University Of Agder

DAT 234

Scripting og Hacking

Final Project

Author:
Bendik Egenes Dyrli
Anders Refsdal Olsen

Supervisor:
Sigurd Munro Assev
Sigurd Kristian Brinch
Arne Wiklund

2015

Abstract

A hacking attempt on a fictive company with an infrastructure close to a real life scenario. The penetration methods used where social engineering of password combinations, exploiting of bugs in OS and software and bruteforcing. The group managed to break in to many systems and exploit multiple weaknesses. The findings proved that it was possible to break into the company with simple methods of hacking.

Contents

Chapter 1

Introduction

1.1 Group members

The Members of this group are

•Anders Refsdal Olsen

•Bendik Egenes Dyrli

Chapter 2

Description

2.1 Environment

The group used a workstation as an interface for the tools needed in Kali Linux v2.0. In such way the group was able to work on many tasks at the same time using screen. Used software was Metasploit for the hacking part, and python for scripting.

2.2 Users in the company

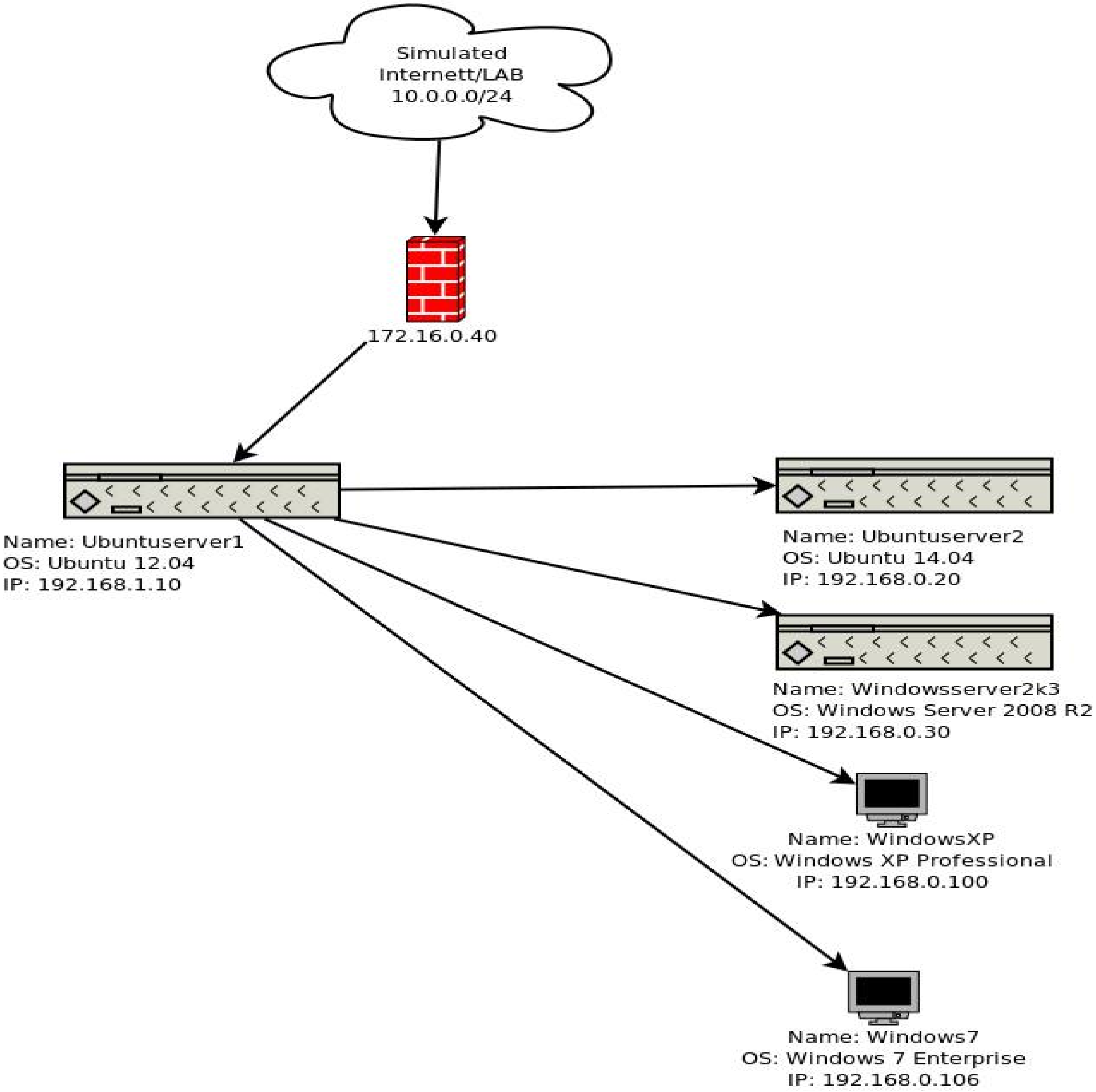
Based on social engineering we managed to obtain the following userlist and as described later in the task, the corresponding user's password. A script for obtaining the usernames is in attachment 2.

Username	Password
frankh	Turid40
arnek	6$tuthEb
turidf	Pearl1975
perj	access14
fridae	RaptorF150
linneaj	-Unknown-
alexs	-Unknown-
inee	-Unknown-
isabellap	-Unknown-
johannesl	-Unknown-
malins	-Unknown-
martins	-Unknown-
solveigl	-Unknown-
sondref	-Unknown-
martins	-Unknown-

We also tried to verify that all these users existed and for doing that we did a domain lookup through active directory. We then found out that all the other users where not a part of the domain and therefore none existent on the network.

2.3 Network topology

The group decided to build a map containing all the devices in the network:



Here is a list over all services running on the network. We found these running various port scanning techniques like syn and verbose scan.

2.3.1 192.168.1.10

Service	Port	Access
FTP	21/TCP	Everyone
FTP	21/TCP	Everyone
SMTP	25/TCP	Everyone
DNS	53/TCP (not UDP)	Everyone
HTTP	80/TCP	Everyone
POP3	110/TCP	Everyone
IMAP	143/TCP	Everyone
SSL/IMAP	993/TCP	Everyone
SSL/POP3	995/TCP	Everyone
RDP	3389/TCP	Everyone

2.3.2 192.168.0.20

Service	Port	Access
SSH	22/TCP	Everyone
NetBIOS	139/TCP	Everyone
Samba	445/TCP	Everyone

2.3.3 192.168.0.30

Service	Port	Access
DNS	53/TCP (not UDP)	Everyone
HTTP	80/TCP	Everyone
Samba	445/TCP	Everyone
RDP	3389/TCP	Everyone

2.3.4 192.168.0.100

Service	Port	Access
RDP	3389/TCP	Everyone

2.3.5 192.168.0.106

Service	Port	Access
FTP	21/TCP	Everyone
Microsoft DS	445/TCP	Everyone
RDP	3389/TCP	Everyone

Chapter 3

Results

3.1 Information gathering

3.1.1 Initial network communication

The first step we needed to do was to figure out which services was up and running on the given IP address (172.16.0.40). We therefore initiated a Metasploit workstation running on Kali which all our operations originated from. We also used this workstation to create SSH tunnels to our computer so that we could simulate and ”camouflaging” our identity.

3.2 Obtaining passwords

All the passwords where obtained from the information given on the company website or through social engineering. We used a small script that we pasted words into and it printed out all the information shuffled and restocked to give us any combination possible in a password text file. With that in place for each employee we could easily start a small bruteforce attack on the given username.

3.2.1 Frank Hansen

To obtain Frank Hansens password we used the social engineering technique and combined the information we knew about him from the company website. We knew that his wife was Turid and that he recently had turned 40 years.

3.2.2 Turid Frost-Hansen

On the machine we was to gain access to there was a company site on it. Just by looking around on the site we quickly found that the Chief of IT was Turid F-Hansen she was in a relationship With Frank who also worked in the same company. Frank had recently Turned 40years old. So we first though of social engineering, that it was the way to go to gain access to (172.16.0.40) By doing a port scan on the target we saw that it was a Ubuntu server and it had an open-ssh server. We could confirm this by netcatting the ip, and receiving a header stating Open-ssh Server and the version number. However we looked around the site for clues for what could the password be. We found out that Turid had a cat, since she had a picture of it on her personal page and if you opened the picture in a new Tab in the browser you could see the filename **”Pearl.jpg”** so we concluded with that name of her cat is Pearl, we also knew that she was in relationship with frank who recently turned 40.. we found out that he was born in 1975 by doing a fairly simple math problem **2015 - 40 = 1975**. So therefor the password was **Pearl1975**

On the machine we was to gain access to there was a company site on it. Just by looking around on the site we quickly found that the Chief of IT was Turid Frost-Hansen she was in a relationship With Frank Hansen who also worked in the same company. Frank had recently Turned 40years old. So we first though of social engineering, that it was the way to go to gain access to (172.16.0.40) By doing a port scan on the target we saw that it was a Ubuntu server and it had an open-ssh server. We could confirm this by netcatting the ip, and receiving a header stating Open-ssh Server and the version number.

3.2.3 Frida

This user was clearly interesting in cars especially those posted on hear profile. So we gathered all the words we could find in the images and made a wordlist containing all the words and used a script to make as many combinations as possible in a file, which we used with metasploit on ubuntuserver2 to bruteforce the ssh password.

3.2.4 Perj

Perj’s password was found using bruteforce on ubuntuserver2 with a password list generated by appendix_2_task2.py from ”top 500 worst passwords list”. We simply used Metasploit to accomplish this.

3.2.5 Arnek

We found this user when we used a bruteforce user attack with metasploit, since we knew the password and there only was 15 alternatives of usernames which it could match, we used a dictionary/list of usernames and only one password. We found the user really fast.

3.3 Compromising systems

The servers and network where located behind a firewall which routed and filtered the traffic approaching it so we needed to get access to the first server we could get access to when first attempting to connect to 172.16.0.40.

3.3.1 Ubuntuserver1 (192.168.1.10)

The first user we tested to access this server was "turidf" because she is claimed to be the cheif of IT in the company. We used the password we have acquired from earlier to try log on to the server. And we managed to do that. The check if the given user had root access we need to run a command which checks if the user has root access: **sudo -v** No output came, which meant that this user has access to run scripts as root.
We then ran the command: **sudo su**
Which switched us to root. We had at this point full access over the target computer and could perform any task we wanted to as root, even though we didn't have the password for that account.

3.3.1.1 Webservices - Company website

Everyone has access to read every page in the web but we wanted to get access to the backend. We therefore took in account that we had access to the server backend and then used the webform for users who have forgotten their password. Then we requested a new password for the “admin” user. Right after that we opened the local user “root”s maildir and found the email with the new password and after that we deleted the email. After we logged in on the website we noticed that we now have full access over the company website. Another way we could do since we allready had full root access on the server was to open **/var/www/data/_site/users.php** This file where all the usernames and password hashes are located. This is an example from the file:

```
array (
  'email' => 'fridae@nettlab.lan',
  'password' => 'b1d41a14e33db605f83befb5679ac1c0e021c97928fdbe2e07dc5c47a466f05bec2c8
  'granted' => 'all',
  'editing' => 'all',
  'passhash' => 'sha512',
  'attempts' => 0,
  'file_name' => 'gpsess_r9Gr1rx7SUTBE72BK5zN4KEh6H9eavLCn3Ted28j.php',
  'lastattempt' => 1446457301
)
```

3.3.1.2 MySQL Server

Since we have noticed that there is a **/phpmyadmin** directory we decided to test if we could gain access to the mysql server. Based on our earlier knowledge with mysql we know that debian based systems generates a plaintext username and password file for mysql which is located at: **/etc/mysql/debian.conf** Here is a username and password for a full privileged user located. We know also have full access over the mysql server on this server as-well. This is the content from that file:

```
# Automatically generated for Debian scripts. DO NOT TOUCH!
[client]
host     = localhost
user     = debian-sys-maint
password = c8Q4nAk3Ecf2rPFz
socket   = /var/run/mysqld/mysqld.sock
[mysql _ upgrade]
host     = localhost
user     = debian-sys-maint
password = c8Q4nAk3Ecf2rPFz
socket   = /var/run/mysqld/mysqld.sock
basedir  = /usr
```

With this information we now have a root account that we may access every database on the server. We also looked into MySQL to see if there was any interesting databases. But didn't find any in particular, this however is our result.

```
mysql> Show schemas;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| phpmyadmin         |
| test               |
| wordpress          |
+--------------------+
6 rows in set (0.00 sec)
```

3.3.1.3 Dovecot Mailserver

We wanted to extract as much information as possible from this service and we figured out that the public/private key would be nice to have in case we should do a MITM attack later. We therefore extracted the locations of the the public and private key by running: **cat /etc/dovecot/conf.d/* — grep pem** . The results where two paths where we could locate the key pair:

```
ssl_cert = </etc/ssl/certs/dovecot.pem
ssl_key = </etc/ssl/private/dovecot.pem
```

The public and private key is included in ”appendix_3_dovecot.pem” This is extremely useful if a hacker intends to do a MITM attack later with emails. With this information it becomes nearly impossible for regular users to detect a MITM attack.

3.3.1.4 vsFTPd

We also noticed that this server ran a vsFTP server. We inspected it to see if it contained any interesting information, but it was only mapping to the users home folder which we already had access to through the local file system.

3.3.2 Ubuntuserver2 (192.168.0.20)

To connect to this host we needed to connect through Ubuntuserver1 since this server was inaccessible from outside the company firewall. We used both ssh tunnel and meterpreter which we installed on Ubuntuserver1 to solve this. On this host we couldn't manage to get root access since the password for the only user which was in the sudoers group on this server was different from both the domain and the rest of the servers. But it did run some interesting services like samba file server which here also was mapped to the home folders for the logged in users on the server.

3.3.3 WindowsServer2K (192.168.0.30)

This server was completely locked down considering user access, we managed to find out through manual attempts that only the local administrator had access to logon to this computer. We therefore used the local administrator hash we found on the "windows7" machine to uploaded a meterpreter through smb using an exploit called psexec. The results where that after that had complete control over the domain controller, which meant that we could in a company network do whatever we wanted to any domain joined computer. This is the result from our hashdump:

host	origin	service	public	private	realm	private_type
192.168.0.30	administrator			aad3b435b51404eeaad3b435b51404ee:a5df61428f279c798c1e664866c3965a	nettlab.lan	NTLM hash
192.168.0.30	192.168.0.30	445/tcp (smb)	krbtgt	aad3b435b51404eeaad3b435b51404ee:6e24f5c43ee0be501a7a69cdb7f3f007		NTLM hash
192.168.0.30	192.168.0.30	445/tcp (smb)	frankh	aad3b435b51404eeaad3b435b51404ee:1098227979e0c22280eb9b130ec0ff6b		NTLM hash
192.168.0.30	192.168.0.30	445/tcp (smb)	turidf	aad3b435b51404eeaad3b435b51404ee:a5df61428f279c798c1e664866c3965a		NTLM hash
192.168.0.30	192.168.0.30	445/tcp (smb)	perj	aad3b435b51404eeaad3b435b51404ee:09c814737cded2114f210c8758db20cd		NTLM hash
192.168.0.30	192.168.0.30	445/tcp (smb)	arnek	aad3b435b51404eeaad3b435b51404ee:6e24f5c43ee0be501a7a69cdb7f3f007		NTLM hash
192.168.0.30	192.168.0.30	445/tcp (smb)	fridae	aad3b435b51404eeaad3b435b51404ee:3f9c1a90191194563cb1504f6f9c66df		NTLM hash
192.168.0.30	192.168.0.30	445/tcp (smb)	WINDOWSSERVER2K$	aad3b435b51404eeaad3b435b51404ee:da85f9a394107d1fb7299d60e434e452		NTLM hash
192.168.0.30	192.168.0.30	445/tcp (smb)	WINDOWS7$	aad3b435b51404eeaad3b435b51404ee:27308e631718b36edcc5e982d1850eca		NTLM hash

3.3.4 WindowsXP (192.168.0.100)

We never got access to this computer since it was not a domain joined computer and it didn't have any of the users locally enabled. We tried some usual password with "administrator" account but we didn't manage to get in to it. All other ports than 3389 (RDP) was filtered so we couldn't use any exploit we found either.

3.3.5 Windows7 (192.168.0.106)

This computer we got access to using username "fridae" and password "RaptorF150". This account was known since we had already found it for the "ubuntuserver2" which made this attempt very easy to complete. The user had also local administrator rights which meant that we could access any files on that computer. We also uploaded a meterpreter and ran it as local administrator which we also used to dump the hash for the local administrator for later use. At this point we had complete control of the given computer.

3.4 Scripts

3.4.1 Task 1

After running our script (see appendix_1_task1.py). We came up with the following solution:

Word 1	Bid
Word 2	Antidote
Word 3	Hedgehog Cactus
Word 4	Henna
Word 5	Xeronic
Word 6	Ineffective
Word 7	Sensor
Word 8	Spherical Harmonic Analysis
Word 9	Win32
Word 10	1st Baron Beaverbrook
Word 11	X-ray Machine
Word 12	Java 2 Platform, Micro Edition

The script used regular expressions to determine which parts of the website it should use. We also choose to try to develop an app that didn't need any out of the ordinary packages for python.

3.4.2 Task 2

To solve Task 2 we didn't use any custom scripts written by the group but rather various of techniques that together gave us direct access to the file. (See appendix_2_task2.py.) We used the meterpreter we got from **Windows7 (192.168.0.106)** and gave our self admin rights using and then dumped the local administrators password to our credentialsdatabase. After retrieving the admin has we used an exploit called psexec. To use it, we executed the following command: ”**use exploit/windows/smb/psexec**” This exploit executes an arbitrary payload on the server through SMB. It also cleans up and hides its tracks after use. The payload executed perfectly and gave us local administrator rights on the domain controller. After that we only needed to start a shell and search for the file. As described in the task, the target file was located in arnek's documents folder. More specific in: **C:\Users\arnek\Documents\VeryImportantDocument.txt.** The contents of the file was:

```
My bank account pin number is: 1313
```

3.5 Hiding our tracks

3.5.1 Wiping logs manually

An important part of our hacking projects was to hide our tracks. We had that in mind since the start and the first thing we did when we connected to a new server was to copy all the logs and prepare them and for example remove all log in attempts that we have failed with earlier. When the group was finished we removed and replaced the current logfiles on the server with the log files that we have prepared for the server. In that way, it may seem that no one ever had tried to login or ever had logged in to the server. We also wiped the bash history on the linux server and removed all meterpreter files that we had uploaded to the server using the command: **history -c && history -w**.

So after we cleaned out the **.bash_history** files and the metasploit payloads, we copied over all of the log files that can be found in **var/log/** to the machine we had running, so we could manually go over them and see that there was no events after 23.October to be found in the log files.

3.5.2 Wiping with meterpreter

On the windows computers we used the meterpreters option to wipe the logs. This was done using a command called **clearev**. This is an example on the output it gave us:

```
meterpreter > clearev

[*] Wiping 2016 records from Application...

[*] Wiping 10520 records from System...

[*] Wiping 5983 records from Security...
```

This command wiped the event log and automated the process with deleting log events on windows computers. We also removed and temporary files that we had used in the meanwhile on the servers.

Chapter 4

Discussions

4.1 Validating how we solved the task

In this attack we solved the task a way think was the easiest way of doing it. There are multiple ways of accomplish the same results, but there are different levels of difficulty to each of them. Most of these are only limited to experience and since the group had allot experience with running GNU/Linux servers we wanted to exploit as many weaknesses as possible through this. We also had less scripting experience than the other groups and therefore didn't focus that much on that in this project. Even though we clearly see how this would save us much time on a much grander scale, we could not see why we should use this in this project. Based on our early knowledge of the company, we knew that it wouldn't be many computers on the inside and not many of users.

4.2 Validating our results

When it comes to our result we wanted to validate if the data are of any use to us and if the security breaches shown in chapter 3 are a real threat to the company. In short, yes they are. We could easily have taken complete control over all the home shares and the domain controller, and if more computers had been added to the domain, we would easily have controlled them aswell. We could even inject meterpreters into the login script, making it even impossible to detect without a propper antivirus or firewall. We also managed to eject passwordhashes even though we knew most of the passwords through social engineering, this would have been useful if this was a large enterprise company with thousands of employees. We could have tried to crack the hashes using some kind of super-computer or a big cluster with shared workload but in this scenario it would have been useless since we did manage to get into all the systems with the domain controller. But in some companies where passwords are rather synced across services instead of authenticated when accessed it would have been useful.

4.3 Alternative ways to solve the tasks

4.3.1 Hacking the ubuntuserver1

We used a small bruteforce attack on the server with the information gathered on during social engineering. Instead of doing this we could have looked for a weaknes in the OS or SSH server so that we could have maybe gained access even without bruteforcing.

4.3.2 Hijacking company website

Instead of exploiting the root user's maildir we could have looked for an exploit for GPEasy to get admin access, or we could have extracted the hashes and then tried to crack them or at least reverse engineered the hashing algorithm on to the CMS if it was a weak one.

4.3.3 Hacking Domaincontroller

Instead of using an admin hash we retrived from the Windows7 computer we could have tried a bruteforce attack on RDP, it would have taken alot of time, but if we didn't find the hash it may have been our only solution unless another exploit was avaiable.

4.3.4 Scripting instead of hacking

Since we gained access to the domain controller and where able to browse files directly on the server we solved that task. But if we haven't had gained that access, then there would probably be no other solution than actually doing the PowerShell scripting.

4.3.5 Used meterpreters more

Meterpreters have many builtin functions that would have produced the same results as we did manually. We could have used them more instead of doing it manually.

4.3.6 Check permissions before executing commands

We made alot more work for ourself when we sometimes forgot to type sudo -v to check if the given user had sudo permissions, because if you try to execute a sudo command on a privileged command and you don't have access, a mail is sent to root on the server and then the attempt has been logged.

4.3.7 More silent scans

The group could have used more methods for scanning that didn't make that much noise on the network. We could for example have used nmap SynScan which would not complete the tcp connection, but would have just rested the connection if response, and therefore we would have known if the port was open or not. Since the group was out of options at some points we used more aggressive methods of scanning, which produced more noise, but not that much that someone could detect it unless they where actively listening.

4.3.8 Phising attack

Since this is a fictive company there was little point in trying this, but on a real company there would most likely have a positive result if the group had tried phishing on the employees. For example if we had sent an email with a promt to enter credentials on a rouge website.

4.4 Different subjects of interests

4.4.1 Persistent connections

In this task we didn't want this since the task was to get in and out without getting detected. If the task where to maintain information gathering over longer periods, then this would have been something to consider. However it is an important subject to discuss since the main concern is often about just getting the connection and not maintain it. For each new connection you have to make you are making more noise on the network, and increasing the chance of getting detected. However we did some research how to do this and on windows it is basically built into metasploit, but for linux you had to do some modding yourself. We figured out that if we wanted this we would save the meterpreter file someplace where it most likely would not be detected and then execute a code through cronscript each 10 minutes or so where it checks if the meterpreter has connection, and if not it tries to connect again.

4.4.2 Scripting vs. Manual work

The group also discussed how much we wanted to wight the scripting vs manual work. On the tasks that required us to script we scripted of course but on the rest of the project we could have scripted alot more than we actually did and we problaby would have saved much time with running the scripts rather than doing it manually. But for us, our main concern was to create the scripts vs the total time of doing it manually. Then again this is something that the hacker must evaluate for each company, if it is a large enterprise company then you should do it, if the work is repetitive. Here however, almost no to very little work was repetitive.

Chapter 5

Conclusion

Based on our results and our discussion around the project, the group concludes that the project was a success. Even though we didn't manage to break into every system, we did get a major understanding about how working as a hacker can be. We did solve the major tasks and the tasks given to us in the scripting part in another way though of by the subject. As described in the discussion subject, PowerShell is not our greatest strength, but there was still another way of solving it. The social engineering helped us with many problems we would encountered later regarding security measures, this just proves how important strong passwords are vs weak and easy passwords.

Chapter 6

Bibliography

This is the references we used to complete the tasks. These are mainly for referencing commands.

[1] - https://www.offensive-security.com/metasploit-unleashed/

Appendix A

Appendix

```
############ Dependencies ############
from time import sleep
import socket
import urllib2
import re

############ Global vars ############
SERVER_ADDR = ”172.16.0.200”
SERVER_PORT = 8085
BUFFER_SIZE = 1024

CORRECT_ANSWERS = []


############ START OF METHODS ############
def motd():
    print (”␣_____␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣___␣”)
    print (”|␣␣__␣\␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/␣␣␣|”)
    print (”|␣|␣␣\/_␣__␣___␣␣_␣␣␣_␣_␣__␣␣/␣/|␣|”)
    print (”|␣|␣__|␣'__/␣_␣\|␣|␣|␣|␣'_␣\/␣/_|␣|”)
    print (”|␣|_\␣|␣|␣|␣(_)␣|␣|_|␣|␣|_)␣\___␣␣|”)
    print (”␣\____|_|␣␣\___/␣\__,_|␣.__/␣␣␣␣|_/”)
    print ("                      | |          ")
    print ("                      |_|          ")
    print (" ")
    print ("Program: Python task 2 - Crossword puzzle ")
    print ("Authors: coral & Klaus Wunderlich\n")


def connect():
    # Create socket object
    """
    Initiates a connection socket and connects it to the server in global vars, returns the object

    :return:
    """
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Connection
    connection_string = (SERVER_ADDR, SERVER_PORT)
    s.connect(connection_string)

    # Return the object
    return s


def disconnect(socket):

    """
    Disconnects the given objects connection to the server and returns it
    :param socket:
    :return:
    """
    socket.close()

    return socket


def send_data(connection, number, initial=False):

    # Send the number
    """
    Send the given number to the given connection, if initial it will skip the first line default is false for that
    :param connection:
    :param number:
    :param initial:
    :return:
    """
    connection.send(number)

    # Read the first line
    if initial:
        connection.recv(BUFFER_SIZE)

    # Save the response
    response = connection.recv(BUFFER_SIZE)

    return response


def get_possible_solutions(string):
    """
    Trying to get a list of possible answers for the given string from the website
    :param string:
    :return:
    """
    print "Starting webscraper"
    # Add all undersites array
    under_sites = []

    # String regex
    string_splitted = string.split("?")
    string_regex = "(" + string.replace("?", "(.+)") + ")"

    # Declare the empty return array
    all_fetched_words_list = []

    # Declare the empty return array
    candidates_list = []

    # Determine the main url for the first letter
    if string[0].isalpha:
        # Retrieve the main url
        site_url = "http://www.webster-dictionary.org/Index-" + string[0]
        print "Trying to fetch from url: " + site_url

        # Read the main page
        site_main = urllib2.urlopen(site_url).read()

        # Retrieve the urls
        site_main_header_raw = re.compile('<div id="p_' + string[0] + '(.*?)</div>',
            re.DOTALL | re.IGNORECASE).findall(site_main)

        if len(site_main_header_raw) > 0:
            # If any undersites add to array
            site_main_list_raw = re.compile('<a href="(.*?)">(.*?)</a>', re.DOTALL |
                re.IGNORECASE).findall(site_main_header_raw[0])
            print "Found " + str(len(site_main_list_raw)) + " undersites"

            # Add each to under_sites list
            for urls in site_main_list_raw:
                under_sites.append("http://www.webster-dictionary.org" + urls[0])

        else:
            # If no undersites just add firstpage
            under_sites.append("http://www.webster-dictionary.org/Index-" + str(string[0]))

    else:
        under_sites.append("http://www.webster-dictionary.org/Index--")
        print "Trying␣to␣fetch␣from␣url:␣http://www.webster-dictionary.org/Index--"

    # Process the list
    for under_site in under_sites:
        # Get the website contents
        print "Scraping:␣" + under_site
        site_whole = urllib2.urlopen(under_site).read()

        # Lets extract only the given table that we need
        site_table = re.compile('<table(.*?)</table>', re.DOTALL | re.IGNORECASE).findall(site_whole)

        # Strip the contents we don't need
        site_list_raw = re.compile('<a␣href="(.*?)">(.*?)</a>', re.DOTALL | re.IGNORECASE).findall(site_table[0])

        # Extract only the word we want
        for item in site_list_raw:
            all_fetched_words_list.append(item[1])

    # Use regex to shorten the list down
    for candicate in all_fetched_words_list:
        if(re.match(string_regex, candicate, re.IGNORECASE)):
            candidates_list.append(candicate)

    # Return the final list
    return candidates_list


def single_number_auto(number):
    """
    Tries to solve the given number with automatching from web
    :param number:
    """
    string = send_data(s, str(number), True)

    print "Server responded: " + str(string)

    # Get possible solutions
    possible_keys = get_possible_solutions(string)
    print "Received a list of " + str(len(possible_keys)) + " possible answers"

    for item in possible_keys:
        print "Trying " + str(item)
        # Try the key
        response = send_data(s, str(number) + str(item))

        if response == "Correct!":
            print "Found correct answer: " + str(item)
            CORRECT_ANSWERS.append([number, item])

            break


def single_number_manual(number, solution):
    """
    Tries to match the given number with the given solution
    :param number:
    :param solution:
    """
    string = send_data(s, str(number), True)

    print "Server responded: " + str(string)

    response = send_data(s, str(number) + str(solution))

    if response == "Correct!":
        print "Found correct answer: " + str(solution)
        CORRECT_ANSWERS.append([number, solution])

    else:
        print str(solution) + " was not the right answer"


def start_task():
    # Start connection
    """
    Starts the main task and asks for which numbers to start and stop on

    """
    print "At which number would you like to start at: (ONLY_NUMBER)"
    start_location = raw_input("")

    print "At which number would you like to stop at: (ONLY_NUMBER)"
    stop_location = raw_input("")

    s = connect()
    initial = True

    # Start the for loop
    for attempt in range(int(start_location), int(stop_location), 1):
        print ""
        print ""
        # Print which solution
```

```python
        print "Starting attempt: " + str(attempt)

        # Assign a variable to indicate if the given attempt is found
        found = False

        # Get the string
        print "Connecting to the server..."
        string = send_data(s, str(attempt), initial)

        initial = False

        print "Server responded: " + str(string)

        # Get possible solutions
        possible_keys = get_possible_solutions(string)
        print "Received a list of " + str(len(possible_keys)) + " possible answers"

        for item in possible_keys:
            if(found == False):
                print "Trying " + str(item)
                # Try the key
                response = send_data(s, str(attempt) + str(item))

                if response == "Correct!":
                    print "Found correct answer: " + str(item)

                    # Add data to array
                    CORRECT_ANSWERS.append([attempt, item])

                    found = True

                    break
            else:
                break
```

```python


    # Print the result
    for i in CORRECT_ANSWERS:
        print "Word " + str(i[0]) + " was " + str(i[1])
############ END OF METHODS ############



############ MAIN ############
Run = True
while (Run == True):
    motd()
    print "Choose task (enter number and press enter)"
    print "[1] - Run through all numbers"
    print "[2] - Run through specific number"
    print "[3] - Manually test specific number"
    print "[4] - Print all the found solutions"
    print "[ELSE] - Exit program"

    selection = raw_input("")

    if selection=="1":
        # Start the main task
        start_task()

    elif selection=="2":
        # Promt for number
        print "Enter the number you would like to test"
        wanted_number = raw_input("")

        # Make connection to server
        print "Connecting to server..."
        s = connect()

        # Trying to send data
        single_number_auto(wanted_number)

        print "Disconnecting from server"
        disconnect(s)

    elif selection=="3":
        # Promt for number
        print "Enter the number you would like to test"
        wanted_number = raw_input("")

        print "Enter the string you would like to test without number infront"
        solution = raw_input("")

        # Make connection to server
        print "Connection to server..."
        s = connect()

        # Trying to send data
        single_number_manual(wanted_number, solution)

        print "Disconnecting from server"
        disconnect(s)

    elif selection=="4":
        print "Printing the found solutions: "

        # Print the result
        for i in CORRECT_ANSWERS:
            print "Word: " + str(i[0]) + ". Solution: " + str(i[1])

    else:
        Run = False
        print "Exiting... Byebye"

    raw_input("Press any button to continue")
```

Appendix B

Appendix

```python
__author__ = 'Anders,␣Bendik'

# Dependencies
from time import sleep
import socket
import urllib2
import re


def motd():
    print ("REMEMBER!!!,␣MUST␣RUN␣WITH␣PYTHON2!")
    sleep(5)
    print ("␣_____␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣___␣")
    print ("|␣␣__␣\␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/␣␣␣|")
    print ("|␣|␣␣\/_␣__␣___␣␣_␣␣␣_␣_␣__␣␣/␣/|␣|")
    print ("|␣|␣__|␣'__/␣_␣\|␣|␣|␣|␣'_␣\/␣/_|␣|")
    print ("|␣|_\␣|␣|␣|␣(_)␣|␣|_|␣|␣|_)␣\___␣␣|")
    print ("␣\____|_|␣␣\___/␣\__,_|␣.__/␣␣␣␣|_/")
    print ("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣|␣␣␣␣␣␣␣␣␣␣")
    print ("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|_|␣␣␣␣␣␣␣␣␣␣")
    print ("␣")
    print ("Authors:␣coral␣&␣klaus␣wunderlich\n")
```

```


def get_worst_passwords():
    # Url of website
    url = "http://www.whatsmypass.com/the-top-500-worst-passwords-of-all-time"

    # Return array
    Return_array = []

    # Get the website
    site_main = urllib2.urlopen(url).read()

    # Get the table to string
    site_main_table = re.compile('<table␣border="1">(.*?)</table>', re.DOTALL | re.
        IGNORECASE).findall(site_main)

    # Get the table rows
    site_main_table_rows = re.compile('<tr>(.*?)</tr>', re.DOTALL | re.IGNORECASE).
        findall(site_main_table[0])

    # Iterate over the rows
    for tr in site_main_table_rows:
        # If not first row
        if tr != site_main_table_rows[0]:
            # Get the table elements in each row
            tds = re.compile('<td>(.*?)</td>', re.DOTALL | re.IGNORECASE).findall(tr)

            # Iterate over each
            for td in tds:
                # If not first column
                if td != tds[0]:
                    Return_array.append(td)

    return Return_array
```

```


def write_to_file(Array):
    # Open a file for writing
     file = open("File", 'w')

    # Process each element
     for i in Array:
         file.write(str(i) + "\n")

    # Close file
     file.close()


# Run the script
write_to_file(get_worst_passwords())
```

Appendix C

Appendix

```
—–BEGIN CERTIFICATE—–
MIIDtTCCAp2gAwIBAgIJALCxHShZiAsMMA0GCSqGSIb3DQEBBQUAMHExHDAaBgNV
BAoME0RvdmVjb3QgbWFpbCBzZXJ2ZXIxFjAUBgNVBAsMDXVidW50dXNlcnZlcjEx
FjAUBgNVBAMMDXVidW50dXNlcnZlcjExITAfBgkqhkiG9w0BCQEWEnJvb3RAdWJ1
bnR1c2VydmVyMTAeFw0xMzAzMjUxNjA3MDlaFw0yMzAzMjUxNjA3MDlaMHExHDAa
BgNVBAoME0RvdmVjb3QgbWFpbCBzZXJ2ZXIxFjAUBgNVBAsMDXVidW50dXNlcnZl
cjExFjAUBgNVBAMMDXVidW50dXNlcnZlcjExITAfBgkqhkiG9w0BCQEWEnJvb3RA
dWJ1bnR1c2VydmVyMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAL+H
JQ7sYRDnU6wh1b8Z7uXwkir9YMJYMLz6WetUce5HS8qluW2AcI0FFe2Pdah+65Yw
015QfS8l5CrbSJGAoBNlk1VGKP99NV5oyUans4IGREgv4a4edhySbJL5veXlpq7k
K3g2hYKsZJCbkEKFE98SSt3YHuzHRx0zucGwGPpdOwYu2XHZLD1TMzr3xqZ6Bjlw
+esme0WFKB30HtLqV1evQeXNnwANWxBoImXoaS5LrW3uTCv/gK1byC69SB5BGFW/
BSsc0706xNvhBYqfo1ZvMqE9F/K7sZZia1NEAUdrjnH5jcqno8nY9NUvc8yspr6w
iTUqPqC8eyMiFcDt6mMCAwEAAaNQME4wHQYDVR0OBBYEFDtT8VeAZomj+EwfiJe5
6UzwsdJfMB8GA1UdIwQYMBaAFDtT8VeAZomj+EwfiJe56UzwsdJfMAwGA1UdEwQF
MAMBAf8wDQYJKoZIhvcNAQEFBQADggEBALMpREVoBQprrvWaDUZWiXjZO835OU4E
XfU1vALcTyXXjcvxn5JnAVifC4sCapKuktAsvTW/mmK9EG0OpoIP8Ek9YzaTEPJI
8ISQN0igf9plH428A8PBcXCnMNFArH7W2BlagdFPjTxZubgHceXWSM6MyTfmH68v
VlxQixX8HGk+RD1CZXWanHeb7zUb8UwOtev+XqQ5uiom6ucsMVucS4C/DcaPNn6K
y2gvjdO0fItnwfzK8Lyx+IBOkUdoU81SImPovK/zdy93JMQQfbUgnvdvkVu+//zz
2Bsr2EsoukaFFbR23NiP5y51REhw4IpHSMnexJH/6Ar7g+3/l74tJT0=
—–END CERTIFICATE—–
```

```
———BEGIN PRIVATE KEY———
MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKkwggSlAgEAAoIBAQC/hyUO7GEQ51Os
IdW/Ge7l8JIq/WDCWDC8+lnrVHHuR0vKpbltgHCNBRXtj3WofuuWMNNeUH0vJeQq
20iRgKATZZNVRij/fTVeaMlGp7OCBkRIL+GuHnYckmyS+b3l5aau5Ct4NoWCrGSQ
m5BChRPfEkrd2B7sx0cdM7nBsBj6XTsGLtlx2Sw9UzM698amegY5cPnrJntFhSgd
9B7S6ldXr0HlzZ8ADVsQaCJl6GkuS61t7kwr/4CtW8guvUgeQRhVvwUrHNO9OsTb
4QWKn6NWbzKhPRfyu7GWYmtTRAFHa45x+Y3Kp6PJ2PTVL3PMrKa+sIk1Kj6gvHsj
IhXA7epjAgMBAAECggEBALPKmLYenavxi7FNfcpL/OArafsjKb6mbPIpjVmFRwIo
OXS9B4lMhdtLh98eyZNnuZ2erfyeUDV/O2YjvUahWSN8OcdV+kyD7CJ4+WDzMmZ/
nX+qOw9plvSRW4zJFkOGxmbs2AWcIM6fKDQ7ZDudkx7XnCIOrFEXbKF0Gg9BUOFM
HMP/q9lIDPDR6cAgVv99s4X8Zs/WVGcfy2T19fQ82lUNfMsO/NV3M9jsjxgVGaPe
ad0dLNh9j9v3ofvgVH/nJwd8lkdAoZXnKPco4Vz7KgP/rIPzicuUy/0vESvlc3I0
4XBX4bHpZJpSL/cXkxos9F4Y4wdZFKB2gNzdW23tPTECgYEA7a2nTM3SYeYN6GC4
1mQ6b/+dTF8fsYbonu7bcsBYdkL8lrWCj7Wm3lA7ZAIGiMxfcV0qnwk05NTnG2sB
hn6QaucIHKzEDPu1+rz9ti7UOzjpUmVl1d9WeCzQTyEyGjkElXf4extMM9G892ge
C1dOvNNt34veRtkn7zripbLR1/kCgYEAzkrCIZC/iZ5RvHG7txSo5M2B6myPH73n
OyePGpe5aYfQRrGgtbLSI9FBNmVc5ygz7LBTHdWlRwFTKCEFiWN81WbxaDNqsfKj
apdSS3oNbqKdjhf2gnci7hya9aFCHNH0n/J6vMhNl0skdO4JklzS77YM4hPfWUP3
OkFHjhGLRDsCgYAQpW7oNCod7SzgL5YCffaRzYdIyAjCOD6mkvgPq2UGs15Zd9Dz
G7faLihassGeZyHwIKRRiyWHOVoOU4pBzy9yLUgmKft5JU/zhbUHQ3RdyXid6rgd
KI030XutbxfBOmkVxtdCWAEYSAJCGaqxBKZhFzXEyFkDAUaIvMkO7d1AkQKBgQCh
4+s7eiPGNReB4mNlBvKQNHK67JzZVSestZvJS753Ad1CTU2Iqh/dee9KZ/ujcFpn
aeygYjSFZ5XBzUBSFGA3/MSAeLFHmtoB4WT01IDMDRPGGiobMrGX2Z0GHJoMkfv+
tcT6d9rkW1Tt6oxabzweYGj11pCVtg/DiM1FwaDgVwKBgQDqI92mSuAAf3WDcK8g
8zuiR7QmIe27tftt80GbpeKW/x9R1CEHmALAbpHCd37FEmnMee/YR5I3VHuIO8gD
2gIobERFY9OLYjN2+Q4WdWh1TAMVuKp9KvVlUDHhgUlk9jqX400v2UUeU8/h2ocn
EhMnWafpqmUylU7+pWxtl3zDtw==
———END PRIVATE KEY———
```

Appendix D

Appendix

```
############ Dependencies ############
from time import sleep
import socket
import urllib2
import re

############ Global vars ############
GROUP_NUMBER = 4

# RESULTS
EMPLOYEES_INFO = []

############ START OF METHODS ############
def motd():
    print ("␣_____␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣___␣")
    print ("|␣␣__␣\␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/␣␣␣|")
    print ("|␣|␣␣\/_␣__␣___␣␣_␣␣␣_␣_␣__␣␣/␣/|␣|")
    print ("|␣|␣__|␣'__/␣_␣\|␣|␣|␣|␣'_␣\/␣/_|␣|")
    print ("|␣|_\␣|␣|␣|␣(_)␣|␣|_|␣|␣|_)␣\___␣␣|")
    print ("␣\____|_|␣␣\___/␣\__,_|␣.__/␣␣␣␣|_/")
    print ("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣|␣␣␣␣␣␣␣␣␣␣")
    print ("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|_|␣␣␣␣␣␣␣␣␣␣")
    print ("␣")
    print ("Program:␣Python␣company␣scraper")
    print ("Authors:␣coral␣&␣Klaus␣Wunderlich\n")


def get_employees():
    print "----------␣Starting␣webscraper␣for␣employees␣----------"

    # Read the main page
    print "[ACTION]␣Opening␣main␣site␣for␣information"
    site_main = urllib2.urlopen("http://172.16.0." + str(GROUP_NUMBER) + "0/index.php").read()

    # Retrieve the urls
    site_main_header_raw = re.compile('<ul␣class="dropdown-menu">(.*?)</ul>', re.DOTALL | re.IGNORECASE).findall(site_main)

    # If found any extract names
    if len(site_main_header_raw) > 0:
        # Fetch the names
        print "[TASK]␣Looking␣for␣employees"
        site_main_header_fetched = re.compile('<li><a␣href="(.*?)"␣title="(.*?)">(.*?)</a></li>', re.DOTALL | re.IGNORECASE).findall(site_main_header_raw[0])

        # Print info about how many
        print "[UPDATE]␣Found␣" + str(len(site_main_header_fetched)) + "␣employees."

        # Process each to get email
        for employee in site_main_header_fetched:
            # Open employee website
            print "[ACTION]␣Opening␣site␣for␣" + str(employee[2]) + "␣-␣http://172.16.0." + str(GROUP_NUMBER) + "0" + str(employee[0])
            employee_site = urllib2.urlopen("http://172.16.0." + str(GROUP_NUMBER) +
                "0" + str(employee[0])).read()

            # Fetch info
            employee_info = re.compile('(\w*)@(\w*).lan', re.DOTALL | re.IGNORECASE).findall(employee_site)
            employee_mail = re.match('(\w*)@(\w*).lan', employee_site)

            print "[UPDATE] Found " + str(len(employee_info)) + " interesting elements"

            # Append the information to the user
            EMPLOYEES_INFO.append([employee[2], employee_info[0], employee_mail])

        print "[UPDATE] Data loaded into RAM and is ready for later use"
    else:
        print "[ERROR] Didn't find any employee info.. Sorry"

    print "---------- Stopping webscraper for employees ----------"


def get_words_on_site():
    print "---------- Starting webscraper for employees ----------"

    # Read the main page
    print "[ACTION] Opening main site for information"
    site_main = urllib2.urlopen("http://172.16.0." + str(GROUP_NUMBER) + "0/index.php").read()

    # Retrieve the urls
    site_main_header_raw = re.compile('>(.*)<', re.DOTALL | re.IGNORECASE).findall(site_main)


#### Main
```

```
while 1:
    motd()

    print "Please select what you want to do?"
    print "[1] - Gather data"
    print "[2] - Look at gathered data"
    print "[3] - Export gathered data"
    main_menu_selection = raw_input("")

    if main_menu_selection == "1":
        # Tell user what they have selected
        print "[1] - Gather data [SELECTED]"
        print ""

        # Ask what to do
        print "Please select what you want to do?"
        print "[1] - Get employees data"
        print "[2] - Get all words on site"
        print "[Any] - Go back"
        sub_menu_1_selection = raw_input("")

        if sub_menu_1_selection == "1":
            # Run get Employees method
            get_employees()

    elif main_menu_selection == "2":
        # Tell user what they have selected
        print "[2] - Look at gathered data [SELECTED]"
        for employee in EMPLOYEES_INFO:
            print employee
            print "----------------------------------------"
            print "Name: " + employee[0]
            print "Username: " + employee[1]
            print "Mail: " + employee[2]
```