 192.168.233.20 and have them cracked either by brute force or with a dictionary. Dumping the hashes is simple enough with a meterpreter shell: issuing the command run hashdump does the trick. For each user, two hashes were stored: an unused LM hash of the empty string and an NT hash. Unfortunately, the version of John the Ripper available on the BackBox machine did not have support for breaking NT hashes. The source code of the latest "jumbo" version of John the Ripper (version 1.7.9-7) with support for NT hashes was downloaded from `http://www.openwall.com/john/`. Building the application was a matter of running two make commands: make and make clean linux-x86-mmx as per the README and the installation instructions. After having run the jumbo version of John the Ripper for several hours without finding a password matching any of the dumped NT hashes, let alone the hash for the administrator account, the trail grew colder and we gave up on retrieving the password.

After having received a hint from Sofia, we decided to look into the possibility of passing the hash of the administrator on 192.168.233.20 (CL3\Administrator). That is, instead of authenticating with a password, we hoped to be able to authenticate with the hash of the password. Some research showed that Microsoft's Remote Desktop Protocol version 8.1 from last year introduced a Restricted Admin mode, in which credentials in the form of passwords are not sent to the remote server; instead, password hashes are sent. This mode was introduced in an effort to improve security in that authentication to a compromised server does not reveal the password in cleartext, but unfortunately, it comes at the expense of allowing pass the hash attacks[1]. Further research showed that an RDP client by the name FreeRDP with support for passing hashes was available in the form of a Git repository at GitHub[2][3].

The repository was cloned and the application was built by following the installation instructions available at GitHub[4]. The necessary packages were installed, cmake was used to generate Makefiles, and make was used to build and install the application. After having completed the installation, the client was available by the name xfreerdp.

The following command was executed to attempt to log on as the administrator with the hash retrieved from host 192.168.233.20: xfreerdp /u:Administrator /pth:fabc417905666832e4b4ba57711a4171 /v:192.168.233.110. Passing the hash turned out to work! A text file containing the secret code without any encryption was easily spotted on the desktop; the code read: vZxkRvEICzhNQ.

Host 3: 192.168.233.106

Host 4: 192.168.233.30

An nmap scan of the fourth and final victim showed that there were two services running on the host, namely, an SSH service and an HTTP service. The command and the most important parts of its output is shown below.

```
security@BB:~$ nmap -v -A 192.168.233.30
...
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 5.9p1 Debian 5ubuntu1.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey: 1024 e8:db:f6:d5:52:fd:ab:68:08:2b:7d:fe:6f:24:df:2a (DSA)
| 2048 91:58:ee:38:84:98:82:42:33:86:c2:c7:04:e0:1e:f2 (RSA)
|_256 b6:d6:f7:fa:a5:c2:44:7e:08:49:db:06:31:b4:b6:2d (ECDSA)
80/tcp open  http    Apache httpd 2.2.22 ((Ubuntu))
|_http-methods: GET HEAD POST OPTIONS
|_http-title:       Nuyorican Poets Cafe
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
...
```

The v flag specifies verbose output, and the A flag specifies that OS and service detection (among other things) should be carried out.

Neither of the services appeared to have vulnerabilities that could easily be exploited, so, seeing as we knew no users on the system, it was a natural decision to look for information at the website that the HTTP server hosted first. Browsing to 192.168.233.30 in a web browser showed the website of Nuyorican Poets Cafe. When looking at the page on the history of the cafe (192.168.233.30/history.php), three particularly interesting names were found: Daniel Gallant, Jason Quinones, and Mahogany Browne, listed as programmers. Furthermore, there were mailto links that would indicate that their usernames were daniel, jason, and mobrowne.

With these usernames, we could carry out a dictionary attack on their accounts over SSH. The tool that we used for this was the network logon cracker THC-Hydra[5], and the word list provided to the program was the list of common Unix passwords available in /opt/backbox/msf/data/wordlists/ on our BackBox system. Within a few minutes, Hydra had discovered that cuteme was the password for the user jason. The important lines of output are shown below:

```
[DATA] attacking service ssh on port 22
[STATUS] 256.00 tries/min, 256 tries in 00:01h, 2753 todo in 00:11h, 16 active
[STATUS] 218.67 tries/min, 656 tries in 00:03h, 2353 todo in 00:11h, 16 active
[22][ssh] host: 192.168.233.30   login: jason   password: cuteme
```

The obtained credentials for jason were used to log on via SSH. Running the command sudo -l to list the commands allowed for jason showed that he was allowed to run any command. The command find / -name *secret* was issued to find the secret file, and it showed that a file by the name secret.txt.enc was located in /root/; the file appeared to be encrypted. In the very same directory, there was a subdirectory called .keys/ that contained a file called passwd-file. It would seem that this file contained the password used when encrypting the secret. The content of the password file is shown below.

```
sdfsdtf3434trdsfvsdvsfdsdfsvzeqwrt34rerty5y6rthjfgnf
```

In order to analyze things more carefully, we copied the encrypted secret and the password file to the home folder of jason and then to our local machine using scp. The encrypted file started with a string "Salted__". Googling on the corresponding bytes in hexadecimal seemed to indicate that it was the work of OpenSSL. This agreed with similar cases in the solutions for the lab last year, which we had asked for and received from Sofia. Not at all knowing which of the 100 encryption methods shown to be available in openssl when executing openssl enc . had been used, we created a bash script for trying each and every one of them. The script is shown in Listing 1 below.

Listing 1: A Bash script for decrypting the secret.

```
cipherCommands=(aes-128-cbc aes-128-cfb aes-128-cfb1
aes-128-cfb8 aes-128-ctr aes-128-ecb
aes-128-gcm aes-128-ofb aes-128-xts
aes-192-cbc aes-192-cfb aes-192-cfb1
aes-192-cfb8 aes-192-ctr aes-192-ecb
aes-192-gcm aes-192-ofb aes-256-cbc
aes-256-cfb aes-256-cfb1 aes-256-cfb8
aes-256-ctr aes-256-ecb aes-256-gcm
aes-256-ofb aes-256-xts aes128
aes192 aes256 bf
bf-cbc bf-cfb bf-ecb
bf-ofb blowfish camellia-128-cbc
camellia-128-cfb camellia-128-cfb1 camellia-128-cfb8
camellia-128-ecb camellia-128-ofb camellia-192-cbc
camellia-192-cfb camellia-192-cfb1 camellia-192-cfb8
camellia-192-ecb camellia-192-ofb camellia-256-cbc
camellia-256-cfb camellia-256-cfb1 camellia-256-cfb8
camellia-256-ecb camellia-256-ofb camellia128
camellia192 camellia256 cast
cast-cbc cast5-cbc cast5-cfb
cast5-ecb cast5-ofb des
des-cbc des-cfb des-cfb1
des-cfb8 des-ecb des-ede
des-ede-cbc des-ede-cfb des-ede-ofb
des-ede3 des-ede3-cbc des-ede3-cfb
des-ede3-cfb1 des-ede3-cfb8 des-ede3-ofb
des-ofb des3 desx
desx-cbc id-aes128-GCM id-aes192-GCM
id-aes256-GCM rc2 rc2-40-cbc
rc2-64-cbc rc2-cbc rc2-cfb
rc2-ecb rc2-ofb rc4
rc4-40 rc4-hmac-md5 seed
seed-cbc seed-cfb seed-ecb
seed-ofb)

for command in "${cipherCommands[@]}"; do
```

```
37 	openssl $command −d −in secret.txt.enc −out output/$command −pass file:passwd−file
38 done
```

The first 34 lines of the script is just assigning a variable an array of the available encryption methods (or cipher commands). Line 36–38 constitute a loop whose body is executed once for each cipher command. The command on line 37 is the one doing all the decryption. The first appearance of `$command` specifies the cipher command, the d flag denotes decryption, the in and out parameters specify the input file to decrypt and the output file in which to store the result, and the pass parameter specifies the password to use when decrypting. As can be seen, the output file for each decryption is specified to be placed in the directory output and be named after the cipher command used.

After having executed the shell script, all the output files were manually scanned for intelligible information. It turned out that the output file corresponding to the cipher command aes-256-ofb contained the secret in plaintext. The code read: P2EOlbVN4vf5A, and the secret in its entirety is shown below.

```
  _____________ ___  _________               _______________  ____   _____
 /   _____/    |   \/   _____/ ____   ____   \_____  \   _  \/_   | /  |  |
 \_____  \|    |   /\_____  \_/ __ \_/ ___\   /  ____/  /_\  \|   |/   |  |_
 /        \    |  / /        \  ___/\  \___  /       \  \_/   \   /    ^   /
/_______  /______/ /_______  /\___  >\___  > \_______ \_____  /___\____   |
        \/                 \/     \/     \/          \/     \/         |__|

                        +------------------------+
                        |The Code this misson is:|
                        |     P2EOlbVN4vf5A      |
                        +------------------------+
```

References

[1] MRL. New "restricted admin" feature of rdp 8.1 allows pass-the-hash. Accessed 2014-03-04. [Online]. Available: https://labs.portcullis.co.uk/blog/new-restricted-admin-feature-of-rdp-8-1-allows-pass-the-hash/

[2] ——. Freerdp-pth. Accessed 2014-03-04. [Online]. Available: https://labs.portcullis.co.uk/tools/freerdp-pth/

[3] awakecoding. Freerdp. Accessed 2014-03-04. [Online]. Available: https://github.com/awakecoding/FreeRDP

[4] FreeRDP. Compilation. Accessed 2014-03-04. [Online]. Available: https://github.com/FreeRDP/FreeRDP/wiki/Compilation

[5] van Hauser. Thc-hydra. Accessed 2014-03-04. [Online]. Available: https://www.thc.org/thc-hydra/