university of new south wales

/sec/soc

Wargame Writeup: 01

OverTheWire Bandit - How2Linux

March 12, 2016

Contents

1 Disclaimer

Please be responsible with this writeup and use your newfound powers for good and not evil. ;)
If you appreciate this resource, please consider supporting us by coming along to our events!

Note that like with many problems, there are also multiple ways of solving the challenges in this wargame. This writeup will likely only cover a single way to solve a problem, but I may add alternatives if I'm feeling spicy.

As this wargame is intended for beginners, I'll try to keep my solutions as simple and easy to understand as possible. If you are a more experienced hax0r who knows how to drive Linux, you might want to skip this writeup to avoid having an aneurysm.[1]

Learning how to find information and read documentation is an invaluable skill, Google and man pages are your friends. As such, **it is highly encouraged that you attempt to do all of the challenges before you read this writeup** - getting spoonfed won't help you learn.
Kthnx.

Happy hacking!

2 Precursor

The goal of this wargame is to complete the final level, level 26. Each level has a password in it - this password is used to progress to the next level. Thus to complete this wargame, you must complete level 26 and every level preceding it.

Levels are accessible via ssh, where user *bandit1 is level 1, bandit2 is level 2, etc.* You begin the game at level 0.

Before starting, we've got to first log into the game using SSH. The host to which you need to connect is **bandit.labs.overthewire.org**. The username is **bandit0** and the password is **bandit0**.
To achieve this, log in with ssh:

```
crye@Rapier:~| ssh bandit0@bandit.labs.overthewire.org
```

When prompted for a password simply enter bandit0, as provided for us above.
Now we're ready to begin the game and hack some planets.

Additional Reading

- Secure Shell (SSH) on Wikipedia.
- How to use SSH on wikiHow.

[1] UNSW SecSoc will not be held accountable for any aneurysms caused by this writeup.

3 Solutions

3.1 Level 0

Goal The password for the next level is stored in a file called '**readme**' located in the home directory.

Solution After we've ssh'd into the server, by default our current working directory will be the home folder of the user. Conveniently this is also where the password file is stored. We can use 'ls' to list the directory contents so we can double check that the readme file is there.

```
bandit0@melinda:~$ ls
readme
```

To get to the next level, print out the contents of readme to extract the password.

```
bandit0@melinda:~$ cat readme
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

3.2 Level 1

Goal The password for the next level is stored in a file called '**-**' located in the home directory.

Solution Similarly to the last level, we're required to print the contents of a file to extract the password for the next level. However, in this case we're unable to cat the file as we did in the prior level.

```
bandit1@melinda:~$ cat -
IsThereAnEchoInHere?
IsThereAnEchoInHere?
```

This is because '-' is an alias for standard input, so by typing 'cat -', we're telling cat to output whatever comes in through stdin. (In this case, you can see I've typed 'IsThereAnEchoInHere?' and it's been printed back out via cat.)

In order to allow cat to understand that we're talking about a file rather than stdin, we need to specify the full file path.

```
bandit1@melinda:~$ cat /home/bandit1/-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Alternatively, to quickly reference the home directory, we can use '∼'. In Bash, '∼' expands to the current user's home directory, so the following is equivalent to the above.

```
bandit1@melinda:~$ cat ~/-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Another useful thing to know is that you can quickly reference the current working directory (where your shell is right now) with '.'. As our shell was located in the home directory after we ssh'd in, we could have also typed.

```
bandit1@melinda:~$ cat ./-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Keep these in mind as they're *incredibly* useful and will save you *a lot* of time. I will be using them extensibly.

Additional Reading

- Google Search for "dashed filename".
- Advanced Bash-scripting Guide - Chapter 3 - Special Characters.

3.3 Level 2

Goal The password for the next level is stored in a file called '**spaces in this filename**' located in the home directory.

Solution Yet again we're required to print the contents of a file to extract the password for the next level. Similarly to previous levels, there's another spicy twist which prevents us from simply typing.

```
bandit2@melinda:~$ cat spaces in this filename
cat: spaces: No such file or directory
cat: in: No such file or directory
cat: this: No such file or directory
cat: filename: No such file or directory
```

This is because command arguments are delimited by spaces in Linux. As cat will print any argument given to it to stdout, cat is interpretting each word in 'cat spaces in this filename' as a different argument, thus is trying to print files 'spaces', 'in', 'this', 'filename'.

In order to allow cat to know that we're referring to a file with spaces in it rather than multiple files, we need to 'escape' the spaces by placing a backslash in front of them. This explicitly outlines that you are referring to a space character rather than delimiting your arguments.

```
bandit2@melinda:~$ cat spaces\ in\ this\ filename
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Alternatively, we could wrap the file with quotes to ensure it's processed as a single argument.

```
bandit2@melinda:~$ cat "spaces in this filename"
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Additional Reading

- Google Search for "spaces in filename".
- Important Bash Tricks.

3.4 Level 3

Goal The password for the next level is stored in a hidden file in the '**inhere**' directory.

Solution To complete this we need to navigate to the 'inhere' directory from the home directory and then cat the file. However, as the file is hidden, we have to add an additional argument to ls to discover it's filename.

```
bandit3@melinda:~$ ls
inhere
bandit3@melinda:~$ cd inhere/
bandit3@melinda:~/inhere$ ls -a
.  ..  .hidden
bandit3@melinda:~/inhere$ cat .hidden
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Note: The '-a' argument in the above 'ls -a' tells ls to not ignore hidden files.
Additionally, in most Linux distros, hidden files are prefixed with a '.'. (However it is possible to have hidden files that are not!)

Additional Reading

- More on hidden files.
- Man page for ls.

3.5 Level 4

Goal The password for the next level is stored in the only human-readable file in the '**inhere**' directory.

Solution This level can be completed by navigating to the 'inhere' directory and running 'file' on every file in the directory to determine what type they are. As there is only one file that is ASCII text, it is clear that it's our password.

```
bandit4@melinda:~$ cd inhere/
bandit4@melinda:~/inhere$ file ./*
./-file00: data
./-file01: data
./-file02: data
./-file03: data
./-file04: data
./-file05: data
./-file06: data
./-file07: ASCII text
./-file08: data
./-file09: data
bandit4@melinda:~/inhere$ cat ./-file07
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Note: '*' in Bash is a special 'wildcard' character which means 'anything'. This means that the './*' in the above 'file ./*' refers to every file in the current working directory.

3.6 Level 5

Goal The password for the next level is stored in a file somewhere under the '**inhere**' directory and has all of the following properties: - human-readable - 1033 bytes in size - not executable.

Solution Here we can see there are a bunch of junk folders to obfuscate where the file is stored. Rather than manually checking them, run a find command on files of size 1033.

```
bandit5@melinda:~$ cd inhere/
bandit5@melinda:~/inhere$ ls
maybehere00  maybehere02  maybehere04  maybehere06  maybehere08
   maybehere10  maybehere12  maybehere14  maybehere16  maybehere18
maybehere01  maybehere03  maybehere05  maybehere07  maybehere09
   maybehere11  maybehere13  maybehere15  maybehere17  maybehere19
bandit5@melinda:~/inhere$ find ./* -size 1033c -readable ! -perm /111
./maybehere07/.file2
bandit5@melinda:~/inhere$ cat maybehere07/.file2
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Note: If you didn't understand what all of the flags in the above find command meant, now would be a great time to practice using man. You can view a manual page for find by typing 'man find'. If for example you wanted to find out more about the -size argument because you didn't understand what the '1033c' meant, you could search for it by typing '/-size' whilst in the man page. You can press q to exit a man page.

3.7 Level 6

Goal The password for the next level is stored **somewhere on the server** and has all of the following properties: - owned by user bandit7 - owned by group bandit6 - 33 bytes in size.

Solution This level is an extension on the previous level in that we have a couple more attributes to satisfy and we are no longer artificially confined to the home folder. Running a quick find command finds the password file quite nicely.

```
bandit6@melinda:~$ find / -size 33c -group bandit6 -user bandit7 2>/
   dev/null
/var/lib/dpkg/info/bandit7.password
bandit6@melinda:~$ cat /var/lib/dpkg/info/bandit7.password
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Note: The '2>/dev/null' part of the find command is an example of something called 'piping'. In this particular instance, I am redirecting or 'piping' the output of standard error (stderr) to /dev/null, which is a special device that discards information given to it. If I did not pipe stderr to /dev/null, the screen would be full of permission errors as the find command attempted to look in locations I did not have permission to access.

This is because I'm running find from the 'root' ('/') of the filesystem. If you're a Windows user, it is similar to running find on C:/ - you're going to be trying to poke around in a lot of places you're not authenticated to access as a regular user.

3.8 Level 7

Goal The password for the next level is stored in the file **'data.txt'** next to the word **'millionth'**.

Solution This level can be quickly solved by running a grep for 'millionth' on data.txt.

```
bandit7@melinda:~$ grep "millionth" data.txt
millionth XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Additional Reading

- Using grep in shell.
- Interative regex website with explanation.

3.9 Level 8

Goal The password for the next level is stored in the file **'data.txt'** and is **the only line of text that occurs only once**.

Solution To find the only line of text that occurs only once, sort the file alphabetically and pipe it into uniq -u to find lines that appear only once.

```
bandit8@melinda:~$ sort data.txt | uniq -u
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Note: Due to how uniq works, it is imperative that it receives sorted data, otherwise it won't function correctly! Think about it as uniq 'collapsing' down the same lines on top of each other - they need to be grouped in order to squash them together without having anything else inbetween them.

You may have noticed I used the words 'pipe' in the solution, even though there's no '>' sign. I am still using piping, it's just in a different context. In this example, I'm piping the output of one command to the input of another. (Forming a pipeline or chain between commands.) If this is unfamiliar to you, I highly suggest practicing it, as it is an incredibly powerful tool and will be widely used.

Additional Reading

- The unix commandline: pipes and redirects.

3.10 Level 9

Goal The password for the next level is stored in the file **'data.txt'** in one of the few human-readable strings, beginning with several '=' characters.

Solution As we're going to be searching a file consisting of both binary data and strings, run strings on the file to find only the strings (stripping out the irrelevant binary) then run a grep for lines that start one or more '='.

```
bandit9@melinda:~$ strings data.txt | grep -E "^=+"
========== password
========== ism
========== XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Note: If you're unfamiliar with regular expressions (regex), you might be a little confused about the search pattern we're using for grep. In regex, a '^' denotes that the following pattern is at the start of the line, and the suffix '+' means that there are one or more occurences of that pattern. In this case, the regex '^=+' translates to something like 'match any line beginning with one or more '=' characters'. I also highly recommend practise with regex, as it's a very powerful tool.

3.11 Level 10

Goal The password for the next level is stored in the file **'data.txt'**, which contains base64 encoded data.

Solution As the password is base64 encoded, we simply decode it with base64.

```
bandit10@melinda:~$ base64 -d data.txt
The password is XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Note: Base64 is an encoding scheme commonly used in web applications, you'll likely become very familiar with it.

Additional Reading

- Base64 on Wikipedia.
- What is Base64 used for?

3.12 Level 11

Goal The password for the next level is stored in the file **'data.txt'**, where all lowercase (a-z) and uppercase (A-Z) letters have been rotated by 13 positions.

Solution This is an example of the infamous 'rot13' substitution cipher, where each character is replaced with the character 13 places after it. (Place is given by n + 13 % 26). To reverse it we just transpose each character in the set a-z by 13.

```
bandit11@melinda:~$ cat data.txt | tr a-zA-Z n-za-mN-ZA-M
The password is XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Additional Reading

- Man page for tr.

3.13 Level 12

Goal The password for the next level is stored in the file **'data.txt'**, which is a hexdump of a file that has been repeatedly compressed.

Solution This level requires you to reverse a hexdump and repeatedly uncompress a file which has been compressed many times. We can unravel the file by repeatedly running file and the appropriate unzipping commands until we hit the password.

```
bandit12@melinda:~$ cd /tmp
bandit12@melinda:/tmp$ cp ~/data.txt .
bandit12@melinda:/tmp$ xxd -r data.txt > data2.file
bandit12@melinda:/tmp$ file data2.file
data2.file: gzip compressed data, was "data2.bin", from Unix, last
   modified: Fri Nov 14 10:32:20 2014, max compression
bandit12@melinda:/tmp$ zcat -d data2.file > data3.file
bandit12@melinda:/tmp$ file data3.file
data3.file: bzip2 compressed data, block size = 900k
bandit12@melinda:/tmp$ bzip2 -d data3.file
bzip2: Can't guess original name for data3.file -- using data3.file.
   out
bandit12@melinda:/tmp$ file data3.file.out
data3.file.out: gzip compressed data, was "data4.bin", from Unix, last
    modified: Fri Nov 14 10:32:20 2014, max compression
bandit12@melinda:/tmp$ zcat -dd data3.file.out > data4
bandit12@melinda:/tmp$ file data4
data4: POSIX tar archive (GNU)
bandit12@melinda:/tmp$ tar xvf data4
data5.bin
bandit12@melinda:/tmp$ file data5.bin
data5.bin: POSIX tar archive (GNU)
bandit12@melinda:/tmp$ tar xvf data5.bin
data6.bin
bandit12@melinda:/tmp$ file data6.bin
data6.bin: bzip2 compressed data, block size = 900k
bandit12@melinda:/tmp$ bzip2 -d data6.bin
bzip2: Can't guess original name for data6.bin -- using data6.bin.out
bandit12@melinda:/tmp$ file data6.bin.out
data6.bin.out: POSIX tar archive (GNU)
bandit12@melinda:/tmp$ tar xvf data6.bin.out
data8.bin
bandit12@melinda:/tmp$ file data8.bin
data8.bin: gzip compressed data, was "data9.bin", from Unix, last
   modified: Fri Nov 14 10:32:20 2014, max compression
bandit12@melinda:/tmp$ zcat -d data8.bin
The password is XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Additional Reading

- Hex dump on Wikiapedia.

3.14 Level 13

Goal The password for the next level is stored in **'/etc/bandit_pass/bandit14' and can only be read by user bandit14**. For this level, you dont get the next password, but you get a private SSH key that can be used to log into the next level.

Solution For this level, we're given a private SSH key which can be used to authenticate as bandit14. Simply ssh into the localhost as bandit14 using the key, then cat the password file.

```
bandit13@melinda:~$ ls
sshkey.private
bandit13@melinda:~$ cat sshkey.private
-----BEGIN RSA PRIVATE KEY-----
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
-----END RSA PRIVATE KEY-----
bandit13@melinda:~$ ssh -i sshkey.private bandit14@localhost
bandit14@melinda:~$ cat /etc/bandit_pass/bandit14
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Additional Reading

- SSH/OpenSSH/Keys.

3.15 Level 14

Goal The password for the next level can be retrieved by submitting the password of the current level to **port 30000 on localhost**.

Solution Pretty straightforward. Use netcat to connect to the localhost (127.0.0.1), port 30000 and send the key for current level using it.

```
bandit14@melinda:~$ cat /etc/bandit_pass/bandit14 | nc -v 127.0.0.1
   30000
Connection to 127.0.0.1 30000 port [tcp/*] succeeded!
Correct!
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Note: It's important to understand this question. Networking is a big part of computing - learn how to use netcat properly as it is your bread and butter. In a lot of cases it's your stock standard method of popping a reverse shell on a target machine.

Additional Reading

- TL;DR How Do I Internet.
- IP address on Wikiapedia.
- IP addresses.
- Localhost on Wikiapedia.
- Ports.
- Ports on Wikiapedia.
- Netcat tutorial for beginners.

3.16 Level 15

Goal The password for the next level can be retrieved by submitting the password of the current level to port 30001 on localhost using SSL encryption.

Solution This is pretty much identical to the above, except we're now using SSL! Groovy.

```
bandit15@melinda:~$ cat /etc/bandit_pass/bandit15 | openssl s_client -
   connect 127.0.0.1:30001 -quiet
depth=0 CN = li190-250.members.linode.com
verify error:num=18:self signed certificate
verify return:1
depth=0 CN = li190-250.members.linode.com
verify return:1
Correct!
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

read:errno=0
```

Additional Reading

- Secure Socket Layer on Wikiapedia.
- OpenSSL man page.

3.17 Level 16

Goal The credentials for the next level can be retrieved by submitting the password of the current level to **a port on localhost in the range 31000 to 32000**. There is only 1 server that will give the next credentials, the others will simply send back to you whatever you send to it.

Solution First find out which of these ports have a server listening on them. Then find out which of those speak SSL and which dont. To do this we can use an nmap service scan on the port range 31000-32000 on the localhost.

```
bandit15@melinda:~$ nmap -sV -p 31000-32000 127.0.0.1

Starting Nmap 6.40 ( http://nmap.org ) at 2016-03-11 12:16 UTC
Stats: 0:00:36 elapsed; 0 hosts completed (1 up), 1 undergoing Service
    Scan
Service scan Timing: About 40.00% done; ETC: 12:17 (0:00:54 remaining)
Stats: 0:00:41 elapsed; 0 hosts completed (1 up), 1 undergoing Service
    Scan
Service scan Timing: About 40.00% done; ETC: 12:18 (0:01:02 remaining)
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00094s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE VERSION
31046/tcp open  echo
31518/tcp open  msdtc   Microsoft Distributed Transaction Coordinator
   (error)
31691/tcp open  echo
31790/tcp open  msdtc   Microsoft Distributed Transaction Coordinator
   (error)
31960/tcp open  echo
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at
   http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 41.32 seconds

bandit16@melinda:~$ cat /etc/bandit_pass/bandit16 | openssl s_client -
   connect 127.0.0.1:31518 -quiet
depth=0 CN = li190-250.members.linode.com
verify error:num=18:self signed certificate
verify return:1
depth=0 CN = li190-250.members.linode.com
verify return:1
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
^C
bandit16@melinda:~$ cat /etc/bandit_pass/bandit16 | openssl s_client -
   connect 127.0.0.1:31790 -quiet
depth=0 CN = li190-250.members.linode.com
verify error:num=18:self signed certificate
verify return:1
depth=0 CN = li190-250.members.linode.com
```

```
verify return:1
Correct!
-----BEGIN RSA PRIVATE KEY-----
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
-----END RSA PRIVATE KEY-----

read:errno=0
bandit16@melinda:~$ vi /tmp/HeresAPrivateKeyIPreparedEarlier
bandit16@melinda:~$ chmod 600 /tmp/HeresAPrivateKeyIPreparedEarlier
bandit16@melinda:~$ ssh -i /tmp/HeresAPrivateKeyIPreparedEarlier
   bandit17@127.0.0.1
bandit17@melinda:~$
```

From our portscan, 31046, 31691 and 31860 are all out, as they are clearly echo. This leaves either 31518 and 31960. Choosing the first and sending the current user password results in nothing being sent back, but the second returns an RSA private key. Saving it locally and changing it to have valid permissions (ssh private keys must NOT be accessible by others), we can use it to ssh into the next level.
Note: It's important to understand how to use nmap, as it's a very powerful and useful tool!

Additional Reading

- Port scanner on Wikiapedia.
- Using nmap.

3.18 Level 17

Goal There are 2 files in the homedirectory: **'passwords.old'** and **'passwords.new'**. The password for the next level is in passwords.new and **is the only line that has been changed between passwords.old and passwords.new**.

Solution To find the changes between the files, run diff on them and observe the additions and subtractions.

```
bandit17@melinda:~$ diff passwords.old passwords.new
42c42
< XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
---
> XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Note: If you're trying to log into level 18 with this password and you get kicked out with a message 'Byebye!', don't sweat - it's part of level 19.

Additional Reading

- How to read diffs.
- Diff on Wikiapedia.

3.19 Level 18

Goal The password for the next level is stored in a file **'readme'** in the home directory. Unfortunately, someone has modified **.bashrc** to log you out when you log in with SSH.

Solution As the bashrc file has been modified to log you out, we can bypass this roadblock by simply getting ssh to run 'cat readme' as soon as it logs on, before bash is even started.

```
crye@Rapier:~| ssh bandit18@bandit.labs.overthewire.org cat readme
bandit18@bandit.labs.overthewire.org's password:
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Alternatively, we could also pop a shell by simply running /bin/sh on login.

```
crye@Rapier:~| ssh bandit18@bandit.labs.overthewire.org /bin/sh
bandit18@bandit.labs.overthewire.org's password:
ls
readme
cat readme
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Somewhat humorously, if you're lucky you can ctrl-c before the part of the .bashrc that logs you out for the same effect. It's very finnicky on the timing, though.

Additional Reading

- What is a .bashrc file and how does it work?

3.20 Level 19

Goal To gain access to the next level, you should use the setuid binary in the homedirectory. Execute it without arguments to find out how to use it. The password for this level can be found in the usual place (/etc/bandit_pass), after you have used to setuid binary.

Solution The setuid binary allows us to run commands as bandit20, as such, we have access to bandit20s password file and can cat it to progress to the next level.

```
bandit19@melinda:~$ ./bandit20-do cat /etc/bandit_pass/bandit20
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Additional Reading

- Setuid on Wikiapedia.

3.21 Level 20

Goal There is a setuid binary in the homedirectory that does the following: it makes a connection to localhost on the port you specify as a commandline argument. It then reads a line of text from the connection and compares it to the password in the previous level (bandit20). If the password is correct, it will transmit the password for the next level (bandit21).

Solution For this level, we need to connect to the level twice. Once to start a network daemon to which the setuid will connect and again to run the setuid binary.

Connection 1: Set up daemon listening on port 1337 and send old password once setuid binary connects.

```
bandit20@melinda:~$ nc -v -l -p 1337
  Listening on [0.0.0.0] (family 0, port 1337)
  Connection from [127.0.0.1] port 1337 [tcp/*] accepted (family 2,
     sport 42006)
  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
```

Connection 2: Run setuid binary

```
  bandit20@melinda:~$ ./suconnect 1337
  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

  Read: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  Password matches, sending next password
```

3.22 Level 21

Goal A program is running automatically at regular intervals from cron, the time-based job scheduler. Look in **/etc/cron.d/** for the configuration and see what command is being executed.

Solution First navigate to /etc/cron.d and list the cron jobs. Inspecting them reveals a cronjob called cronjob_bandit22 - which is interesting. Printing the file reveals that the cronjob is executing a script in /usr/bin called cronjob_bandit22.sh. Following this up and printing out the shell script reveals that the password has been copied to a folder in /tmp/, allowing us to cat the file and retrieve the password for the next level.

```
bandit21@melinda:~$ cd /etc/cron.d
bandit21@melinda:/etc/cron.d$ l
behemoth4_cleanup  cronjob_bandit23       leviathan5_cleanup    natas-
   session-toucher  natas25_cleanup~  php5        semtex0-ppc  vortex0
cron-apt           cronjob_bandit24       manpage3_resetpw_job  natas-
   stats            natas26_cleanup   semtex0-32  semtex5
   vortex20
cronjob_bandit22   cronjob_bandit24_root  melinda-stats
   natas25_cleanup        natas27_cleanup   semtex0-64  sysstat
bandit21@melinda:/etc/cron.d$ cat cronjob_bandit22
* * * * * bandit22 /usr/bin/cronjob_bandit22.sh &> /dev/null
bandit21@melinda:/etc/cron.d$ cat /usr/bin/cronjob_bandit22.sh
#!/bin/bash
chmod 644 /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
cat /etc/bandit_pass/bandit22 > /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
bandit21@melinda:/etc/cron.d$ cat /tmp/
   t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Additional Reading

- Cron on Wikiapedia.
- Intro to cron.

3.23 Level 22

Goal A program is running automatically at regular intervals from cron, the time-based job scheduler. Look in **/etc/cron.d/** for the configuration and see what command is being executed.

Solution To solve this, we cd to the cronjob directory and have a look at what's running. We can see there's a file called 'cronjob_bandit23', so we cat it to view its contents. We can see it's running a script at '/usr/bin/cronjob_bandit23.sh', if we cat the script, we can see that there's some password copying shenanigans going on. If we follow up on this and cat the file the script copies to, we can observe that it prints the password for level 22, our current level. If we modify the whoami component to pretend to be bandit23, it will output the password for bandit23.
Humourously we can also use this to get to level 24, but I think this is unintentional. ;)

```
bandit22@melinda:~$ cd /etc/cron.d
bandit22@melinda:/etc/cron.d$ l
behemoth4_cleanup  cronjob_bandit23       leviathan5_cleanup    natas-
   session-toucher  natas25_cleanup~  php5        semtex0-ppc  vortex0
cron-apt           cronjob_bandit24       manpage3_resetpw_job  natas-
   stats            natas26_cleanup   semtex0-32  semtex5
   vortex20
cronjob_bandit22   cronjob_bandit24_root  melinda-stats
   natas25_cleanup        natas27_cleanup   semtex0-64  sysstat
bandit22@melinda:/etc/cron.d$ cat cronjob_bandit23
* * * * * bandit23 /usr/bin/cronjob_bandit23.sh  &> /dev/null
bandit22@melinda:/etc/cron.d$ cat /usr/bin/cronjob_bandit23.sh
#!/bin/bash

myname=$(whoami)
mytarget=$(echo I am user $myname | md5sum | cut -d ' ' -f 1)

echo "Copying passwordfile /etc/bandit_pass/$myname to /tmp/$mytarget"

cat /etc/bandit_pass/$myname > /tmp/$mytarget
bandit22@melinda:/etc/cron.d$ cat /tmp/`echo I am user $(whoami) |
   md5sum | cut -d ' ' -f 1`
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
bandit22@melinda:/etc/cron.d$ cat /tmp/`echo I am user bandit23 |
   md5sum | cut -d ' ' -f 1`
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
bandit22@melinda:/etc/cron.d$ cat /tmp/`echo I am user bandit24 |
   md5sum | cut -d ' ' -f 1`
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
```

Note: Looking at shell scripts written by other people is a very useful skill. The script for this level is intentionally made easy to read. If you are having problems understanding what it does, try executing it to see the debug information it prints.

3.24 Level 23

Goal A program is running automatically at regular intervals from cron, the time-based job scheduler. Look in **/etc/cron.d/** for the configuration and see what command is being executed.
Note: Keep in mind that your shell script is removed once executed, so you may want to keep a copy around.

Solution As with our previous cron levels, we cd to the cronjob directory and list the cronjobs. Again, there's a juicy cronjob_bandit24 to have a look at. Catting this file reveals that the cronjob is running a script which executes and then deletes all scripts in /var/spool/bandit24/. As we know they will be executed by bandit24 or a higher user, we can copy a script in there that will output the contents of his password file to the /tmp/ folder for us to collect. Remember that crons are a time based service, so you might need to wait a minute or so for it to be executed.

```
bandit23@melinda:/etc/cron.d$ l
behemoth4_cleanup  cronjob_bandit23       leviathan5_cleanup    natas-
   session-toucher  natas25_cleanup~  php5        semtex0-ppc  vortex0
cron-apt           cronjob_bandit24       manpage3_resetpw_job  natas-
   stats            natas26_cleanup   semtex0-32  semtex5
   vortex20
cronjob_bandit22   cronjob_bandit24_root  melinda-stats
   natas25_cleanup        natas27_cleanup   semtex0-64  sysstat
bandit23@melinda:/etc/cron.d$ cat cronjob_bandit24
* * * * * bandit24 /usr/bin/cronjob_bandit24.sh &> /dev/null
bandit23@melinda:/etc/cron.d$ cat /usr/bin/cronjob_bandit24.sh
#!/bin/bash

myname=$(whoami)

cd /var/spool/$myname
echo "Executing and deleting all scripts in /var/spool/$myname:"
for i in * .*;
do
    if [ "$i" != "." -a "$i" != ".." ];
    then
  echo "Handling $i"
  timeout -s 9 60 "./$i"
  rm -f "./$i"
    fi
done
bandit23@melinda:/etc/cron.d$ vim /tmp/roomservice.sh
bandit23@melinda:/etc/cron.d$ cat /tmp/roomservice.sh
#!/bin/bash

cat /etc/bandit_pass/bandit24 > /tmp/GiffPassword
bandit23@melinda:/etc/cron.d$ chmod 755 /tmp/roomservice.sh
bandit23@melinda:/etc/cron.d$ copy /tmp/roomservice.sh /var/spool/
   bandit24/
cat /tmp/GiffPassword
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

3.25 Level 24

Goal A daemon is listening on port 30002 and will give you the password for bandit25 if given the password for bandit24 and a secret numeric 4-digit pincode. There is no way to retrieve the pincode except by going through all of the 10000 combinaties. This is a method called brute-forcing.

Solution As there's no way for us to know what the secret pin is, we'll have to guess it. Rather than doing it manually, we're going to set up a Python script to do it for us. Fortunately there is no rate-limiting, so we are able to send very large amounts of guesses per second.

```
bandit24@melinda:~$ cd /tmp
bandit24@melinda:/tmp$ vim PushyStoreClerk.py
bandit24@melinda:/tmp$ cat PushyStoreClerk.py
#!/usr/bin/python

import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('localhost', 30002))
s.recv(1024)

for i in range(1111,9999):
    print "Trying: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX "+str(i)
    s.sendall("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX "+str(i)+"\n")
    data = s.recv(1024)
    print data
bandit24@melinda:/tmp$ chmod 755 PushyStoreClerk.py
bandit24@melinda:/tmp$ ./PushyStoreClerk.py
Trying: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 1111
Wrong! Please enter the correct pincode. Try again.

Trying: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 1112
Wrong! Please enter the correct pincode. Try again.

Trying: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 1113
Wrong! Please enter the correct pincode. Try again.

...

Trying: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX YYYY
Correct!
The password of user bandit25 is ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
```

Additional Reading

- Python 2.7 Socket Programming.
- Python 3.5 Socket Programming.

3.26 Level 25

Goal Logging in to bandit26 from bandit25 should be fairly easy... The shell for user bandit26 is not **/bin/bash**, but something else. Find out what it is, how it works and how to break out of it.

Solution Logging into bandit26 using the private sshkey reveals that a banner is being printed before the connection is terminated. To inspect further, we can find out what bandit26's login shell is in /etc/-passwd. Running a grep on /etc/passwd for 'bandit26' reveals that his login shell is something called showtext in /usr/bin/. Inspecting this reveals that it is really a shell script that uses more to print what is presumably the bandit26 ASCII art banner we were observing earlier.

This is great news for us, as we can exploit the behaviour of more so that it does not immediately terminate. If we resize our terminal to be less than 5 lines tall (the amount of lines in the bandit26 ascii art), more will only be able to show us part of the text and will allow us to navigate up and down to see the set of the bandit26 ascii art. (Meaning it will stay open and not exit.)

However, as we're able to now interact with more, we can abuse it's functionality to gain access a text editor called Vim. If we press v whilst in more, it'll open a Vim session. From here, we can type ':e /etc/bandit_pass/bandit26' to open bandit26's password file in Vim, allowing us to see what the password is.

```
bandit25@melinda:~$ ls
bandit26.sshkey
bandit25@melinda:~$ ssh -i bandit26.sshkey bandit26@127.0.0.1
  _                     _ _ _   ___   __
 | |                   | (_) | |__ \ / /
 | |__   __ _ _ __   __| |_| |_   ) / /_
 | '_ \ / _` | '_ \ / _` | | __| / / '_ \
 | |_) | (_| | | | | (_| | | |_ / /| (_) |
 |_.__/ \__,_|_| |_|\__,_|_|\__|____\___/
Connection to 127.0.0.1 closed.
bandit25@melinda:~$ grep 'bandit26' /etc/passwd
bandit26:x:11026:11026:bandit level 26:/home/bandit26:/usr/bin/
   showtext
bandit25@melinda:~$ cat /usr/bin/showtext
#!/bin/sh

more ~/text.txt
exit 0
```

```
v
:e /etc/bandit_pass/bandit26
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
~
~
~
~
"/etc/bandit_pass/bandit26" [readonly] 1L, 33C                1,1
             All
```

That's all, folks. Grats on clearing Bandit! - C