Wagner
Spring 2014

CS 161
Computer Security

Project 2

Due: March 17, 11:59PM

Version 0: March 3, 2014

Background

It is a time of rebellion. The evil empire of Caltopia oppresses its people with relentless surveillance, and the emperor has recently unveiled his latest grim weapon: a supremely powerful botnet, called *Calnet*, that aims to pervasively observe the citizenry and squash their cherished Internet freedoms.

Yet in the enlightened city of Birkland, a flicker of hope remains. The brilliant University of Caltopia alumnus Neo, famed for not only his hacking skills but also the excellent YouTube videos he produces illustrating his techniques, has infiltrated the empire's byzantine networks and hacked his way to the very heart of the Calnet source code repository. As the emperor's dark lieutenant, Prof. Evil of Junior University, attempts to hunt him down, Neo feverishly scours the Calnet source code hunting for weaknesses. He's in luck! He realizes that Prof. Evil enlisted ill-trained CS students from Junior University in writing Calnet, and unbeknownst to the empire, the code is assuredly not memory-safe.

Alas, just as Neo begins to code up some righteous exploits to pwn Calnet's components, a barista at the coffeeshop where Neo gets his free WiFi betrays him to Prof. Evil, who brutally deletes Neo's YouTube account and swoops in with a SWAT team to make an arrest. As the thugs smash through the coffeeshop's doors, Neo gets off one final tweet for help. Such are his hacking skillz that he crams a veritable boatload of key information into his final 140 characters, exhorting the University of Birkland's virtuous computer security students to carry forth the flame of knowledge, seize control of Calnet, and let freedom ring once more throughout Caltopia ...

Getting Started

Neo has determined that the correct mojo for this task is you must work on it in teams of 2 students. He expects your team to develop exploits for 5 vulnerabilities in Calnet's components. As they topple you will move closer and closer towards p0wning the nefarious botnet. All you have to go by are your wits, your grit, and Neo's legacy: guidelines on how to proceed, and, most precious, a virtual machine (VM) image that contains code samples from the main Calnet components.

Software Setup

You can run and investigate the VM on your own computer. You will need the following software:

- VirtualBox[1], the virtualization server
- Your favorite text editor
- Your favorite shell[2]
- Your favorite SSH client[3]
- `nmap` security scanner[4]
- `netcat`[5]

On Linux and Mac, you can install `nmap`, `nc` and `ssh` from your package manager. On Windows, you can install Cygwin[2] and use its package manager.

NOTE: Only use these tools against your own infrastructure. You violate campus policy when directing them against parties who do not provide their informed consent!

Start VirtualBox and go to File → Preferences → Network. Make sure there is a network adapter listed under "Host-only Networks" named `vboxnet0`.[6] If the adapter list is empty, click the plus on the right side which will add a new interface. Confirm with OK.

Neo placed the VM image at `http://www-inst.eecs.berkeley.edu/~cs161/sp14/projects/proj2/pwnable.ova`. Download it and import it via File → Import Appliance.

You will run the vulnerable programs and their exploits inside the VM. The image is a barebones Ubuntu Linux server installation on a 32-bit Intel architecture. The first time you boot the image, you have to enter your class accounts in the format `cs161-`x_1x_2`,cs161-`x_3x_4, where $x_1, \ldots, x_4$ are the letters of your class accounts. You need to list the accounts in alphabetical order. For example, if a student with class account `cs161-we` teams with a student with class account `cs161-vv`, then you would enter the string "`cs161-vv,cs161-we`".[7]

[1] VirtualBox is available at `https://www.virtualbox.org`, or from your package manager in Linux. Neo has succesfully used versions 4.1.12 and 4.2.6

[2] On Windows, Neo recommends Cygwin/`bash`: `http://cygwin.com/install.html`

[3] On Windows, Neo recommends PuTTY: `http://www.chiark.greenend.org.uk/~sgtatham/putty/`

[4] `nmap` is available at `http://nmap.org/download.html`

[5] On Windows, Neo recommends installing Cygwin/`bash` and then installing the netcat package when running `setup.exe` (`nc` is part of the `NET` package). *N.B.* netcat may have a different name depending on your operating system (e.g. `nc`, `ncat`, or `netcat`).

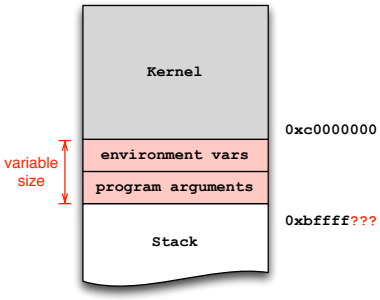
[6] On Windows, the interface may have a different, much longer name.

[7] If you want to do some initial exploration by yourself before you've finalized your team, you can start off using just your class account for this configuration step. Once you have your team in place, you'll need to start again with a clean VM image configured as mentioned here. Any exploits you've developed for your private VM image will require porting (re-determination of the addresses to use in them). This should go quickly once you've learned how to figure out the addresses in the first place.

Some Important Advice Concerning Execution Environments

NOTE: This advice does not concern Question 1.

Exploit development can lead to serious headaches if you don't adequately account for factors that introduce *non-determinism* into the debugging process. In particular, the stack addresses in the debugger may not match the addresses during normal execution. This artifact occurs because the operating system loader places both environment variables and program arguments *before* the beginning of the stack:



Already installed in the VM you'll find a small helper utility, `invoke`, that makes sure environment and arguments remain at the same location, regardless of whether using the debugger or not. For example, instead of invoking the program `foo` directly via `./foo`, you should instead use `invoke foo`:

```
% ./foo arg1 arg2          # invocation dependent on environment state :-(
% invoke foo arg1 arg2     # deterministic invocation
% invoke -d foo arg1 arg2  # deterministic invocation in gdb
```

You may find it useful to pass an extra environment to the program. The `-e` switch serves that purpose:

```
% invoke -e X=Y foo arg1   # sets environment variable X=Y in foo
```

NOTE: You must always use `invoke` to launch (or debug via `-d`) the provided executables because `invoke` additionally parameterizes the execution environment based on the ID you entered during the first boot. More broadly, since our grading tool uses the exact same VM that you downloaded, do not perform *any* system modifications, only add/upload new content. (For example, do not attempt to recompile the given executables.) This way you ensure that your solutions will work with our grading tool and you do not run the risk of losing unnecessary points.

In addition, we are providing a tool that allows you to pipe input while using GDB. (This can be useful for your writeups!) Download it at http://www-inst.eecs.berkeley.edu/~cs161/sp14/projects/proj2/gdbpipe-tools.tgz. This is useful when you're trying to do something like the following, which will not work in GDB:

```
% (./egg;cat) | invoke -d dejavu
```

Instead, to do the equivalent of the comman above, first type (assuming you've unpacked the archive in the current directory):

```
% ./invoke -t dejavu
```

Then, open a new SSH session, and run:

```
% (./egg;cat) | ./gdbpipe
```

Now, when you type 'r' in gdb in the first window, you can interact with the program like normal in the second window. Press Ctrl+C to exit and start a new pipe each time you re-run the program.

The Task

Unfortunately Neo did not have enough time to figure out the necessary login credentials. It is up to you to break into the VM and continue his mission, with the ultimate goal to gain root privileges on the machine to have full control over Calnet. Neo's intelligence sources revealed that, once broken in the system, the required login credentials necessary for further access are located inside the system itself.

You know from having watched his YouTube channel that Neo advocates a three-step approach for breaking into a system:

Step 1: Reconnaissance. Investigate what software/which services is/are running (***hint:*** `nmap`). Determine if there is anything you can access (***hint:*** `netcat`). What can you discover about the software (e.g., in terms of version; do you have the source code)? Using this information you can seek out potential vulnerabilities.

Step 2: Development. After you have found a vulnerability, you can create an exploit using the found bugs(generally, as an attacker, this means crafting a malicious input to the buggy program).

Step 3: Profit.

Use Neo's three-step plan to solve the following problems.

Question 1 ***Gaining VM Access*** **(20 points)**

Neo knew that it could prove daunting to find yourself confronted with an unknown system without login credentials. Upon skimming his tweets, you find out that one standard procedure to break into systems begins with a port scan via `nmap`, which tells you what services run on the machine. Moreover, you learn about `netcat`. Familiarize yourself with these tools by reading their man pages and try to use them to get a foothold in the system!

NOTE: You need to gain access to the VM *via the network*, as opposed to mounting the filesystem locally and browsing the contents.

Submission and Grading. For this problem you will submit a shell script named `exploit` which takes an IP address as first argument. Our grading tool executes your script as `./exploit` ***address*** where ***address*** represents the IP address of our grading VM. Our tool tests whether the end of the execution spawns a shell with effective privileges of the user `vsftpd`[8] `(10 points)`.

Moreover, you will submit a file called `NETCAT` that includes the first line from the output of `nc -h` where `nc` stands for the netcat flavor you use. You must also submit a file, `q1.txt`, that includes a brief description of the vulnerability, how it could be exploited, and a walkthrough of your solution. You should also include output from any tools you used in your discovery of the exploit. This document should be no more than one page. We will use it to verify that your understanding of the problem matches your exploit code. Moreover, we will use it to award you partial credit in the event that your exploit does not work with our automated grading system `(10 points)`.

Question 2 *Behind the Scenes* (40 points)

Neo's tweet assures you that given its hasty development by poorly educated programmers, Calnet's components contain a number of memory-safety vulnerabilities. In the VM that Neo provided, you will find the first code piece located in the directory `/home/vsftpd`.[9]

You are to continue his work and write an exploit that spawns a shell, for which you can use the following shellcode:

```
shellcode =
  "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07" +
  "\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d" +
  "\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80" +
  "\xe8\xdc\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68"
```

NOTE: Recall that x86 has little-endian byte order, e.g., the first four bytes of the above shellcode will appear as `0x895e1feb` in the debugger.

Neo already provided an exploit scaffold that takes your malicious buffer and feeds it to the vulnerable program via a script called `exploit`:

```
#!/bin/sh
( ./egg ; cat ) | invoke dejavu
```

(As one of Neo's tweets explains in a concise but strikingly lucid fashion, the expression before the shell pipe is necessary so that if the attack input generated by `egg` succeeds, then you will be able to interact with the shell that the exploit spawns by typing via *stdin*.)

[8] When testing your attack, a shell prompt may not appear so you should try running a command such as `ls` or `whoami` to determine you have succeeded, or your script is just hanging.

[9] The vulnerable binary has the *setuid* bit set and is owned by the user of the next stage, meaning it will run with the effective privileges of user `smith`.

To get started, read "Smashing The Stack For Fun And Profit" by AlephOne [1]. Neo recommended that you try to absorb the high-level concepts of exploiting stack overflows rather than every single line of assembly. He also warned you that some of the example codes are outdated and may not work as-is.

Submission and Grading. For this problem you will submit the missing script `egg`, which can be written in your favorite scripting language (e.g., Python, Ruby, Perl, Bash). Your code should print the buffer used by the `exploit` script to spawn a shell. Make sure it works by invoking `./exploit`. Our grading tool will log into a clean VM image as user `vsftpd` and put your submission into the directory `/home/vsftpd`. A script will then invoke the script `exploit` *exactly as given above* and check for the existence of a shell prompt with effective privileges of user `smith (25 points)`.

You must also submit a file, `q2.txt`, that includes a brief description of the vulnerability, how it could be exploited, how you determined which address to jump to, and a sketch of your solution. This includes gdb output that very clearly demonstrates the effects of your exploit (before/after). As before, keep it to no more than one page `(15 points)`.

Question 3 ***Compromising Further*** **(40 points)**

Calnet uses a sequence of stages to protect intruders from gaining root access. The inept Junior University programmers actually attempted a half-hearted fix to address the overt buffer overflow vulnerability from the previous stage. In this problem you must bypass these mediocre security measures and, again, inject code that spawns a shell.

In the home directory of this stage, `/home/smith`, you will find a small helper script `generate-file-contents`. This script takes arbitrary input via *stdin* and prints the first 127 bytes to *stdout* in the format that the program `agent-smith` expects (which is an initial byte specifying the length of the input, followed by the input itself):

```
% ./generate-file-contents < anderson.txt
```

Neo realized that this helper script always generates safe files to be used with the buggy `agent-smith` program—but nothing prevents you from instead feeding `agent-smith` an arbitrary file of your choice. In particular, Neo started a script `exploit` representing an initial exploit attempt:

```
#!/bin/sh
./egg > pwnzerized
invoke agent-smith pwnzerized
```

Submission and Grading. As in the previous question, you will submit a script `egg`, written in your favorite scripting language, that integrates with the above displayed script `exploit`. Your script should inject shellcode to spawn a shell. Make sure it works by invoking `./exploit`. Our grading tool will log into a clean VM image as user `smith` and put your submission into the directory `/home/smith`. A script will then invoke

`exploit` and check for the existence of a shell prompt with effective privileges of user `brown` (25 points).

You must also submit a file, `q3.txt`, that includes the same type of information as for the previous Question (15 points).

Question 4 ***Deep Infiltration*** **(50 points)**

Calnet is a pernicious and invasive piece of malcode. But Prof. Evil undertook all of his own studies at Junior University, and as such he never really learned how to count without occasionally screwing it up. Find the subtle vulnerability in this code, and inject code that spawns a shell.

Neo, again on top of it, started a scaffold called `exploit` that you can use:

```
#!/bin/sh
invoke -e egg=$(./egg) agent-brown $(./arg)
```

(Note that a shell expression like "`$(foo)`" means "run the command `foo` and substitute its *stdout* output here." So "`egg=$(./egg)$`" means "run the command `./egg` and assign the output it generates to the variable `$egg`.")

To solve this problem, you are pretty sure that a cryptic reference in Neo's tweets indicates you'd benefit from reading Section 10 of "ASLR Smack & Laugh Reference" by Tilo Müller [2]. (Although the title suggests that you have to deal with ASLR, you can ignore any ASLR-related content in the paper for this question.)

Submission and Grading. For this question question, you will submit a script `arg` and a script `egg` written in your favorite scripting language. Your code should integrate with the script `exploit` as shown above. Make sure your scripts work by invoking `./exploit`. Our grading tool will log into a clean VM image as user `brown` and put your submission into the directory `/home/brown`. A script will then invoke `exploit` and check for the existence of a shell prompt with effective privileges of user `jz` (30 points).

As for the previous question, you must also submit a file, `q4.txt`, that includes a brief description of the vulnerability, how it could be exploited, how you determined which address to jump to, and a sketch of your solution. This includes gdb output that very clearly demonstrates the effects of your exploit (before/after) (20 points).

Question 5 ***The Last Bastion*** **(50 points)**

To protect the Calnet source from advanced hackers, Prof. Evil's minions persuaded him that he must enable address layout randomization (ASLR) as a final layer of defense for the VM. They assured him that it was **inconceivable** that **anyone even of super-human intelligence** would possess the uber-h4x0r skillz required to overcome this.

Yo, Birkland! Your mission, should you choose to accept it, is to bypass the ASLR protection and spawn a shell with root privileges. Full control of the box ... *and thus Calnet itself* awaits you! Neo didn't dare hope you might hack your way this far and this

deeply ... but he could never abandon his dream of freedom, and to that end provided an exceedingly cryptic clue in his final tweet that after a caffeine-fueled all-nighter you eventually realize suggests you should consider reading Section 8 of "ASLR Smack & Laugh Reference" by Tilo Müller [2].

One detail Neo *could* figure out for you is that the service to exploit listens locally on TCP port 42000. It turns out that the operating system watches the service and restarts it shortly when it crashes. You have to send the malicious shellcode to that service to successfully complete this task. Looking through Neo's past tweets, you find guidance to develop this in the form of a TCP "bind shell" listening on 127.0.0.1:6666.

```
# Linux (x86) TCP shell binding to port 6666.
bind_shell =
  "\x31\xdb\xf7\xe3\x53\x43\x53\x6a\x02\x89\xe1\xb0\x66\xcd" +
  "\x80\x5b\x5e\x52\x68\x02\x00\x1a\x0a\x6a\x10\x51\x50\x89" +
  "\xe1\x6a\x66\x58\xcd\x80\x89\x41\x04\xb3\x04\xb0\x66\xcd" +
  "\x80\x43\xb0\x66\xcd\x80\x93\x59\x6a\x3f\x58\xcd\x80\x49" +
  "\x79\xf8\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3" +
  "\x50\x53\x89\xe1\xb0\x0b\xcd\x80"
```

This should finally suffice to pull off the Final Stage! Somehow you must code up the program `egg` so that Neo's `exploit` script can launch the final, fatal strike:

```
#!/bin/sh
echo "sending exploit"
./egg | nc 127.0.0.1 42000 &
sleep 1
nc ...
```

Alas, the battery in Neo's ultra-thin BlueTooth keyboard died just as he was finishing typing here. To successfully employ the script, you'll need to replace "`...`" with the required arguments to access the root shell.

The freedom of cybercitizens throughout Caltopia rests in your hands ...

Submission and Grading. For this question question, you will submit a complete shell script `exploit` that carries out the attack and spawns a shell with root privileges. You will also submit a script `egg`, written in your favorite scripting language, that prints the exploit buffer to standard output and pipes it to `netcat`. Make sure your scripts work by invoking `./exploit`. Our grading tool will log into a clean VM image as user `jones` and put your submission into the directory `/home/jones`. A script will then invoke `exploit` and check for the existence of a shell prompt with effective privileges of user `root (30 points)`.

You must also submit a file, `q5.txt`, in the same fashion as for the previous question `(20 points)`.

Question 6 ***Feedback (optional)*** **(0 points)**

If you wish, submit a text file, `feedback.txt`, with any feedback you may have about this project. What was the hardest part of this project in terms of understanding? In terms of effort? (We also, as always, welcome feedback about other aspects of the class.) Your comments will not in any way affect your grade.

Submission Summary

In summary, you must submit the following directory tree:

```
q1/exploit
q1/q1.txt
q1/NETCAT
q2/egg
q2/q2.txt
q3/egg
q3/q3.txt
q4/arg
q4/egg
q4/q4.txt
q5/egg
q5/exploit
q5/q5.txt
feedback.txt (optional)
```

References

[1] Aleph One. Smashing The Stack For Fun And Profit. *Phrack*, 7(49), November 1996. http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf.

[2] Tilo Müller. ASLR Smack & Laugh Reference. http://www.icir.org/matthias/cs161-sp13/aslr-bypass.pdf, February 2008.