Computer and Information Security

(ECE560, Fall 2020, Duke Univ., Prof. Tyler Bletsch)

Homework 4

Name: ___Yisong Zou_____

Duke NetID: ___yz558_____

Instructions - read all carefully:

- **DON'T SCREW UP:** Read each question carefully and be sure to answer all parts. Some questions are a mix of explanation and questions, so pay close attention to where you are being asked for something.
- **COMPUTERS YOU WILL NEED:**
  - The assignment will make use of the computers described below.
  - VMs you already have on the Duke VCM service:
    - Your **Linux VM** ("*ECE.560.01.F20 - Ubuntu20.04*" on VCM)
    - Your **Windows VM** ("*ECE.560.01.F20 - Win10*" on VCM)
    - A new **Kali VM** ("ECE.560.01.F20 - Kali 2002.2" on VCM)
  - Your own machine on Duke wifi: your **personal computer** (any OS).
    (If working remotely, VPN into the Duke network as needed.)
  - A new **throw-away Ubuntu 20.04 VM**, which we'll call your **backup server.**
  - A new **throw-away Windows 10 VM**, either local or cloud-hosted (not VCM!).
- **WRITTEN PORTION DIRECTIONS:**
  - This assignment is designed to be copied into a new document so you can answer questions inline (either as a Google doc or in a local word processor).
  - This assignment should be submitted as a **PDF through Gradescope**. Other formats or methods of submission will not be accepted.
  - When you submit, the tool will ask you to mark which pages contain which questions. This is easiest if you avoid having two questions on one page and keep the large question headers intact. Be sure to mark your answer pages appropriately.
- **PROGRAMMING PORTION DIRECTIONS:**
  - There is a buffer overflow programming project in this assignment; **your code for this will be submitted as a separate file** via the **Sakai assignment facility**. See the question itself for details.
- **CITE YOUR SOURCES:** Make sure you document any resources you may use when answering the questions, including classmates and the textbook. Please use authoritative sources like RFCs, ISOs, NIST SPs, man pages, etc. for your references.

This assignment is adapted from material by Samuel Carter (NCSU).

Question 0: Accessing the Homework (0 points, but necessary)

To get here, you found the URL via one of two steganography methods. Did you find this hint file along the way?

Question 1: Show your work from Question 0 (2 points)

Below, **show the code and/or image work you did to get here.**

On the top left of the photo, there is a watermark which can be seen. Which is the URL:
https://tinyurl.com/nice560h4

OPTIONAL: For 5 points of extra credit, give the secret message included with the URL in the bitwise-encoded message. See the hints file for details.

https://tinyurl.com/nice560h4
Secret message: "PROPER BITWISE HIJINKS"

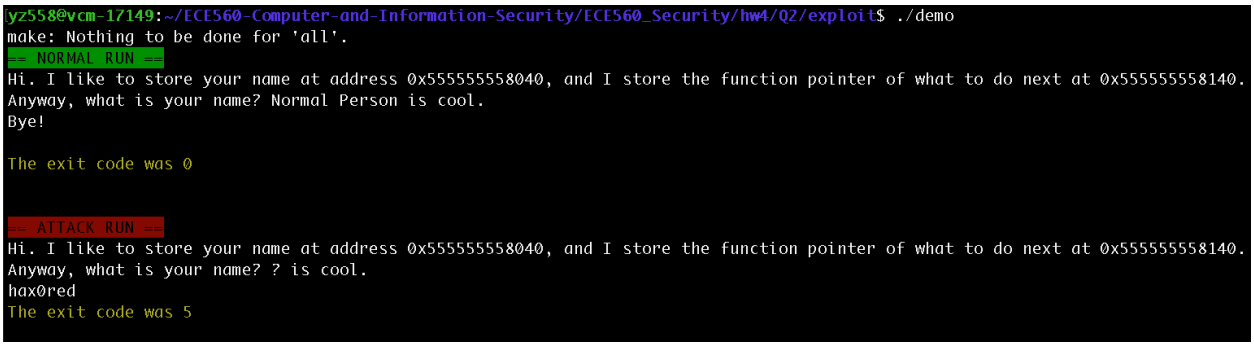
Question 2: Buffer Overflow (10 points)

This question is highly flexible, so read carefully!

Here's how to disable ASLR. This article in the highly esteemed blog publication "m0skit0.org" explains how to turn off W^X (also known as NX support) at compile time (done by default by the Makefile in the starter kit linked below).

Here's the base problem:

- **Prepare VM:** In your Linux VM, install nasm ("sudo apt install nasm").
    - Note: If you want to build an attack for another OS, contact the instructor for permission, which I'll likely give, but you'll be on your own.
- **Get set up:** Download, extract, and test the **example exploit kit**.
    - The Makefile is set up to build the vulnerable program with W^X disabled.
    - Read through the vulnerable program "vuln1.c" and the attack script "attack.asm" closely.
    - **Watch this walkthrough video I recorded.**
      NOTE: This video was recorded for an earlier version of the problem, so there are slight differences, in particular, it claims that we need to avoid null bytes in the attack buffer, this is actually not true for a gets() exploit.
    - The included attack will make the program skip saying "Bye!" and exit with status code 5 instead of the usual 0. Recall that the exit status of a program can be checked by running "echo $?" after the program exits.
    - Assuming your VM's memory map looks like mine, you should be able to run the attack straight away by executing the included "demo" script. If it doesn't work, you may need to update the memory location constants in the attack script, re-build, and try again.
- **Make exploit:** Develop an attack buffer that makes the program print "`hax0red`".
    - Note: your *exploit* must cause it to say this -- merely having the word "`hax0red`" appear among the program's usual "so-and-so is cool" output does not count.
    - You can use any tool you wish to develop the attack buffer, though using the included NASM attack.asm as a starting point is probably easiest.
    - gdb is your friend.
    - If pursuing the extra credit (see below), you may target a program *other than* the provided vulnerable program.
- **Hack:** Run your attack and paste a screenshot of it succeeding below.



- **Document:** Make a file called “readme.txt” that lists all dependencies your attack has (packages needed to be installed, special steps to be taken, including the steps to disable ASLR and W^X)).
- **Submit:** Gather up *EVERYTHING* involved in the attack (readme.txt, the vulnerable program source and binary, the attack source code and attack binary file, and any Makefiles or helper scripts). Zip all of this up and submit it to Sakai as “<NetID>_attack.zip”.

Here’s some random tips to help you:

- GDB quick reference card.
- Syscall numbers.
- Intel x86 assembly reference:
  - The giant Intel big book: the authoritative source.
  - The felixcloutier x86 instruction reference.
  - NASM manual, including instruction list.
  - Low-level conversions between hex bytes and CPU instructions: Numeric order, mnemonic order.
- You can use “`ndisasm -b64 <file>`” to do a raw disassembly of your attack buffer, with output in Intel (NASM) syntax.
- You can use “`hd <file>`” to get a hexdump of it, which is useful when looking for nulls or whitespace bytes that snuck into your attack buffer.

Extra credit achievements

You may apply any combination of the following achievements to the problem for extra credit. Put an "X" in the brackets and color the whole line red for any of these variations that you're claiming. Then, in your readme, ***explain in detail*** how you achieved the goals claimed.

Note -- there are a lot of unknowns in this program, so the instructor reserves the right to adjust extra credit if there's some trivial way to get way too many points.

Extra credit for protections left enabled:

- [ ] (+5) **Defeat ASLR:** Leave ASLR on.
- [ ] (+5) **Defeat W^X:** Leave W^X on.

Extra credit for choice of target:

- [ ] (+2) **Custom target:** Develop a significantly different sample vulnerable program from scratch. This program must use a different exploitable function instead of `gets()` (e.g. `scanf()`) and/or a different kind of control data instead of a function pointer (e.g. a return address).
- [ ] (+2) **Server target:** Your target program accepts the attack buffer over the network.
- [ ] (+10) **Kernel target:** Exploit code operating in kernel space instead of user space. In this case, your printing of the message can go to the kernel log or to another creative destination.
- [ ] (+15) **Real target:** Instead of a toy sample program, attack a real program. The program must have been in significant production at some point in the past 10 years. If you go with this option, your attack must still be completely from scratch -- you can't just use metasploit or a sample script or something. You must also cite sources in your readme and explain in detail how the vulnerability and your exploit work.
- [ ] (+30) **Very real target:** Base your attack on a brand-new never-before-published vulnerability in a real program in significant production use. If you go with this option, you must document in detail how the vulnerability was discovered, how your exploit works, and you must submit a report to the software vendor as well as the instructors. This achievement implies "Real target".
- [ ] (+60) **Bragging rights:** Awarded if you did the "Very real target" achievement and your discovery of the vulnerability results in the publication of an advisory from an industry-standard source (CVE, NVD, an article in a major vendor knowledge base such as the Microsoft KB, public bug report and patch, etc.). The instructor will also buy you dinner.

Extra credit for payload:

- [ ] (+2) **Verbose hacker:** If your attack prints more text than just "hax0red", including multiple newline characters. This can be a challenge since the newline character is the delimiter for `gets()`, which is used in the sample vulnerable program.

- [ ] (+5) **Math hack:** If your payload computes and prints the first 50 Fibonacci numbers in addition to the message. Note that the attack must *compute* the numbers, not just print them from memory.
- [ ] (+5) **Big math hack:** Same as above, but the first 100 Fibonacci numbers, and they must be to full precision. (Note: this exceeds a 64-bit register, so you'll have to get creative)
- [ ] (+5) **File IO:** If your payload causes the program to appear to the user to function normally in every way, except the message you're printing is saved to a file called "0wned.txt". (Note the zero in the filename -- this is used to indicate that we are very cool, mature people.)
- [ ] (+10) **Reverse shell:** If your payload connects out to a TCP host and, upon successful connection, establishes remote control of a /bin/bash shell.
- [ ] (+10) **Code reuse:** If your payload operates *entirely* on code reuse and does no code injection.
- [ ] (+15) **Dr. Bletsch's dissertation:** If you payload operates on code reuse and does no code injection, *and* does not exploit a single `ret` instruction.

Extra credit wildcard:

- [ ] (+$1000) **Overachiever**: If you unlock every extra credit achievement above, I will hand you a thousand dollars.
- [ ] (+?) **Wildcard:** If you're doing something else fancy with your attack that you feel is worth more points, pitch it to the instructor. You will need to document this additional feature in your readme file.

Question 3: Buffer Overflow Prevention (5 points)

Fix it: Take whatever program you used as your target for the previous problem and fix the vulnerability. Submit the fixed code as “<NetID>_fixed.zip” to Sakai.

Prove you actually fixed it: Paste a screenshot of the program NOT being exploited by the attack input from the previous question. Note: If a screenshot cannot show the improvement for your particular program, contact the instructor to discuss an alternative form of response.

```
yz558@vcm-17149:~/ECE560-Computer-and-Information-Security/ECE560_Security/hw4/Q2/fix$ ./demo
make: Nothing to be done for 'all'.
== NORMAL RUN ==
Hi. I like to store your name at address 0x555555558040, and I store the function pointer of what to do next at 0x555555558140.
Anyway, what is your name? Normal Person is cool.
Bye!

The exit code was 0


== ATTACK RUN ==
Hi. I like to store your name at address 0x555555558040, and I store the function pointer of what to do next at 0x555555558140.
Anyway, what is your name? ? is cool.
Bye!

The exit code was 0
```

Prove you didn’t make it worse: Paste a screenshot of the program continuing to function correctly under normal input.

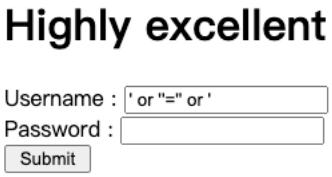
```
yz558@vcm-17149:~/ECE560-Computer-and-Information-Security/ECE560_Security/hw4/Q2/fix$ ./vuln1
Hi. I like to store your name at address 0x555555558040, and I store the function pointer of what to do next at 0x555555558140.
Anyway, what is your name? zys
zys is cool.
Bye!
yz558@vcm-17149:~/ECE560-Computer-and-Information-Security/ECE560_Security/hw4/Q2/fix$
```

Question 4: SQL Injection (6 points)

Use a SQL injection attack to login to http://sqldemo.googz.us/. This is a simple login page written in PHP and modeled after the many intro-to-PHP guides available online.

Note: do not modify the database in any way; just use the attack to fool the script into logging you in. Also, this is running on a production web host, so do not use brute force techniques in your attack.

Paste the username and password you used below.



username: ' or "=" or '

password: anything

Upon successful login, the system will give you a secret code. Paste this code below.

Secret code is: Some Sharks Typing On A Computer

Extra credit +5: Find out jimmy's full name.

Extra credit +5: Find out the actual admin password.

d41d8cd98f00b204e9800998ecf8427e

Note: If you wish to use a tool or script to make successful requests in pursuit of the extra credit, email me first. If such a request is granted, you will need to throttle your attack to no more than one request every five (5) seconds. No tool is needed to get the main part of the question.

Question 5: Backups with rsnapshot (8 points)

*NOTE: This question is similar to one posed later in my Enterprise Storage Architecture course. If you have taken that course in a previous semester, you may simply write "I did this in Enterprise Storage Architecture in <semester+year>" for full credit (though you can do it again if you wish). If you are enrolled in that course now, it gets a little tricky, since you encounter it in group lab work. In that case, you must do it separately *yourself* for this class. This ensures you don't turn in a group member's contribution in that class for credit in this one. To be clear, in any case, you may not simply paste in a solution from the other course for this question.*

Let's build an automated backup system. Make an additional Linux VM in Duke VCM. This new Linux VM we'll refer to as the **Backup Server**. The thing we'll be backing up is your pre-existing Linux VM; this is the **Backup Client**, You can use this guide, and note the following:

- **Backup source**: We're going to back up your user home directory on the backup client machine. This is `/home/<NETID>`.
- **Backup destination**: As root, make a directory `/backup`.
- **SSH keys needed**: Because the source and destination are different machines, you'll need to set up SSH keys so that the backup server can *pull* data as needed.
- **Frequency**: Set up nightly and weekly intervals.
- **Retention**: You should retain seven nightly backups and four weekly backups.
- **Automation:** You do NOT need to do cron-based automation. Once you have rsnapshot configured, you can just run "rsnapshot nightly" to perform the backup that would happen nightly, and "rsnapshot weekly" to perform the backup that would happen weekly. You can run these commands as often as you wish; there's nothing in the software that actually needs the backups to be done nightly/weekly as opposed to every few dozen seconds (i.e., rsnapshot does not actually care that they're called "nightly" or "weekly").

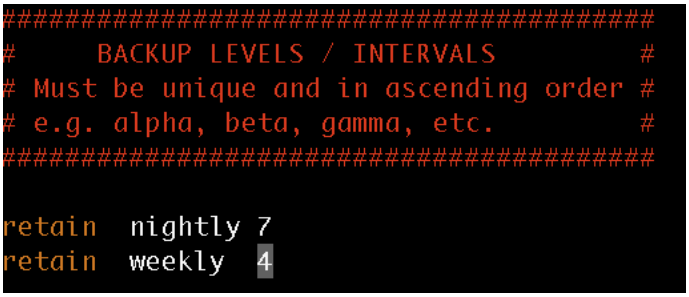
Provide screenshots or a terminal log of these steps in your writeup.

On backup server:

Install rsnapshot:

```
1.  sudo apt-get install rsnapshot
```

Edit the rotation periods inside the /etc/rsnapshot.conf file



Make the directory backup on user root

```
root@vcm-17410:/# mkdir backup
root@vcm-17410:/# ls
backup  bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@vcm-17410:/#
```

setup the directories to backup in **rsnapshot.conf**

```
# EXAMPLE.COM
backup  yz558@vcm-17149.vm.duke.edu:/home/yz558/        vcm-17149.vm.duke.edu/
#backup_exec    /bin/date "+ backup of example.com started at %c"
#backup root@example.com:/home/ example.com/    +rsync_long_args=--bwlimit=16,exclude=cor
#backup root@example.com:/etc/  example.com/    exclude=mtab,exclude=core
#backup_exec    ssh root@example.com "mysqldump -A > /var/db/dump/mysql.sql"
#backup root@example.com:/var/db/dump/  example.com/
#backup_exec    /bin/date "+ backup of example.com ended at %c"
```

Set the backup directory to use upon the localhost in **rsnapshot.conf**

```
###########################
# SNAPSHOT ROOT DIRECTORY #
###########################

# All snapshots will be stored under this root directory.
#
snapshot_root   /backup/

# If no_create_root is enabled, rsnapshot will not automatically create the
# snapshot_root directory. This is particularly useful if you are backing
# up to removable media, such as a FireWire or USB drive.
```

change the /etc/cron.d/rsnapshot file

```
# This is a sample cron file for rsnapshot.
# The values used correspond to the examples in /etc/rsnapshot.conf.
# There you can also set the backup points and many other things.
#
# To activate this cron file you have to uncomment the lines below.
# Feel free to adapt it to your needs.

0 */4           * * *           root    /backup weekly
30 3    * * *           root    /backup nightly
# 0  3          * * 1           root    /usr/bin/rsnapshot gamma
# 30 2          1 * *           root    /usr/bin/rsnapshot delta
```

Enable remote ssh:

```
# Uncomment this to enable remote ssh backups over rsync.
#
cmd_ssh /usr/bin/ssh
```

Automation result:

Run **rsnapshot nightly** **and** **rsnapshot weekly**

```
root@vcm-17410:/backup# ls
nightly.0  nightly.1  nightly.2  nightly.3  nightly.4  nightly.5  weekly.0  weekly.1  weekly.2  weekly.3
root@vcm-17410:/backup# rsnapshot nightly
yz558@vcm-17149.vm.duke.edu's password:
root@vcm-17410:/backup# ls
nightly.0  nightly.1  nightly.2  nightly.3  nightly.4  nightly.5  nightly.6  weekly.0  weekly.1  weekly.2  weekly.3
root@vcm-17410:/backup#
```

Provide the relevant portions of rsnapshot.conf.

```
############################
# SNAPSHOT ROOT DIRECTORY #
############################

# All snapshots will be stored under this root directory.
#
snapshot_root   /backup

# If no_create_root is enabled, rsnapshot will not automatically cr
# snapshot_root directory. This is particularly useful if you are b
# up to removable media, such as a FireWire or USB drive.
#
```

```
#########################################
#     BACKUP LEVELS / INTERVALS         #
# Must be unique and in ascending order #
# e.g. alpha, beta, gamma, etc.         #
#########################################

retain  nightly 7
retain  weekly  4
```

```
# EXAMPLE.COM
backup  yz558@vcm-17149.vm.duke.edu:/home/yz558/        vcm-17149.vm.duke.edu/
#backup_exec    /bin/date "+ backup of example.com started at %c"
#backup root@example.com:/home/ example.com/    +rsync_long_args=--bwlimit=16,exclude=cor
#backup root@example.com:/etc/  example.com/    exclude=mtab,exclude=core
#backup_exec    ssh root@example.com "mysqldump -A > /var/db/dump/mysql.sql"
#backup root@example.com:/var/db/dump/  example.com/
#backup_exec    /bin/date "+ backup of example.com ended at %c"
```

Let's test it. Open one terminal window as root, and another terminal window as the non-root user. Do the following, **and for each step, show the process via screenshots or terminal logs. Label or otherwise identify each step you're doing in your screenshot/log.**

1. As the user on the backup client, add some content to the user home directory.

Here I add a new directory called newContent in backup client

```
yz558@vcm-17149:~$ ls
auth.log  ECE560-Computer-and-Information-Security  mockWebApp  truepolyglot
yz558@vcm-17149:~$ mkdir newContent
yz558@vcm-17149:~$ ls
auth.log  ECE560-Computer-and-Information-Security  mockWebApp  newContent  truepolyglot
yz558@vcm-17149:~$
```

2. As root on the backup server, do several nightly and weekly backups to populate your backups.

Make seven nightly backups first with rsnapshot nightly
Then make the first weekly backup with rsnapshot weekly
Then the number of nightly backup will reduce by one, so I run rsnapshot nightly to make one more nightly backup.
Then make the second weekly backup with rsnapshot weekly
Then the number of nightly backup will reduce by one, so I run rsnapshot nightly to make one more nightly backup.
Then make the third weekly backup with rsnapshot weekly
Then the number of nightly backup will reduce by one, so I run rsnapshot nightly to make one more nightly backup.
Then make the fourth weekly backup with rsnapshot weekly
Then the number of nightly backup will reduce by one, so I run rsnapshot nightly to make one more nightly backup.

```
yz558@vcm-17149.vm.duke.edu's password:
root@vcm-17410:/backup# ls
nightly.0  nightly.1  nightly.2  nightly.3  nightly.4  nightly.5  nightly.6  weekly.0  weekly.1  weekly.2  weekly.3
root@vcm-17410:/backup#
```

3. As the user on the backup client, “corrupt” an important file (overwrite or delete it).

Here I delete the directory called newContent in backup client

```
yz558@vcm-17149:~$ ls
auth.log  ECE560-Computer-and-Information-Security  mockWebApp  newContent  truepolyglot
yz558@vcm-17149:~$ rm -rf newContent/
yz558@vcm-17149:~$ ls
auth.log  ECE560-Computer-and-Information-Security  mockWebApp  truepolyglot
yz558@vcm-17149:~$
```

4. As root on the backup server, take another few nightly backups to simulate time passing.

```
root@vcm-17410:/backup# ls
nightly.0  nightly.1  nightly.2  nightly.3  nightly.4  nightly.5  weekly.1  weekly.2  weekly.3
root@vcm-17410:/backup# rsnapshot nightly
yz558@vcm-17149.vm.duke.edu's password:
root@vcm-17410:/backup# rsnapshot nightly
yz558@vcm-17149.vm.duke.edu's password:
root@vcm-17410:/backup# rsnapshot nightly
yz558@vcm-17149.vm.duke.edu's password:
root@vcm-17410:/backup# ls
nightly.0  nightly.1  nightly.2  nightly.3  nightly.4  nightly.5  nightly.6  weekly.1  weekly.2  weekly.3
root@vcm-17410:/backup#
```

5. Copy the appropriate backup from the backup server to the client’s home directory, thus restoring the damaged file.

Here I can see that the directory newContent still exists inside the nightly.3 backup.

```
root@vcm-17410:/backup# cd nightly.3/vcm-17149.vm.duke.edu/home/yz558/
root@vcm-17410:/backup/nightly.3/vcm-17149.vm.duke.edu/home/yz558# ls
auth.log  ECE560-Computer-and-Information-Security  mockWebApp  newContent  truepolyglot
root@vcm-17410:/backup/nightly.3/vcm-17149.vm.duke.edu/home/yz558#
```

I will use scp to copy the backup to the client:

```
root@vcm-17410:/backup/nightly.3/vcm-17149.vm.duke.edu/home# ls
yz558
root@vcm-17410:/backup/nightly.3/vcm-17149.vm.duke.edu/home# scp -r ./yz558 yz558@vcm-17149.vm.duke.edu:/home/
```

And inside the client we can see the directory newContent is recovered

```
yz558@vcm-17149:~$ ls
auth.log  ECE560-Computer-and-Information-Security  mockWebApp  newContent  truepolyglot
yz558@vcm-17149:~$
```

Note: If your SSH MFA is interfering with this, you may either make a separate user without MFA enabled, or disable MFA altogether.

Question 6: Run a honeypot and see what you get (8 points)

A honeypot is an intentionally-vulnerable system with monitoring. It is used to see what attackers do when they succeed in a given environment. There are many kinds of honeypots (Windows/RDP, Linux/SSH, etc.). They can be divided into “real environment” (i.e., the attacker is really breaking in, but the system they’re gaining access to is one we don’t care about) versus “simulated environment” (i.e., it looks like the real thing, but every operation is simulated and sandboxed).

Setup (4pts)

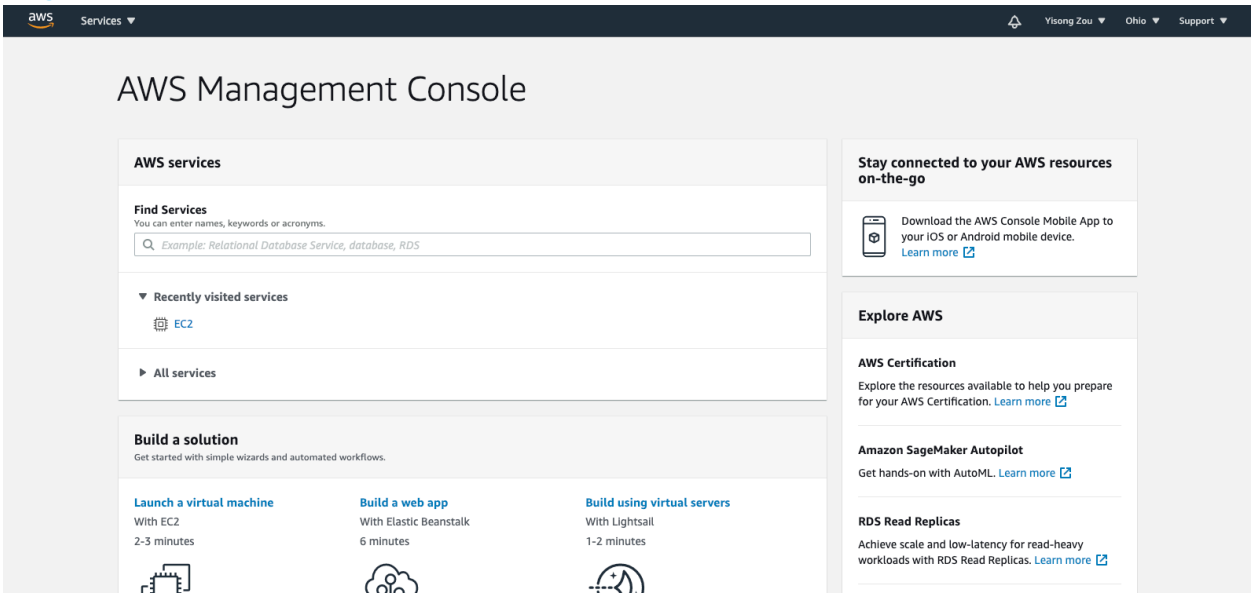
We’ll be setting up a simulated-environment SSH sandbox called Cowrie.

Because Duke IT has a lot of existing automated defenses, for best results, you’ll be installing to the Amazon cloud as an EC2 instance. As far as money goes, you can use the AWS Free Tier or the AWS Educate program, or just give them the ~$5 this experiment should cost. You may use another commercial cloud (Linode, Vultr, Digital Ocean, Azure, etc.) if you wish. You may want to play with what geographic area you place your VM, as this can affect what kind of attacks you see.

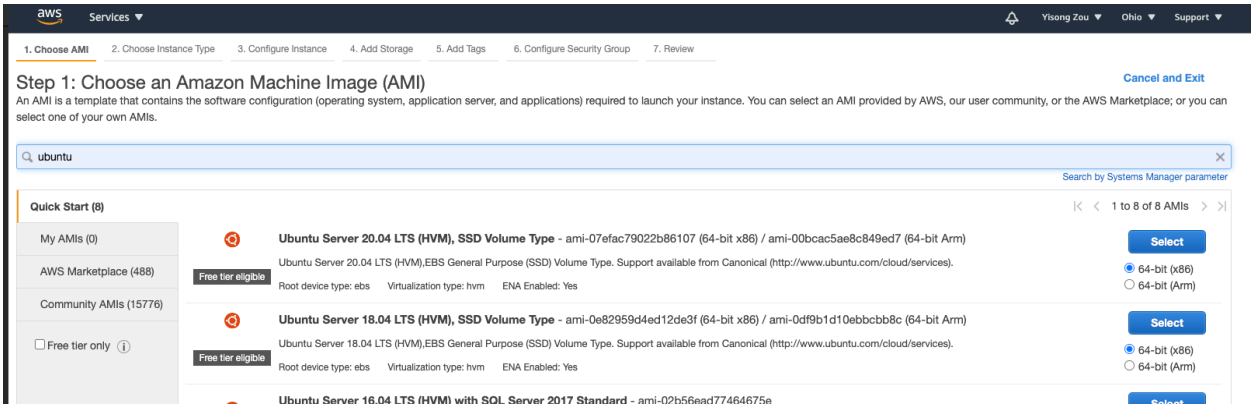
Set up your VM with Cowrie listening on port 22 and the real SSH daemon listening on port 60022. Be sure your cloud’s network settings allow access on both ports.

Show the steps needed to set up this environment.

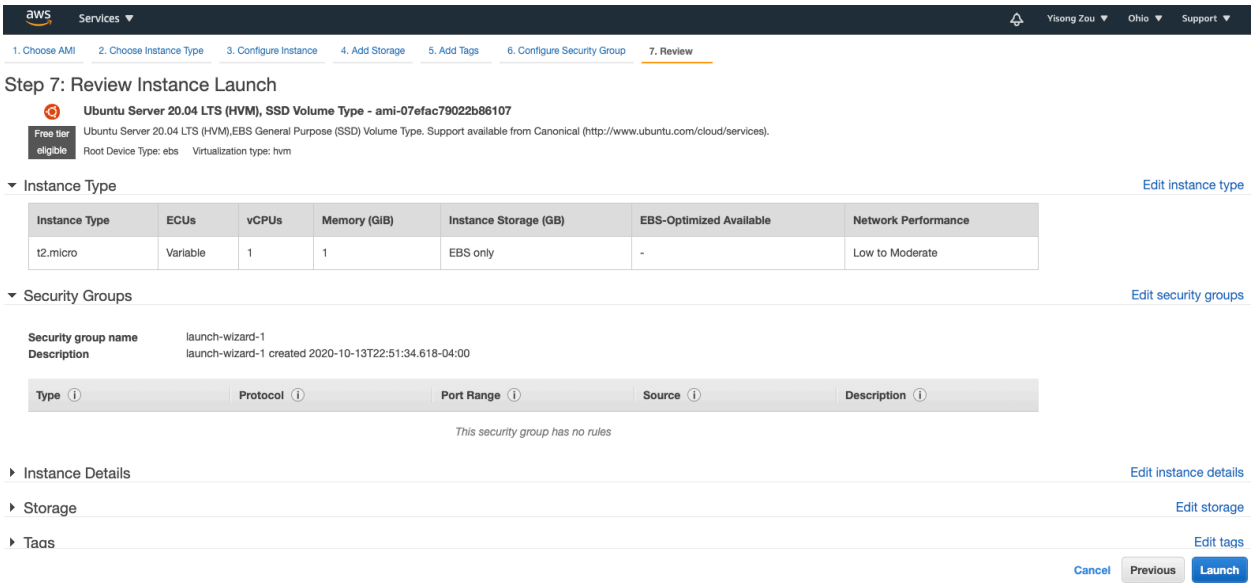
Register AWS Free Tier Account



Reserve an ununtu vm



Launch it



Login into the ubuntu vm using Mac terminal

```
zouyisong@EASSONZOU-MB0 AWS % ssh -i ./ZYSMacbookPro.pem ubuntu@ec2-3-138-181-38.us-east-2.compute.amazonaws.com
The authenticity of host 'ec2-3-138-181-38.us-east-2.compute.amazonaws.com (3.138.181.38)' can't be established.
ECDSA key fingerprint is SHA256:072hnIXMmy/e9emvVroIvkTEMMqmB1NaMhvLPqcqwt0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-138-181-38.us-east-2.compute.amazonaws.com,3.138.181.38' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-1024-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Oct 15 00:14:32 UTC 2020

  System load:  0.0               Processes:             102
  Usage of /:   16.7% of 7.69GB   Users logged in:       0
  Memory usage: 19%               IPv4 address for eth0: 172.31.2.139
  Swap usage:   0%

1 update can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-2-139:~$ ls
```

Make SSH daemon listening on port 60022
sudo perl -pi -e 's/^#?Port 22$/Port 60022/' /etc/ssh/sshd_config
sudo service sshd restart || service ssh restart

Use ssh to login through port 60022

```
zouyisong@EASSONZOU-MB0 AWS % ssh -p 60022 -i ./ZYSMacbookPro.pem ubuntu@ec2-3-138-181-38.us-east-2.compute.amazonaws.com
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-1024-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Oct 15 00:16:07 UTC 2020

  System load:  0.0               Processes:             98
  Usage of /:   16.8% of 7.69GB   Users logged in:       0
  Memory usage: 19%               IPv4 address for eth0: 172.31.2.139
  Swap usage:   0%


1 update can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable


The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Thu Oct 15 00:14:33 2020 from 174.109.74.221
ubuntu@ip-172-31-2-139:~$ ls
ubuntu@ip-172-31-2-139:~$
```

Setup Cowrie

(Resource: https://cowrie.readthedocs.io/en/latest/INSTALL.html)
(https://null-byte.wonderhowto.com/how-to/use-cowrie-ssh-honeypot-catch-attackers-your-network-0181600/)

First do:
sudo apt-get update

Then I followed the steps in the above link to setup cowrie.

Start Cowrie:

```
(cowrie-env) cowrie@ip-172-31-2-139:~/cowrie/bin$ ./cowrie start

Join the Cowrie community at: https://www.cowrie.org/slack/

Using activated Python virtual environment "/home/cowrie/cowrie/cowrie-env"
version check
Starting cowrie: [twistd   --umask=0022 --pidfile=var/run/cowrie.pid --logger cowrie.python.logfile.logger cowrie ]...
(cowrie-env) cowrie@ip-172-31-2-139:~/cowrie/bin$
```

Listening on port 22:
sudo iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 2222

Login to your own honeypot and look around. Can you tell it's simulated?
Yes, because the color of the prompt is weird, showed in the following pic

```
zouyisong@EASSONZOU-MB0 ~ % ssh root@3.138.181.38
root@3.138.181.38's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@svr04:~# ls
root@svr04:~# cd ..
root@svr04:/# ls
bin   boot  dev   etc   home  initrd.img lib   lost+found media  mnt   opt   proc  root
run   sbin  selinux srv   sys   tmp   usr   var   vmlinuz
root@svr04:/#
```

From outside the honeypot, show that you can find logs of your attempted "intrusion".
If I look at the cowrie.log, there will be some intrusions such as unauthorized login

```
cowrie@ip-172-31-2-139:~/cowrie/var/log/cowrie$ cat cowrie.log
2020-10-15T01:40:57.075984Z [-] unauthorized login:
2020-10-15T01:40:57.359681Z [HoneyPotSSHTransport,3,74.112.136.159] connection lost
2020-10-15T01:40:57.359878Z [HoneyPotSSHTransport,3,74.112.136.159] Connection lost after 2 seconds
```

Once you're happy it's set up well, let it run for at least ~3 days or until you get several attacks logged!

Results (4pts)

Take a look at what attackers did. **How many attacks did you capture? What kinds of things did attackers tend to do?**

I kept it running for 5 days and the following are the log files the honeypot got

```
cowrie@ip-172-31-2-139:~/cowrie/var/log/cowrie$ ls
cowrie.json            cowrie.json.2020-10-16  cowrie.json.2020-10-18  cowrie.log            cowrie.log.2020-10-16  cowrie.log.2020-10-18
cowrie.json.2020-10-15  cowrie.json.2020-10-17  cowrie.json.2020-10-19  cowrie.log.2020-10-15  cowrie.log.2020-10-17  cowrie.log.2020-10-19
cowrie@ip-172-31-2-139:~/cowrie/var/log/cowrie$
```

And inside the downloads directory, there are many files transferred from the attacker to the honeypot:

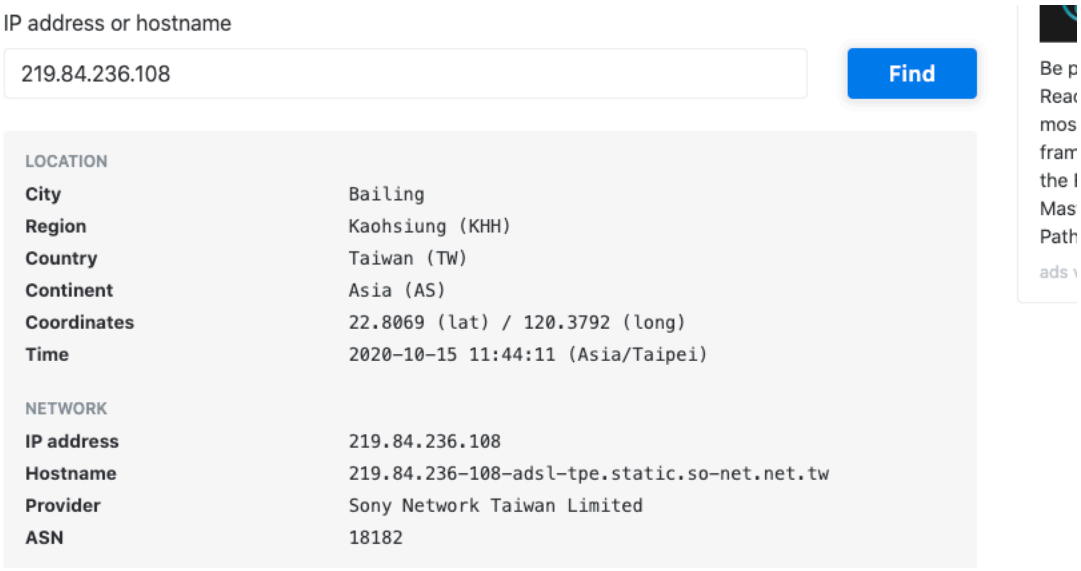
```
cowrie@ip-172-31-2-139:~/cowrie/var/lib/cowrie/downloads$ ls
03700f94e1a06ce38b5cdf3a6dfb5d3362c19d2e625c5251510079d0dad16ef8  tmp08ob5816  tmp_qkemqzd  tmph0iqg8al  tmpqg0qop3u
0667f316202a51aba356033a6dacbbf76eb028139691cc68e943a5726f6b9b57  tmp0wchbq0f  tmpa29la1wd  tmphllp2165  tmpqzv3lvrg
0a49ff64a730565f4d479f098e2e1d1112a3015293c521591f9160e6a831bd2d  tmp19kaxn38  tmpaapaxj67  tmpi4pfzihn  tmprb21jzhv
32120e5912787cc11fa2354325adb068146973d676b1a46d76d00f85463d5987  tmp2h283g4r  tmpabaphw0q  tmpif_s05bu  tmprld0pnpx
3b503e9da73d43a68c97e934a2d099985cae6763bc4aece1095ea1e01e0dcf0c  tmp2qwbesx_  tmpawdo_ozs  tmpikykm_oz  tmpsfof0jto
3e29caf29feb32677210aa2c36792dc25615c90163fdc852e0f3433b93216fa71  tmp36slkfkj  tmpb73tqakm  tmpjbw49djs  tmpskqff7uo
44a72a6b76c0f173ea6c646e1abf3834e57e2835cd998b08b132e06c6adc1dde  tmp4p6oktf8  tmpb7swskoe  tmpjnevoa4f  tmpsnj5g9aq
63062aed0a51177d8bbe2f675e37eab45becef3227ad0df8a041907aa2a6a6cd  tmp51_z_lsv  tmpbaur4lu3  tmpjr3e4ywk  tmpsss9qsun
6e690566943352a4d5cc4f39fa550ff57cd5056be27a0b64be5a8dcc566c2863d  tmp58do8gii  tmpbidsq2gc  tmpm6igkh7t  tmpurbuf5bp
7ef865f59fa8a070886ec71989cde88cba5164f2303fabb647ddfcdbff4c40d5  tmp5bkw06a9  tmpbrtjz8eb  tmpmnvlcabj  tmpuwfbprcl
7f24aa65fc23017f8e20dae80ce444c09bb63b83926f28b6fe900239433e8008  tmp6079lcdj  tmpcaky42zh  tmpn4oqyZfl  tmpv5jwxi8n
7f66ab9527b58de66b4ca11af64719985259c018e3ed0f0f7766567062887e22  tmp62rgmdtg  tmpcr9p3swy  tmpn5tzf4_g  tmpvqh1k8s1
8ac542d6d58eeed4de0fe49d4d1518eb922f409f877ea38bb22f61407f9c97e1  tmp65anmmdd  tmpdfvg_jpb  tmpn8f14udw  tmpwnkvb0tx
97d97134b162fbf7d6a73efcb11451c44bb6787a5c7d4295d5038b4e86965fbc  tmp66277yal  tmpekbo8jy9  tmpnmgdkrz3  tmpwutc52qy
9bee5ecdf19dcee9f9d0b9124c53024a7c04b23c32def8cebe2d8e2aa21c2899  tmp85pcbm3v  tmpeq0mpgwb  tmpnt0ybu7o  tmpxzt83z97
a8460f446be540410004b1a8db4083773fa46f7fe76fa84219c93daa1669f8f2  tmp8i8i8nvc  tmpf4qqal2l  tmpo4nx6f34  tmpy15zix2w
b30ce023323e95abd9d8eeb0b12edd4e31703c3bbeec2e3ce368083cbf7180bb  tmp8pfk8s7c  tmpfg8833kz  tmpocc8d270  tmpygnlib19
b61e7b83938aa5fb1d3a19d6e5016a630060031490f9190669133ad3cf0d7bb5  tmp8xuuyvdt  tmpfqgpllnl  tmppe2coizd  tmpyhwonuzn
dbd2c4175b961016101a19c3cf8c633094b64ccb33da0f9c7b267272810 6d49c  tmp9_7e9wo_  tmpg2p8mqtv  tmpphbrdlm4  tmpyiwilhd6
edd604cd0cbdd7c5c0c3a747e9314c51c34b452c18c5053d11b24789b08a7699  tmp9mojgwyp  tmpgozidd6w  tmppj4dpkj7
tmp0037ctmk                                                       tmp_hlb_4__  tmpgqe01m63  tmppxkzir8s
```

Look into the file var/log/cowrie/cowrie.log
use the command cat cowrie.log | grep 'HoneyPotSSHTransport'
I can find out that there are about hundreds of attacks each day.

Pick a particular attack and describe:

```
2020-10-15T02:46:22.447139Z [HoneyPotSSHTransport,7,219.84.236.108] Remote SSH version: b'SSH-2.0-OpenSSH_7.9p1 Raspbian-10'
2020-10-15T02:46:22.464767Z [HoneyPotSSHTransport,8,219.84.236.108] Remote SSH version: b'SSH-2.0-OpenSSH_7.9p1 Raspbian-10'
2020-10-15T02:46:22.659428Z [HoneyPotSSHTransport,7,219.84.236.108] SSH client hassh fingerprint: ec7378c1a92f5a8dde7e8b7a1ddf33d1
2020-10-15T02:46:22.660682Z [HoneyPotSSHTransport,7,219.84.236.108] kex alg, key alg: b'curve25519-sha256' b'ssh-rsa'
2020-10-15T02:46:22.660768Z [HoneyPotSSHTransport,7,219.84.236.108] outgoing: b'aes128-ctr' b'hmac-sha2-512' b'none'
2020-10-15T02:46:22.660832Z [HoneyPotSSHTransport,7,219.84.236.108] incoming: b'aes128-ctr' b'hmac-sha2-512' b'none'
2020-10-15T02:46:22.664436Z [HoneyPotSSHTransport,8,219.84.236.108] SSH client hassh fingerprint: ec7378c1a92f5a8dde7e8b7a1ddf33d1
2020-10-15T02:46:22.665461Z [HoneyPotSSHTransport,8,219.84.236.108] kex alg, key alg: b'curve25519-sha256' b'ssh-rsa'
2020-10-15T02:46:22.665532Z [HoneyPotSSHTransport,8,219.84.236.108] outgoing: b'aes128-ctr' b'hmac-sha2-512' b'none'
2020-10-15T02:46:22.665591Z [HoneyPotSSHTransport,8,219.84.236.108] incoming: b'aes128-ctr' b'hmac-sha2-512' b'none'
2020-10-15T02:46:23.076101Z [HoneyPotSSHTransport,8,219.84.236.108] NEW KEYS
2020-10-15T02:46:23.094803Z [HoneyPotSSHTransport,7,219.84.236.108] NEW KEYS
2020-10-15T02:46:23.276345Z [HoneyPotSSHTransport,8,219.84.236.108] starting service b'ssh-userauth'
2020-10-15T02:46:23.307332Z [HoneyPotSSHTransport,7,219.84.236.108] starting service b'ssh-userauth'
2020-10-15T02:46:23.476321Z [SSHService b'ssh-userauth' on HoneyPotSSHTransport,8,219.84.236.108] b'pi' trying auth b'none'
2020-10-15T02:46:23.519388Z [SSHService b'ssh-userauth' on HoneyPotSSHTransport,7,219.84.236.108] b'pi' trying auth b'none'
2020-10-15T02:46:23.676168Z [SSHService b'ssh-userauth' on HoneyPotSSHTransport,8,219.84.236.108] b'pi' trying auth b'password'
2020-10-15T02:46:23.676492Z [SSHService b'ssh-userauth' on HoneyPotSSHTransport,8,219.84.236.108] Could not read etc/userdb.txt, default database activated
2020-10-15T02:46:23.676634Z [SSHService b'ssh-userauth' on HoneyPotSSHTransport,8,219.84.236.108] login attempt [b'pi'/b'raspberry'] failed
2020-10-15T02:46:23.731346Z [SSHService b'ssh-userauth' on HoneyPotSSHTransport,7,219.84.236.108] b'pi' trying auth b'password'
2020-10-15T02:46:23.731635Z [SSHService b'ssh-userauth' on HoneyPotSSHTransport,7,219.84.236.108] Could not read etc/userdb.txt, default database activated
2020-10-15T02:46:23.731811Z [SSHService b'ssh-userauth' on HoneyPotSSHTransport,7,219.84.236.108] login attempt [b'pi'/b'raspberryraspberry993311'] failed
2020-10-15T02:46:24.678526Z [-] b'pi' failed auth b'password'
2020-10-15T02:46:24.678748Z [-] unauthorized login:
2020-10-15T02:46:24.733383Z [-] b'pi' failed auth b'password'
2020-10-15T02:46:24.733535Z [-] unauthorized login:
2020-10-15T02:46:24.880124Z [HoneyPotSSHTransport,8,219.84.236.108] connection lost
2020-10-15T02:46:24.880323Z [HoneyPotSSHTransport,8,219.84.236.108] Connection lost after 2 seconds
2020-10-15T02:46:24.947377Z [HoneyPotSSHTransport,7,219.84.236.108] connection lost
2020-10-15T02:46:24.947568Z [HoneyPotSSHTransport,7,219.84.236.108] Connection lost after 2 seconds
```

- **The network origin (IP address organization and geolocation)**



- **The steps carried out**

It first tries to login using the password raspberry to ssh login, failed, then used the password raspberryraspberry993311 to ssh login and failed.

- **Does it appear to be automated or manual?**

It appears to be automated.

- **What appears to have been the attacker's goal?**

The attacker just wants to get into the linux vm

- **If they downloaded file(s), analyze these**

They didn't download files

Question 7: Denial of Service attack using TorsHammer (6 points)

Let's do a small DOS attack from your Kali VM to your Linux VM using TorsHammer, a slow POST attack similar to the slowloris attack we discussed in class.

On your **Kali VM**, download the TorsHammer from https://sourceforge.net/projects/torshammer/

To launch an attack, use the torshammer.py file and pass the necessary parameters to it.

```
# ./torshammer.py
```

On your **Linux VM**, If you haven't already, install the Apache web server and ensure it's working on port 80.
(source: https://ubuntu.com/tutorials/install-and-configure-apache#4-setting-up-the-virtualhost-configuration-file)

From the Kali VM, attack your Linux VM. Use 512 threads. Show the command used.

./torshammer.py -t vcm-17149.vm.duke.edu -r 512

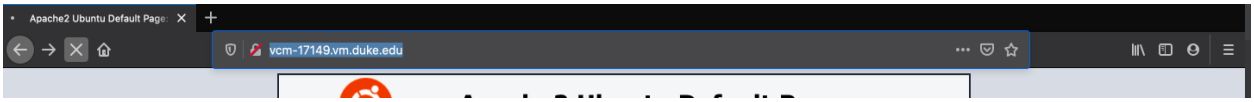
```
[yz558@kali Torshammer 1.0 10:24 PM]$  ./torshammer.py -t vcm-17149.vm.duke.edu -r 512

/*
 * Tor's Hammer
 * Slow POST DoS Testing Tool
 * Version 1.0 Beta
 * Anon-ymized via Tor
 * We are Anonymous.
 * We are Legion.
 * We do not forgive.
 * We do not forget.
 * Expect us!
 */


/*
 * Target: vcm-17149.vm.duke.edu Port: 80
 * Threads: 512 Tor: False
 * Give 20 seconds without tor or 40 with before checking site
Connected to host...
Posting: 0
Posting: Z
```

From your local machine's web browser, try to navigate to the Linux VM. Refresh the page a few times; what do you observe?

I observed that the website is sometimes slowly loading, which will not happen before the attack.

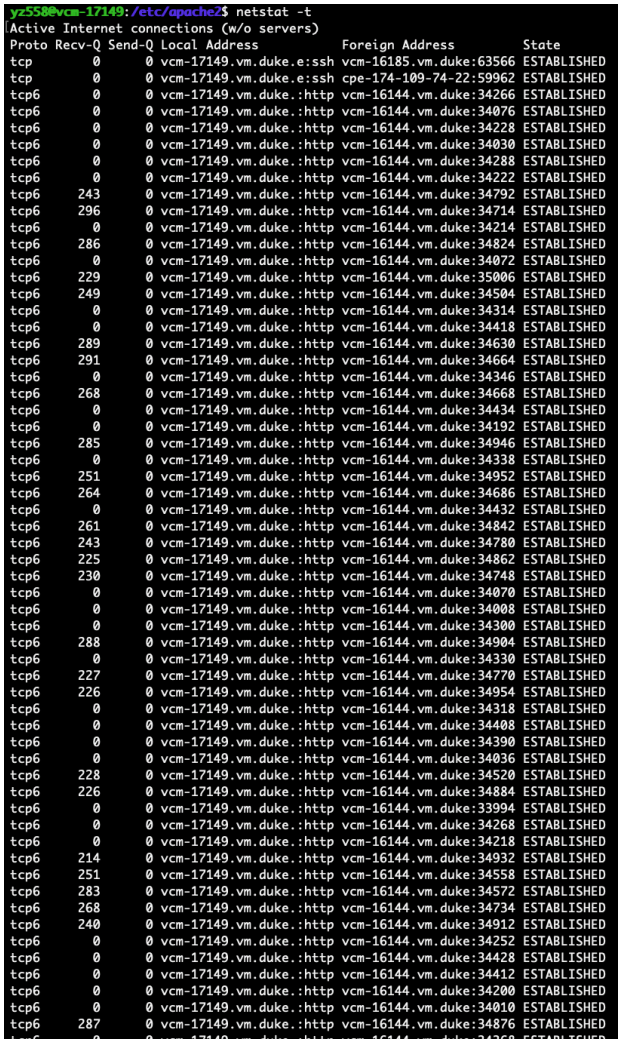


Show the output of "`netstat -t`" (list TCP connections) before versus during an attack.

Before:

```
yz558@vcm-17149:/etc/apache2$ netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 vcm-17149.vm.duke.e:ssh vcm-16185.vm.duke:63566 ESTABLISHED
tcp        0    200 vcm-17149.vm.duke.e:ssh cpe-174-109-74-22:59962 ESTABLISHED
```

During:



```
yz558@vcm-17149:/etc/apache2$ netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 vcm-17149.vm.duke.e:ssh vcm-16185.vm.duke:63566 ESTABLISHED
tcp        0      0 vcm-17149.vm.duke.e:ssh cpe-174-109-74-22:59962 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34266 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34076 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34228 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34030 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34288 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34222 ESTABLISHED
tcp6     243      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34792 ESTABLISHED
tcp6     296      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34714 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34214 ESTABLISHED
tcp6     286      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34824 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34072 ESTABLISHED
tcp6     229      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:35006 ESTABLISHED
tcp6     249      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34504 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34314 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34418 ESTABLISHED
tcp6     289      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34630 ESTABLISHED
tcp6     291      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34664 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34346 ESTABLISHED
tcp6     268      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34668 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34434 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34192 ESTABLISHED
tcp6     285      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34946 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34338 ESTABLISHED
tcp6     251      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34952 ESTABLISHED
tcp6     264      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34686 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34432 ESTABLISHED
tcp6     261      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34842 ESTABLISHED
tcp6     243      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34780 ESTABLISHED
tcp6     225      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34862 ESTABLISHED
tcp6     230      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34748 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34070 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34008 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34300 ESTABLISHED
tcp6     288      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34904 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34330 ESTABLISHED
tcp6     227      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34770 ESTABLISHED
tcp6     226      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34954 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34318 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34408 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34390 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34036 ESTABLISHED
tcp6     228      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34520 ESTABLISHED
tcp6     226      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34884 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:33994 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34268 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34218 ESTABLISHED
tcp6     214      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34932 ESTABLISHED
tcp6     251      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34558 ESTABLISHED
tcp6     283      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34572 ESTABLISHED
tcp6     268      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34734 ESTABLISHED
tcp6     240      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34912 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34252 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34428 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34412 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34200 ESTABLISHED
tcp6       0      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34010 ESTABLISHED
tcp6     287      0 vcm-17149.vm.duke.:http vcm-16144.vm.duke:34876 ESTABLISHED
```

Use "`top`" to assess CPU usage before versus during the attack. Does CPU usage increase significantly? Why or why not?

Before

```
top - 23:00:39 up 11 days,  1:35,  2 users,  load average: 0.10, 0.03, 0.01
Tasks: 160 total,   1 running, 159 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.0 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3715.9 total,   2464.4 free,    311.1 used,    940.4 buff/cache
MiB Swap:    976.0 total,    976.0 free,      0.0 used.   3164.5 avail Mem
```

During

```
top - 23:00:04 up 11 days,  1:34,  2 users,  load average: 0.00, 0.00, 0.00
Tasks: 160 total,   1 running, 159 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.3 sy,  0.0 ni, 99.0 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
MiB Mem :   3715.9 total,   2449.8 free,    325.9 used,    940.2 buff/cache
MiB Swap:    976.0 total,    976.0 free,      0.0 used.   3149.7 avail Mem
```

There isn't a large change in CPU usage before and after the attack.
This is because the resource under attack is the web connection limit. However, the CPU isn't active for most of the time, it's just waiting for the next byte of request to come in.
The post requests is very slow.

In another terminal from the Kali VM (or another Linux machine), use the `time` and `curl` commands a few times to roughly gauge the latency of HTTP responses before, during, and after the attack.

```
curl -s -w %{time_total}\\n -o /dev/null http:// vcm-17149.vm.duke.edu
```

Here I tested three times for each condition

Before
0.005717s 0.015189s 0.004892s

During
12.572629s 13.943034s 15.659671s

After
0.004212s 0.014081s 0.005724s

Question 8: Shell practice (9 points)

For each of the following questions, give the command(s) and output. If the output is more than a few lines, you may truncate it.

Web file hash (2 points)

Develop a shell command to find the SHA 512 hash of the JPEG picture of the instructor's dog, found on this page. To help check your work, the last byte is 0x6d. Show the command and its output. A properly fancy solution will avoid the need to write any files to disk.

Command:

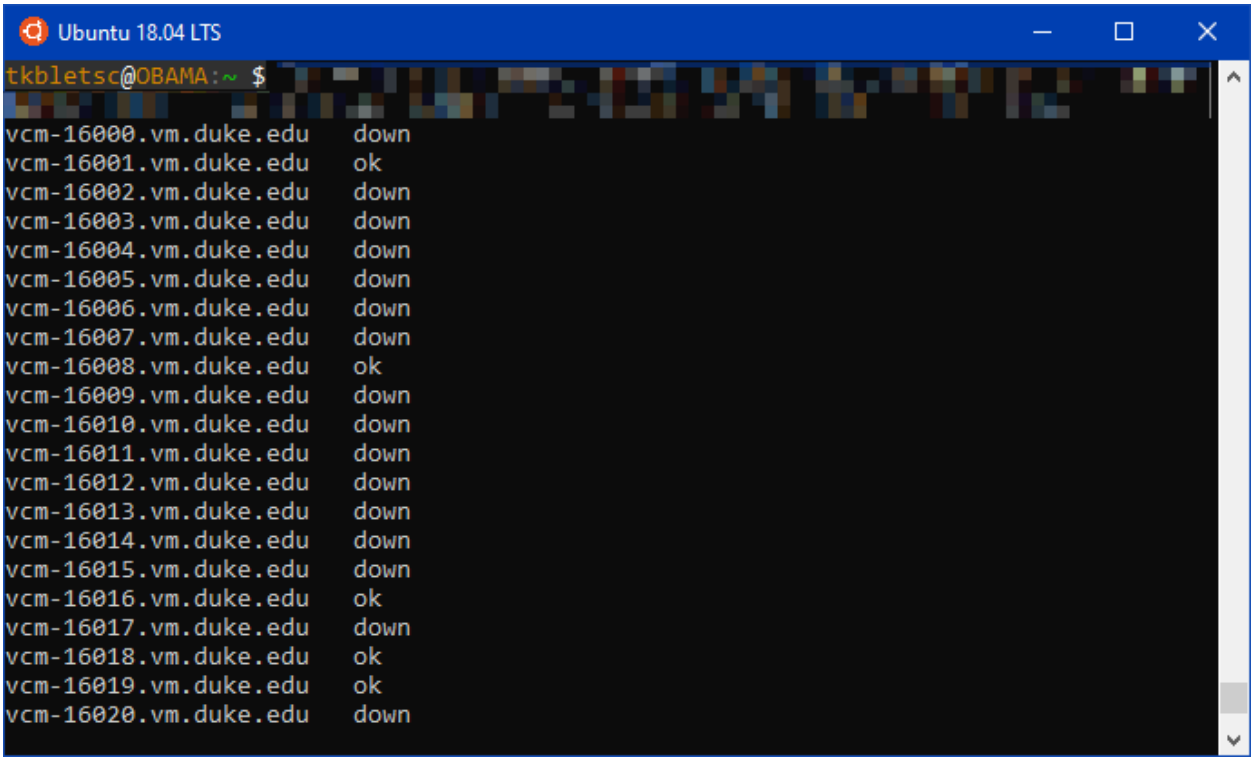
wget -qO- http://people.duke.edu/~tkb13/images/reg-chair.jpg | sha512sum

Output:

```
yz558@vcm-17149:~/ECE560-Computer-and-Information-Security/ECE560_Security/hw4$ wget -qO- http://people.duke.edu/~tkb13/images/reg-chair.jpg | s
ha512sum
9848f6b4ae43805c111f76eead20e32aecf2e3b5a00540eb09aba74acdedd9c18acb30649c2c14a4e232539fddb29d1399ca3d6f006d2ddf603d89b26a991c6d  -
```

Ping check (2 points)

Develop a shell command or shell script (max 5 lines) to **`ping`** several VCM VMs with a single packet with a timeout of 1 second with an output of simply "ok" or "down" for each host, one per line. The range of hosts to check is vcm-16000.vm.duke.edu through vcm-16100.vm.duke.edu (101 hosts total). Shorter solutions are preferred to longer ones. Example output (higher numbered hosts omitted for space):



Command:

Use the following script(Only two lines):

```
#! /bin/bash
for varible1 in {16000..16100};do ping -c 1 -W 1 vcm-$varible1.vm.duke.edu > /dev/null;if [ $? -eq 0 ]; then echo "vcm-$varible1.vm.duke.edu up\
";else echo "vcm-$varible1.vm.duke.edu down";fi; done
```

```
1. #! /bin/bash

2. for varible1 in {16000..16100};do ping -c 1 -W 1 vcm-
   $varible1.vm.duke.edu > /dev/null;if [ $? -eq 0 ]; then echo "vcm-
   $varible1.vm.duke.edu up";else echo "vcm-$varible1.vm.duke.edu down";fi; done
```

Output:

```
yz558@vcm-17149:~/ECE560-Computer-and-Information-Security/ECE560_Security/hw1$ ./ping.sh
vcm-16000.vm.duke.edu down
vcm-16001.vm.duke.edu down
ping: vcm-16002.vm.duke.edu: Name or service not known
vcm-16002.vm.duke.edu down
vcm-16003.vm.duke.edu down
vcm-16004.vm.duke.edu down
vcm-16005.vm.duke.edu down
vcm-16006.vm.duke.edu down
vcm-16007.vm.duke.edu down
vcm-16008.vm.duke.edu up
vcm-16009.vm.duke.edu down
vcm-16010.vm.duke.edu down
vcm-16011.vm.duke.edu down
vcm-16012.vm.duke.edu down
ping: vcm-16013.vm.duke.edu: Name or service not known
vcm-16013.vm.duke.edu down
vcm-16014.vm.duke.edu down
ping: vcm-16015.vm.duke.edu: Name or service not known
```

Binary file analysis (3 points)

Using **`strings`**, and **`hd`**, let's start analyzing **`cryptotest.pyc`** from the earlier "Simple Encryption Program" question for possible reverse engineering. Answer each of the following questions and show the command(s)/output that led you to your answer.

- What message will likely be printed if the tool is able to compress the cipher text too much?

Here I used the command:

strings cryptotest.pyc |grep compress

The output is:

```
yz558@vcm-17149:~/ECE560-Computer-and-Information-Security/ECE560_Security/hw1$ strings cryptotest.pyc |grep compress
compress
compressedr
get_compression_ratioB
Encryption exit status == 0?zCNon-zero status indicates an error where none should have occurred.z Calculating compression ratio...z
Cipher compressability > 95%?rG
z4The cipher file is too compressable (Ratio: %.2f%%).
```

So from the following output, if the tool is able to compress the cipher too much, it will likely print:

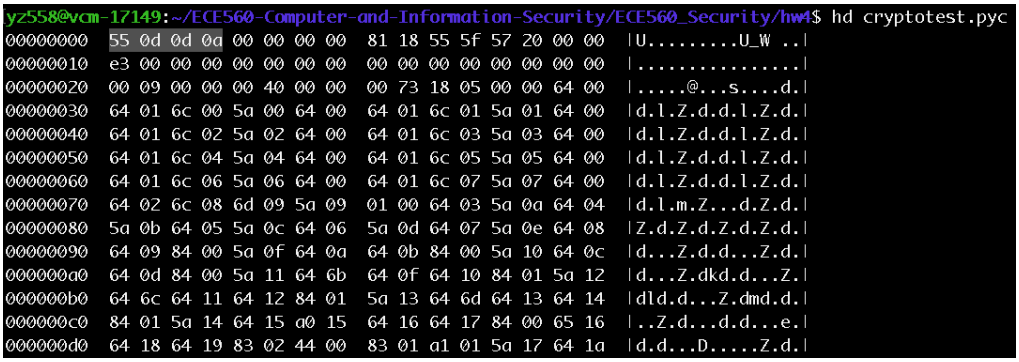
The cipher file is too compressable

- What is the “magic number” of the file, as described here?

As the magic number is the first fout bytes of the file, I use the following command:

hd cryptotest.pyc

The output is:



So the magic number is:

550d0d0a

Bulk hash (2 points)

On your Kali VM, develop a shell command that will print the MD5 hash of every `.py` file under `/usr/lib/python3.7`.

Command:

md5sum *.py

Output:

Question 9: Analyze forensic server packet capture network logs (10 points)

Below is a network capture from an attack on a real Linux server. The capture was created by setting up a new CentOS 5.9 Linux server, turning off the firewall, and setting the root password to root. The server was compromised within a few hours. Before it was put online, Wireshark was run and configured to capture the network event before, during, and after the compromise.

The network captures are located here:
http://people.duke.edu/~tkb13/fixed/iasg_capture_files.tgz

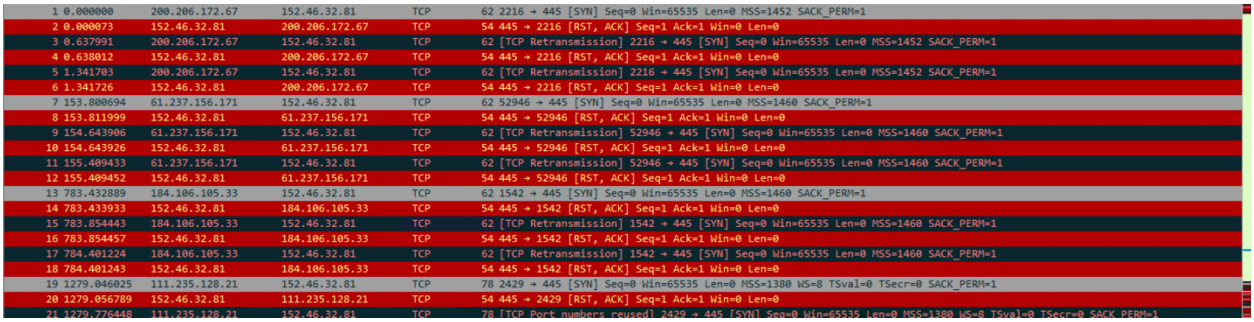
Download this file to your home directory and analyze it there.

Using tcpdump or Wireshark, analyze the packet capture files from that attack and describe/show the following activities or small samples:

1. **Reconnaissance**

Recon is things like portscans, connection attempts, etc. - anything to gather info about the machine and others like it.

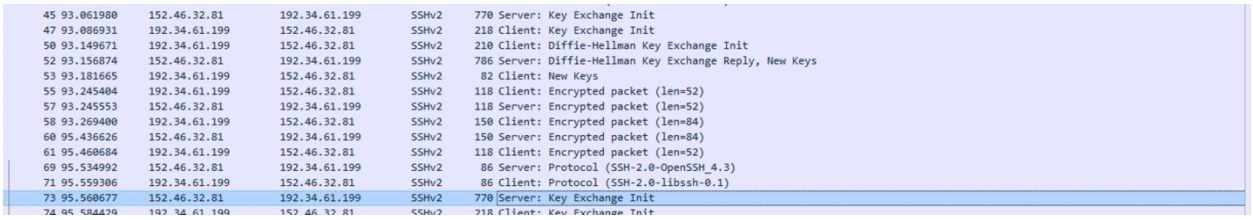
The following is an example of port scan



1	0.000000	200.206.172.67	152.46.32.81	TCP	62 2216 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1452 SACK_PERM=1
2	0.000073	152.46.32.81	200.206.172.67	TCP	54 445 → 2216 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	0.637991	200.206.172.67	152.46.32.81	TCP	62 [TCP Retransmission] 2216 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1452 SACK_PERM=1
4	0.638012	152.46.32.81	200.206.172.67	TCP	54 445 → 2216 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	1.341703	200.206.172.67	152.46.32.81	TCP	62 [TCP Retransmission] 2216 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1452 SACK_PERM=1
6	1.341726	152.46.32.81	200.206.172.67	TCP	54 445 → 2216 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7	153.800694	61.237.156.171	152.46.32.81	TCP	62 52946 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
8	153.811999	152.46.32.81	61.237.156.171	TCP	54 445 → 52946 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9	154.643906	61.237.156.171	152.46.32.81	TCP	62 [TCP Retransmission] 52946 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
10	154.643926	152.46.32.81	61.237.156.171	TCP	54 445 → 52946 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11	155.409433	61.237.156.171	152.46.32.81	TCP	62 [TCP Retransmission] 52946 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
12	155.409452	152.46.32.81	61.237.156.171	TCP	54 445 → 52946 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13	783.432889	184.106.105.33	152.46.32.81	TCP	62 1542 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
14	783.433933	152.46.32.81	184.106.105.33	TCP	54 445 → 1542 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15	783.854443	184.106.105.33	152.46.32.81	TCP	62 [TCP Retransmission] 1542 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
16	783.854457	152.46.32.81	184.106.105.33	TCP	54 445 → 1542 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
17	784.401224	184.106.105.33	152.46.32.81	TCP	62 [TCP Retransmission] 1542 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
18	784.401243	152.46.32.81	184.106.105.33	TCP	54 445 → 1542 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19	1279.046025	111.235.128.21	152.46.32.81	TCP	78 2429 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1380 WS=8 TSval=0 TSecr=0 SACK_PERM=1
20	1279.056789	152.46.32.81	111.235.128.21	TCP	54 445 → 2429 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
21	1279.776448	111.235.128.21	152.46.32.81	TCP	78 [TCP Port numbers reused] 2429 → 445 [SYN] Seq=0 Win=65535 Len=0 MSS=1380 WS=8 TSval=0 TSecr=0 SACK_PERM=1

2. **Actual SSH Attacks**

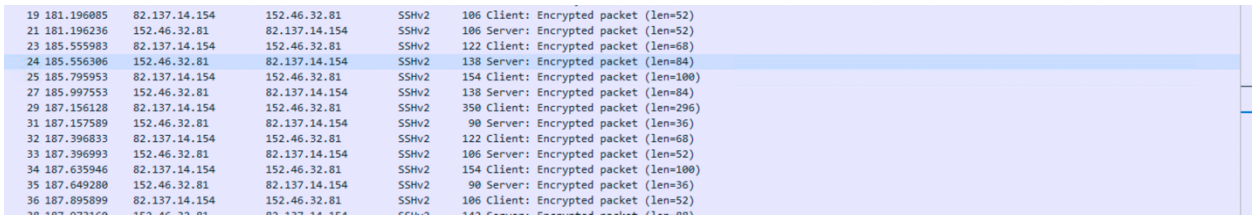
Started from iasgcap_00008, there started to be ssh and there are many times of Key Exchanges which seems to be attackers trying to connect



45	93.061980	152.46.32.81	192.34.61.199	SSHv2	770 Server: Key Exchange Init
47	93.086931	192.34.61.199	152.46.32.81	SSHv2	218 Client: Key Exchange Init
50	93.149671	192.34.61.199	152.46.32.81	SSHv2	210 Client: Diffie-Hellman Key Exchange Init
52	93.156874	152.46.32.81	192.34.61.199	SSHv2	786 Server: Diffie-Hellman Key Exchange Reply, New Keys
53	93.181665	192.34.61.199	152.46.32.81	SSHv2	82 Client: New Keys
55	93.245404	192.34.61.199	152.46.32.81	SSHv2	118 Client: Encrypted packet (len=52)
57	93.245553	152.46.32.81	192.34.61.199	SSHv2	118 Server: Encrypted packet (len=52)
58	93.269400	192.34.61.199	152.46.32.81	SSHv2	150 Client: Encrypted packet (len=84)
60	95.436626	152.46.32.81	192.34.61.199	SSHv2	150 Server: Encrypted packet (len=84)
61	95.460684	192.34.61.199	152.46.32.81	SSHv2	118 Client: Encrypted packet (len=52)
69	95.534992	152.46.32.81	192.34.61.199	SSHv2	86 Server: Protocol (SSH-2.0-OpenSSH_4.3)
71	95.559306	192.34.61.199	152.46.32.81	SSHv2	86 Client: Protocol (SSH-2.0-libssh-0.1)
73	95.560677	152.46.32.81	192.34.61.199	SSHv2	770 Server: Key Exchange Init
74	95.584429	192.34.61.199	152.46.32.81	SSHv2	218 Client: Key Exchange Init

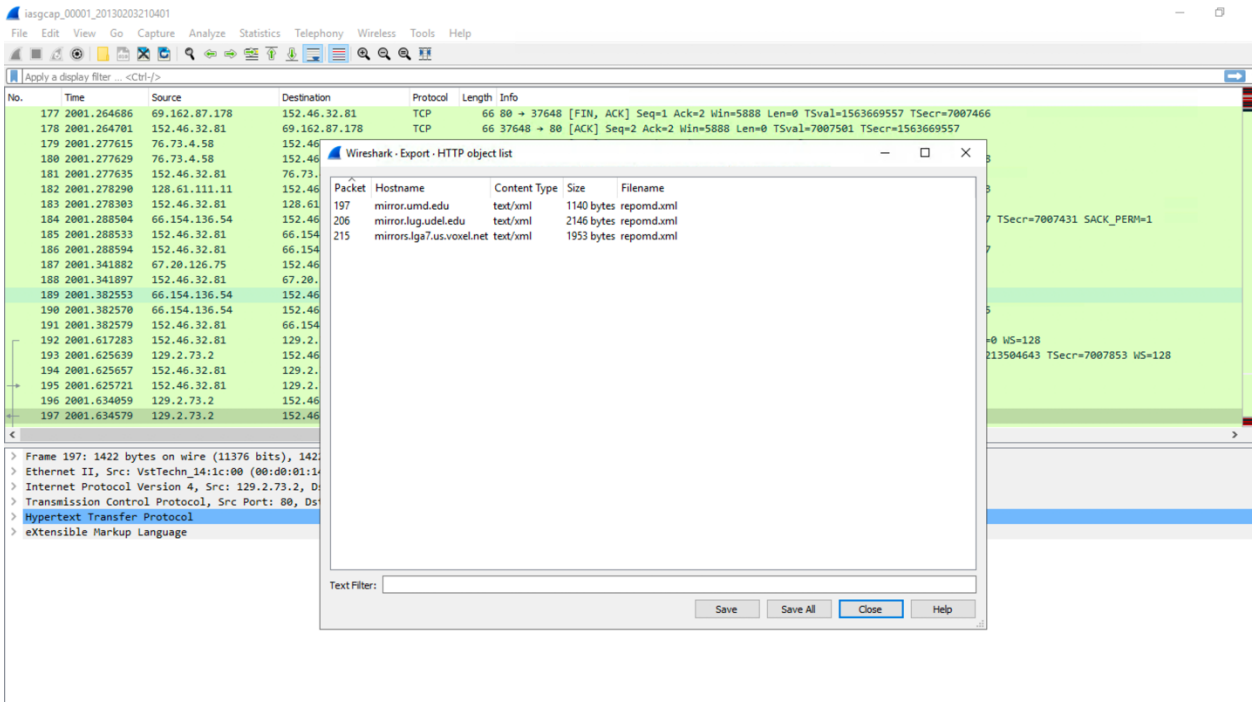
3. **Successful SSH authentication**

Started from test_000000 which is extracted from iasgcap_00012, there are not so many key exchanges. Instead, there are constant packet communications between client and the server on port 22



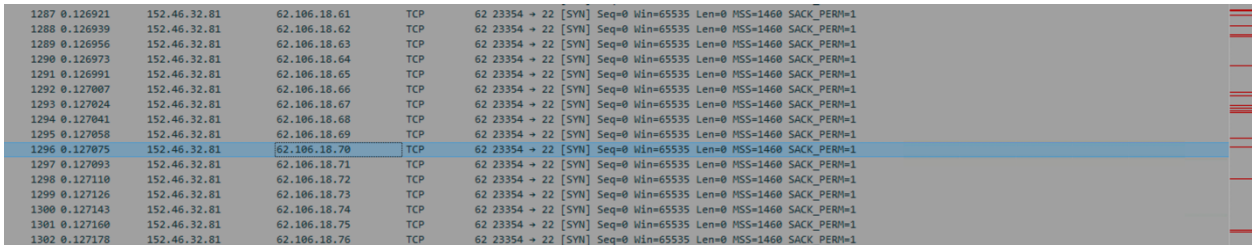
4. External Tools Downloaded

In test_000001 which is extracted from iasgcap_00012, use export objects we can see there are three xml files downloaded



5. External Target attacks

For the following inside test 00073, the source continuously attacks different external targets.



The answers will not be obvious or clear-cut. You will need to use inferences and your own best judgment. For example, SSH is encrypted, so you won't see actual successful SSH logins take place. However, from volume and timing of traffic, you can make an informed guess with reasonable justification; that's what we're looking for.

Tip: Some of the packet capture files are very large and you will need to break them into smaller chunks before you analyze them. A good working size is around 100,000 packets per file. Sample Pcap Split on Windows:

```
"c:\Program Files\Wireshark\editcap.exe" -c 100000 iasgcap_00012_20130204080402 test
```

Note: Credit for this dataset goes to Samuel Carter at NCSU.

Question 10: Analyze forensic server hacker package (8 points)

Analyze the **gosh.tar.gz** package downloaded as part of the server attack described above. The gosh.tar.gz file is located here:
https://people.duke.edu/~tkb13/courses/ece560/homework/hw4/gosh.tar.bz2

Just download a copy of it to your Linux VM to perform the analysis.

Explain what each file in the package is/does/used for and how they are related (if applicable).

Tips:

- Use **`file`** to figure out what kinds of files these are (text files, shell scripts, executables (ELF binaries), etc.).
- For shell scripts, read them without executing.
- For binary executables, your first step would usually be to set up a sandbox in a throwaway VM for initial analysis, but to save you some work: the binaries are safe to run without arguments. However, **don't run the executable files with any arguments or it will start attacking!** Running without arguments will actually give you a little bit of usage information.
- You can feed any file here to virustotal.com, which will scan it with every common malware scanner and give you a report. For some files, it may even have community info (postings by security researchers about the file).
- On 64-bit Ubuntu-based VMs, you'll need to install some 32-bit support files to run the binaries; see here for info. Without this step, such binaries will give a cryptic "file not found" error on running.
- You may want to use Google Translate to understand some of the messages. Some of the language is Romanian and sometimes explicit.

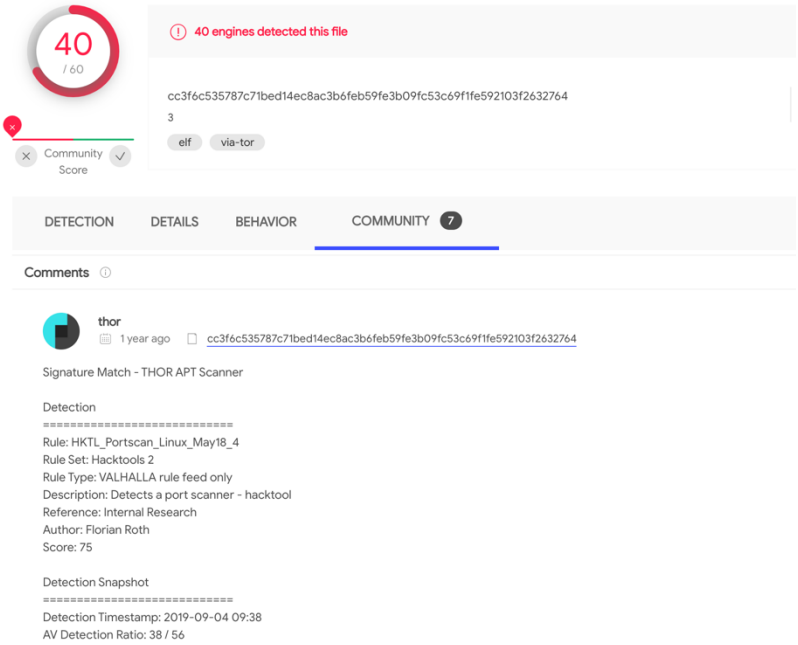
All the files inside:

```
yz558@vcm-17149:~/ECE560-Computer-and-Information-Security/ECE560_Security/hw4/gosh$ ls
1  2  3  4  5  a  common  gen-pass.sh  go.sh  mfu.txt  pass_file  pscan2  scam  secure  ss  ssh-scan  vuln.txt
```

1: User names followed by possible passwords
2: User names followed by possible passwords
3: User names followed by possible passwords
4: User names followed by possible passwords
5: Possible passwords of the user root
a: Print slogans as well as some Romanian words which means: "I'M TRYING TO GIVE CYBERNETIC LIFE", and do some stuffs try to find out if the current attack works well.
common: Some common passwords
gen-pass.sh: Generate password for users and store them inside pass_file
go.sh: run the ss scan the botnets and store the sorted unique IP addresses from bios.txt inside mfu.txt, after that run the ssh-scan file to do SSH login brute hack, and after that remove the bios.txt without any prompt.
mfu.txt: Store the sorted unique IP addresses from bios.txt
pass_file: The user and passwords generated by the script gen-pass.sh

scam: Do the scanner stuff. As the content in it says: “SCANNER USED ONLY BY TEAMUL MaLaSorTe”, “THE SACNER CONTAINS A PASS_FLIE OF 3MEGA”
secure: Shell script to check if the user is now root and if it is root he can rename /usr/bin/mail to /usr/bin/s8 and do scan stuff.
vuln.txt: An empty txt file used to store the message content of the email to send in the shell script file scam.
pscan2: An ELF 32-bit LSB executable, which is a ports canner malware according to the virustotal.com

```
yz558@vcm-17149:~/ECE560-Computer-and-Information-Security/ECE560_Security/hw4/gosh$ ./pscan2
Usage: ./pscan2 <b-block> <port> [c-block]
```



ss: malware to be used by the IoT hacker to scan their botnets

```
yz558@vcm-17149:~/ECE560-Computer-and-Information-Security/ECE560_Security/hw4/gosh$ ./ss
usage: ./ss <port> [-a <a class> | -b <b class>] [-i <interface] [-s <speed>]
speed 10 -> as fast as possible, 1 -> it will take bloody ages (about 50 syns/s)
```

unixfreaxjp
2 years ago
97093a1ef729cb954b2a63d7ccc304b18d0243e2a77d87bbbb94741a0290d762

Malware verdict:
Name: Linux/SS (AV calls it Shark) Category: ELF / Hacktool Purpose: used by the IoT hacker to scan their botnets. Analysis: unixfreaxjp / malwaremustdie.org
uses linux C sources:
libnet.h,stdio.h,socket.h,in.h, inet.h, types.h,unistd.h, pcap.h, time.h
usage:
usage: %s <port> [-a <a class> | -b <b class>] [-i <interface] [-s <speed>]\n speed 10 -> as fast as possible, 1 -> it will take bloody ages (about 50 syns/s)\n
do the syn scan:
struct in_addr sources;myip=libnet_get_ipaddr4(l); sources.s_addr=myip; sprintf(filter,"(tcp[tcpflags]=0x12) and (src port %d) and (dst port %d)",port,sport); printf("using \"%s\" as pcap filter\n",filter); printf("my detected ip on %s is %s\n",l->device,inet_ntoa(sources)); pcap_lookupnet(l->device, &net, &mask, errbuf);
sniffing packet, append bios.txt:
struct in_addr someaddr; struct pcap_pkthdr header; const unsigned char *packet; struct pcap_pkthdr header; while (TRUE_) {packet = pcap_next(handle, &header); memcpy(&someaddr.s_addr,packet+26,4); printf("%s\n",inet_ntoa(someaddr)); FILE * fp;fp=fopen("bios.txt","a+");//append fprintf(fp,"%s\n",inet_ntoa(someaddr));fclose(fp);}
capture:
libnet_t *var0; libnet_ptag_t var1; if(pid > 0) {printf("capturing process started pid %d\n",pid);usleep(500000); while(TRUE_) {var1=LIBNET_PTAG_INITIALIZER; var1=libnet_build_tcp_options(tcpopt, 8, var0, 0); var1=libnet_build_tcp(sport,port,rand(),rand(),TH_SYN,65535,0,0,LIBNET_TCP_H+8,NULL,0,l,0); if(rclass) dstip=rand(); if(aclass) { if(d==0) printf("scanning %d.%d.%d.*\n",a,b,c);d++;} if(d>255) {c++;d=0;} if(c>255) {b++;c=0;} sprintf(ip,"%d.%d.%d.%d\n",a,b,c,d);//report
and print the stuff:
if((b==255)&& (c==255) && (d==255)) { sleep(10);kill(pid,2);return 0;} sc.s_addr=inet_addr(ip); dstip=sources.s_addr; if(bclass) { if(d==0) printf("scanning %d.%d.%d.*\n",a,b,c); d++;} if(d>255) {c++;d=0;} // ====write here================ sprintf(ip,"%d.%d.%d.%d",a,b,c,d); //====end write===unixfreaxjp==== if((c==255) && (d==255)){ printf("%s\n",ip); //or here sleep(10);kill(pid,2);return 0;} #MalwareMustDie!

ssh-scan: ELF SSH login brute hack tool called "scanssh" or "ssh_scan" according to the virustotal.com

```
yz558@vcm-17149:~/ECE560-Computer-and-Information-Security/ECE560_Security/hw4/gosh$ ./ssh-scan
./ssh-scan <cate pizde sa incerc...>
```

unixfreaxjp

6 years ago 93df64cc0ff902ad1e80ada56023610ec2c44c3ecde2d36d37a3a748c7fd42bd

ELF SSH login bruter hack tool called "scanssh" or "ssh_scan"
Analysis: http://blog.malwaremustdie.org/2014/05/a-payback-to-ssh-bruting-crooks.html
$MalwareMustdie

Question 11: Hardware level attacks (4 points)

Read PoC||GTFO 04:10 (“Forget Not the Humble Timing Attack”).

a. Explain how the attack on the hard drive enclosure was able to reduce the search space from 1,000,000 attempts to 60.

Because we can measure the time between when the microcontroller has read in the entire PIN and when the LED goes on, which is a time attack. After the last button was pressed, we fix other 5 digits and change a single digit from 0 to 9 for test. If a try among those has the correct number, the delay will be longer. So we can try 10 times for each and 60 attempts in total. And if we just brute force all the combinations, it will take 1,000,000 attempts.

b. How is the TinySafeBoot firmware “better” than the hard drive enclosure?

This can avoid timing attack because it will not tell the user if his input is correct using the ‘Wrong pin’ LED. When a wrong password is entered jumps into an endless loop, effectively avoiding providing information that would be useful for a timing attack.

c. A “side channel” attack is one where we use a normally-ignored side effect as useful information about a target. What is the side channel used to defeat the TinySafeBoot firmware despite the defense referred to (b) above?

There is a significant difference between the power signatures of the correct and wrong password guesses. And the attacker can use this difference to do side channel attack.

d. What standard password-handling technique would have defeated the attacks described? Why wasn’t the above technique deployed in these cases? Hint: what is the code storage capacity and the RAM size of the ATmega328P?

The standard handling technique is to store the hash of the password instead of the plain text. Because the hardware in these cases does not have enough space to store the hashes. It needs to save the space for other use.

Question 12: A practical man-in-the-middle hardware attack (6 points)

Consider the lens project by Zach Banks and Eric Van Albert, entertainingly presented in a presentation at Def Con 23. Watch the talk, then answer the questions below.

a. What is the importance of the punch-down method of connection as opposed to simply cutting and plugging in the conductors? How does this help the attacker?
   Simply cutting and plugging in will bring extra resistance and make the signal worse and cause data loss.
b. What is the accelerometer for?
   Detect if the board is jostled or otherwise disturbed while in operation.
c. What is the difference between “passive tap” and “active tap”? What does an active tap allow an attacker to do that would otherwise not be possible?
   Passive tap is just like the original cable, just connect the data. But the active tap will allow them to connect DUTA to tap A and DUTB to tap B. And take control of the ethernet cable. This will allow the attacker to tamper with the data flow.
d. To achieve the goal of looping camera footage, why can’t they just record and replay the raw packets seen on the network?
   There is sequence numbers in the data stream that will make this replay impossible.
e. Briefly summarize the network layers and protocol involved in the final video looping demo.
   For TCP/IP model, the layers are application layer, transport layer, network layer, and also link layer. The protocols are TCP, UDP, RTP, HTTP, IPV4, ARP, ICMP, SSH.
f. They mention that they’re “glossing over” the issue of HTTPS. How would HTTPS address this problem?

Because HTTPS has SSL encryption and it is much harder to decrypt and analyze the data to perform the man-in-the-middle attack. But as on an embedded camera it is hard to deploy good SSL and updates, they just gloss over it.

Question 13: Malware Analysis (18 points)

This question will walk you through some rudimentary analysis of *REAL* malware. This is dynamic analysis (observing the malware's behavior in a VM); a deeper analysis would incorporate static code analysis (looking at the machine code).

NOTICE! Do *not* run this malware on your **Windows VM!** Follow the instructions to set up a scratch VM on your local system or a cloud service.

Further, if running the malware from a VM on a machine on the Duke network, please email the **Duke IT Security Office** first with the message below. Do this each separate day you are about to run the malware. The network requests made by the malware trigger alerts in systems monitored by the IT Security Office (ITSO), and they've requested this advance notice to avoid triggering needless intrusion response, especially after-hours.

This alert need not be sent if your throw-away Windows VM is hosted on a cloud service, because Duke IT won't see those malware request packets.

> To: security@duke.edu
> Subject: Running Emotet malware for ECE 560
>
> ITSO team: I am in Prof. Bletsch's Computer and Information Security class (ECE 560) and am studying malware analysis. I will be running an instance of Emotet malware on a temporary VM within the next few hours, and want to give you advance notice. This procedure has been approved by Duke ITSO, and the VM will be reverted to pre-infection state within 30 minutes. Please contact Tyler Bletsch (Tyler.Bletsch@duke.edu) and Anthony Miracle (Anthony.Miracle@duke.edu) if there are any issues. Thank you.

Note: This question includes a lot of steps, not all of which require a response from you. Steps that are asking for a response have whitespace after them, and the prompt is highlighted green.

Side note: The Windows Registry
If you are not familiar with what the Windows Registry is, research it before proceeding.

Part 1: Setup

1. We need a Windows VM, but because we're going to be intentionally executing malware on it, it cannot be hosted on Duke VCM. You have two choices:
    a. **CHOICE 1:** Install a hypervisor onto your own computer.
        i. For this, I recommend VirtualBox: it's free, supports snapshots, and works cross platform. If you have your own preferred hypervisor, you can use that, assuming it has support for VM snapshots.
        ii. You can get the install ISO for Windows 10 here. If you're on Windows and don't want to download Microsoft's "download tool" (a laudable goal), here are directions to get a direct ISO download link.

iii. Just make a VM, boot the ISO, and next-next-next your way through the install, noting these facts:
    1. No need to provide an activation key, as our experiment will be over long before the activation grace period is up.
    2. If prompted, indicate that the version of Windows 10 you're installing is 64-bit Professional.
    3. Any time Microsoft gives you a privacy question, answer "no".
    4. When it comes to making a user, do NOT give it a Microsoft account! You don't want this machine to have any access to a persistent account. For dumb+bad reasons, Microsoft has increasingly hidden the ability to use a non-cloud account, but it's still there. Indicate that you want Windows to be part of a "domain", then it will let you make a local account. You may need to disable internet access during install to force the installer to let you make a local account.

        Note: After booting, Microsoft claims in the security panel that having a cloud account is "more secure"; if you see this, be sure to laugh out loud directly into the computer's microphone, because that is very wrong.
    5. Do not use a password you use anywhere else.

iv. You should now be able to finish the install and boot into a fresh Windows VM.

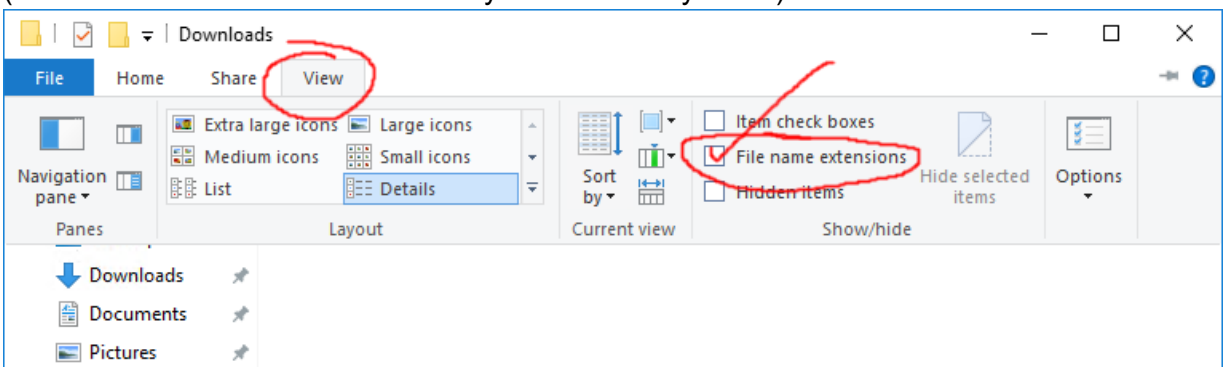
v. In your hypervisor of choice, enable clipboard sharing and drag-and-drop. This will make it easy to inject files into the VM and copy useful content out for analysis.

b. **CHOICE 2:** If you don't want a local VM, you can use a cloud service. Amazon EC2, Vultr, Digital Ocean, or Microsoft Azure should all be able to host a Windows VM easily. For Amazon, you have educational credits you can use to make the exercise free. The downside of this route is that cloud snapshot facilities are generally slower and more cumbersome than on a local hypervisor. For this, you'll be connecting to the VM via Remote Desktop.

2. Once the VM is running (either local or cloud), we'll permanently disable the built-in Windows malware protection "Windows Defender". Follow the "group policy" process described here.

    Side comment: You might think that malware isn't that dangerous if Defender can simply catch it. That's not because this malware isn't dangerous, but simply because it's old. Threats you face in the wild will often not be flagged by anti-malware software. We disable this protection here because we're learning on a piece of known malware.

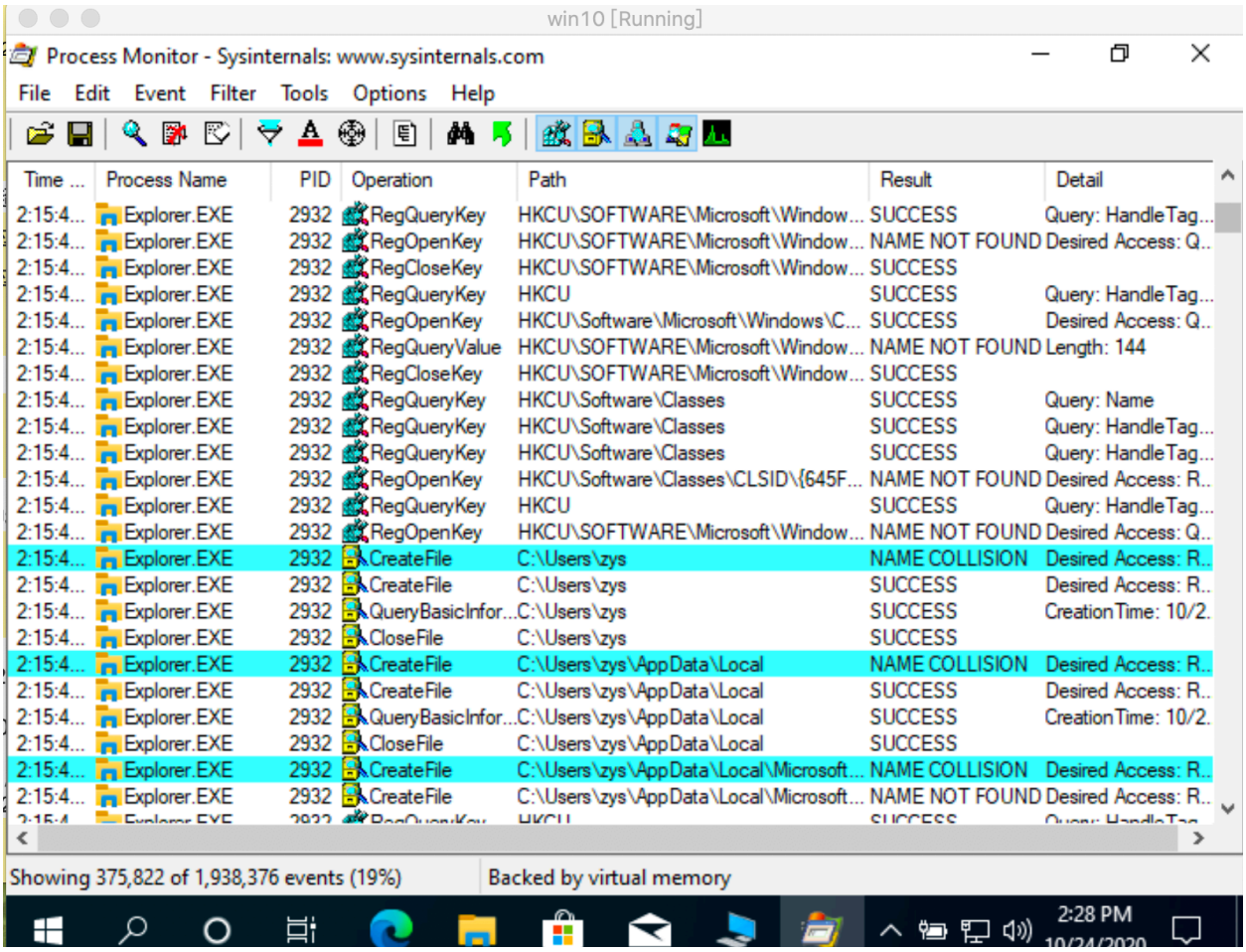
3. As described in Homework 1, be sure to enable file extensions in Explorer for this VM (and on all other Windows machines you touch until you die):



4. Before you proceed, do a bit of Windows-specific research. What does the "CurrentVersion\Run" family of registry keys do?
   Run registry keys cause programs to run each time that a user logs on. The data value for a key is a command line no longer than 260 characters. Register programs to run by adding entries of the form description-string=commandline. We can write multiple entries under a key. If more than one program is registered under any particular key, the order in which those programs run is indeterminate.

Part 2: Process Monitor

1. Download and unzip Process Monitor from here. Process Monitor is like WireShark, except that instead of network traffic, it captures the traffic between user processes and the operating system, breaking these down into “File IO”, “Registry IO”, “Network IO”, and “Process and Thread Activity”.
2. Play around a bit with it.
3. Just like WireShark, Process Monitor captures events, displaying only those events that match the current filter. In preparation for the malware test we’ll be doing, let’s reduce how “noisy” the output is -- right click the process name for events that seems superfluous, such as “svchost.exe”, “services.exe”, “SearchIndexer.exe”, etc., and choose “Exclude <thing>”. Do NOT exclude “explorer.exe”. This implicitly updates the filters.
4. Process Monitor also has a highlighting filter -- events matching this filter are marked in cyan. Let’s highlight events that change something as opposed to simply reading -- add a highlight filter for “Category” set to “Write”. This filter is a high-level catch-all for all operations that “do something” rather than just passively read something (e.g. file writes, registry changes, etc.).
5. Paste a screenshot below of Process Monitor showing some events with a few “write” events highlighted.

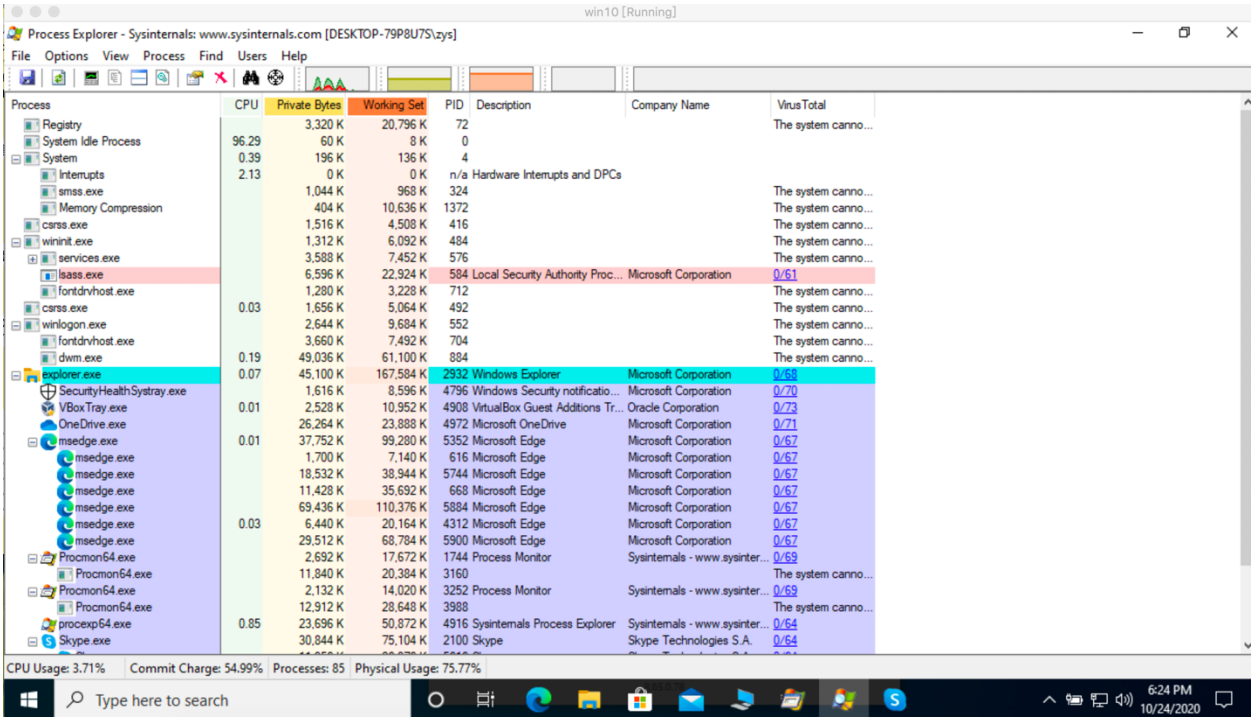


6. Stop the capture, clear the buffer, and leave Process Monitor running in preparation for later steps.

Part 3: Process Explorer

1. Download and unzip Process Explorer from here. Process Explorer is like a heavyweight version of the built-in Task Manager, showing all running programs, but it can do much more.
2. Play around a bit with it.
3. Try running the Skype that's included with Windows 10 (but do not login!). Find this program in Process Explorer. Right click the process, and view properties. Check the TCP/IP tab to see connections being made in realtime. At this time, turn off "resolve addresses".
4. One of the special abilities of Process Explorer is the ability to submit any running program to VirusTotal.com, a website that scans any file given to it with virtually every virus scanner on the market today. Right click the calculator process and choose "Check VirusTotal.com". The VirusTotal column will populate with a number, hopefully zero out of something (e.g., "0/67").

5. You can even check ALL running processes - do so from the menu: “Options” -> “VirusTotal.com” -> “Check VirusTotal.com”. Wait for all the VirusTotal entries to populate. Did you get any results showing non-zero hits? If so, click the number to see details. If not, click on the Skype VirusTotal link to see details.
6. Now kill the calculator process using Process Explorer.
7. Paste a screenshot of Process Explorer showing all the running processes with their VirusTotal results.



Part 4: Wireshark

1. You know about Wireshark. Install it in your VM.

Part 5: The Malware

ULTRA-DANGER! Take EXTREME care in handling of the Windows malware linked below. Do not even *extract* it anywhere but the sandbox of the throwaway VM unless you seriously know what you’re doing!

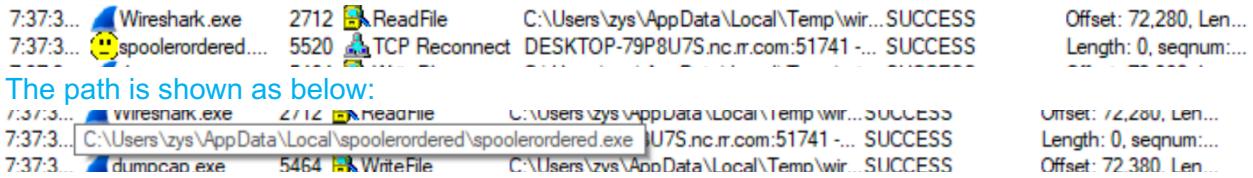
This is real malware recently analyzed by security researchers. It was the payload of a few spam email campaigns that included malicious javascript, macro-enabled Word document, etc. We’re skipping the attack vector and analyzing the payload directly.

1. Download the malware to the VM: https://people.duke.edu/~tkb13/courses/ece560/homework/hw4/MALWARE.zip

2. Extract the malware. The ZIP password is:
   `infected`
3. At this time, take a snapshot of your VM (so if something goes wrong, you can restart at this point). You may need to shut down your VM to do so, depending on your hypervisor/cloud.
4. Ensure that Process Monitor, Process Explorer, and Wireshark are running.
5. In Wireshark, begin monitoring on the LAN connection. Leave it running for a few minutes to get a baseline for traffic on the system (as even idle Windows machines are notoriously chatty on the network). When you're satisfied, restart the capture to clear it.
6. In Process Monitor, clear the log and enable event capture.

This is the point of no return for the **first infection test**. Once you pass this point, you must complete this test fairly quickly or revert to the snapshot you took and start again. To help with this, read all the steps below before you start them. It's okay if you miss something or make a mistake, but you must revert the VM to the snapshot before you try again. The malware was fairly quiet in my analysis, but given enough time, it might wake up and start attacking other machines or taking unknown action at the behest of criminals. DO NOT LEAVE AN INFECTED VM LYING AROUND FOR MORE THAN 30 MINUTES.

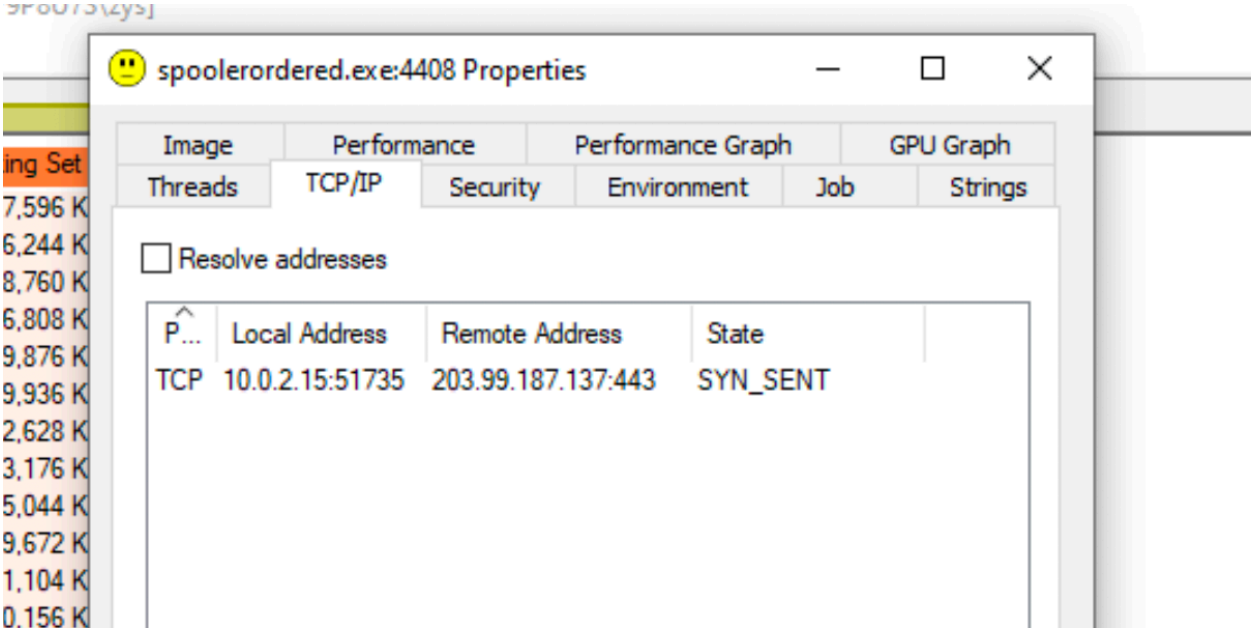
7. The malware is provided with no file extension to further reduce the risk of accidental execution. Rename the malware to `malware.exe`. The icon should change to an ambivalent face:
   
8. Execute the malware.
9. Find the malware's activity in Process Monitor. It does a LOT of registry reads, so you may want to filter some of that out.
10. Using Process Monitor, find out where the malware hides itself on the system.
    Hint: The malware uses a rename operation that shows up as SetRenameInformationFile in Process Monitor. What is the new path to the malware? Show a screenshot indicating this.



11. Now that we know the new filename of the malware, scroll further through Process Monitor to see activities under the new name.
12. After several seconds, the malware will *edit* (not just read) several registry settings. One of the registry entries contains "CurrentVersion\Run" (which you researched earlier

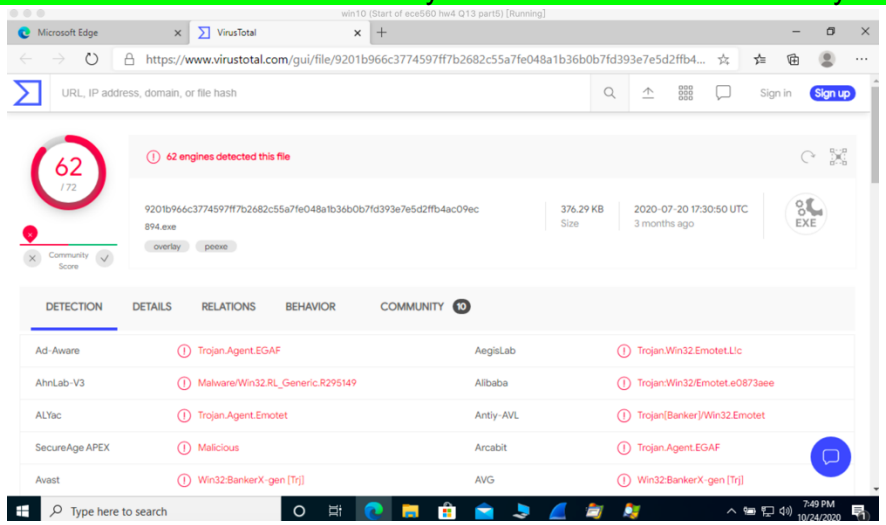
in this problem). Based on this and your findings in step 10, how does this malware hide itself and remain persistent?
It changes its path to a new one and renamed itself. The AppData directory along the path cannot be seen in its father directory.

13. Using Process Explorer, identify the running malware resident in memory. Right click and open the properties, and view the TCP/IP tab. Ensure that “resolve addresses” is disabled, and monitor the tab for a few minutes. You should spot an outgoing connection. If you don’t, you probably missed the event -- revert to snapshot and try again, skipping the steps you’ve already done so you can catch the outgoing connection.
14. What is the IP address it is connecting to? Get a screenshot of it in Process Explorer.



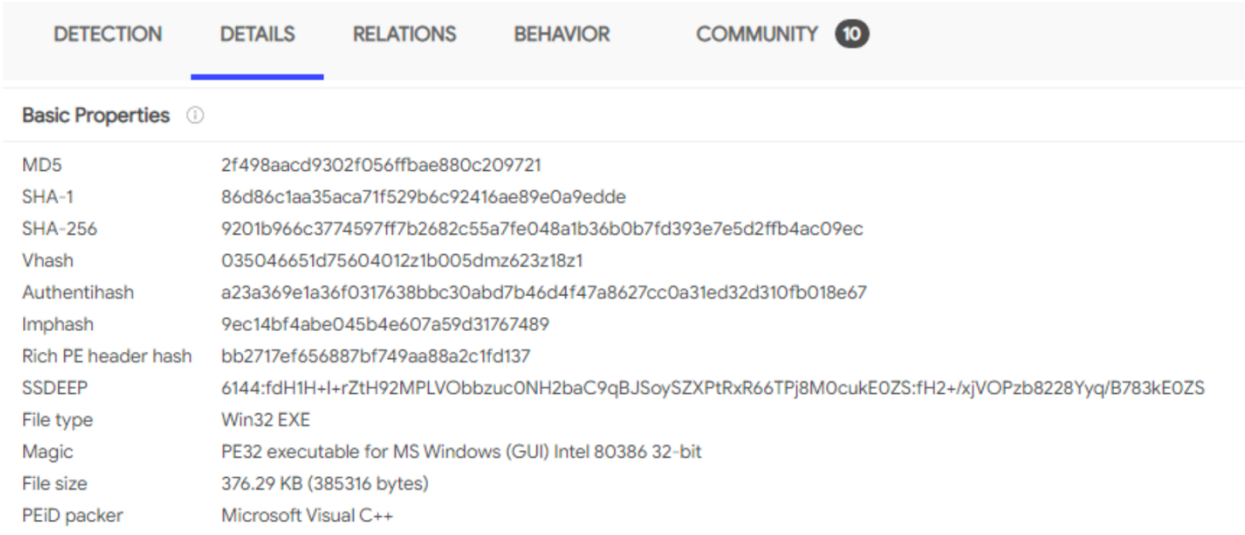
15. Use the VirusTotal option from Process Explorer to evaluate the malware. Click on the resulting number to see the detailed analysis. Paste a screenshot of the result. How

many scanners detected it? How many did not? How does this make you feel?

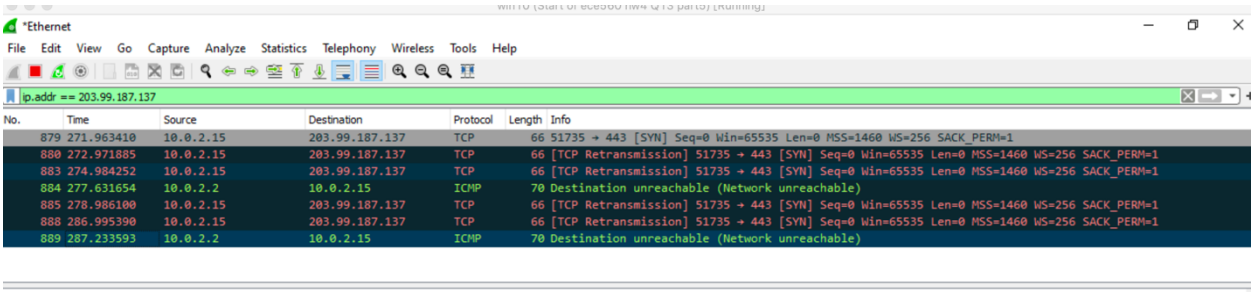


62 scanners detected it. While 10 did not. I feel that is really import to use a professional defender on the personal computer to make sure you are safe. And at the same time, some engines may not be able to find the malware.

16. On VirusTotal.com, under "additional information", several long hex strings are listed -- what are these?



17. Now that we know the IP it connects to, set the Wireshark display filter to only show this IP address.
18. You'll find that most of the traffic is encrypted with HTTPS, but strangely, two requests are made on the HTTPS port (443), but the content is unencrypted HTTP. What are the URLs of these requests? Describe the content you see transmitted/received.



The behavior is different from what is described. There are two other ICMP connections among the HTTPS port 443 requests. But there is no requests are made on the HTTPS port (443), but the content is unencrypted HTTP.

19. Reboot the VM, and run Process Explorer. Confirm that the malware is again running.
20. Revert the VM to its pre-infection snapshot. Boot it and confirm the malware is not running. This concludes the **first infection test**.

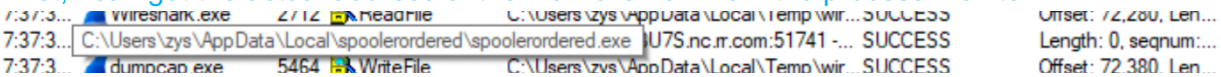
Part 6: Developing a cure

21. Using the information you've gathered, what steps should you take to remove an instance of this malware from a real infected system (i.e., one where you don't have a known-good VM snapshot)?
    I should use steps like part 5 step 10 use a Process Monitor to find the real place of the malware and try to remove it. And I can also use process explorer to kill the malicious process. After that I can delete malware and clear the recycle bin to remove it.

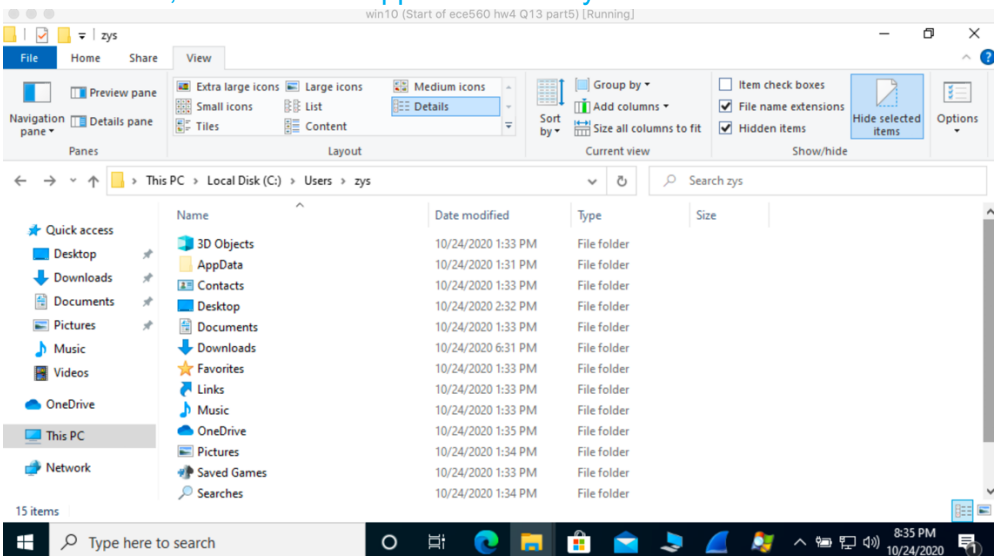
This is the point of no return for the **second infection test**. Again, once you pass this point, you must either succeed in clearing the infection or or revert to the snapshot you took and start again.

22. Run the malware on the clean VM again.
23. Perform your proposed procedure to remove the malware and reboot the VM. Using Process Explorer, verify the malware is gone. Did you get it? If it is not, try again until you successfully remove it. Note your actions and results as you go.

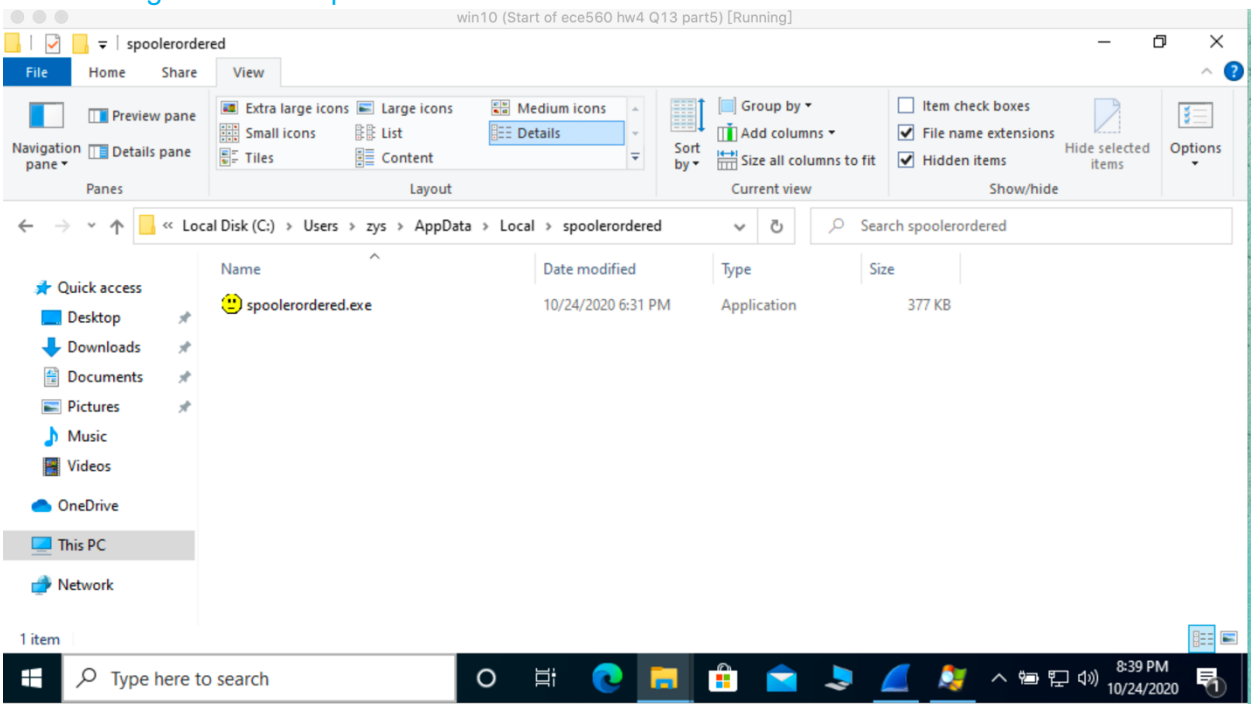
    First, I can get the actual address of the malware now from the process monitor.



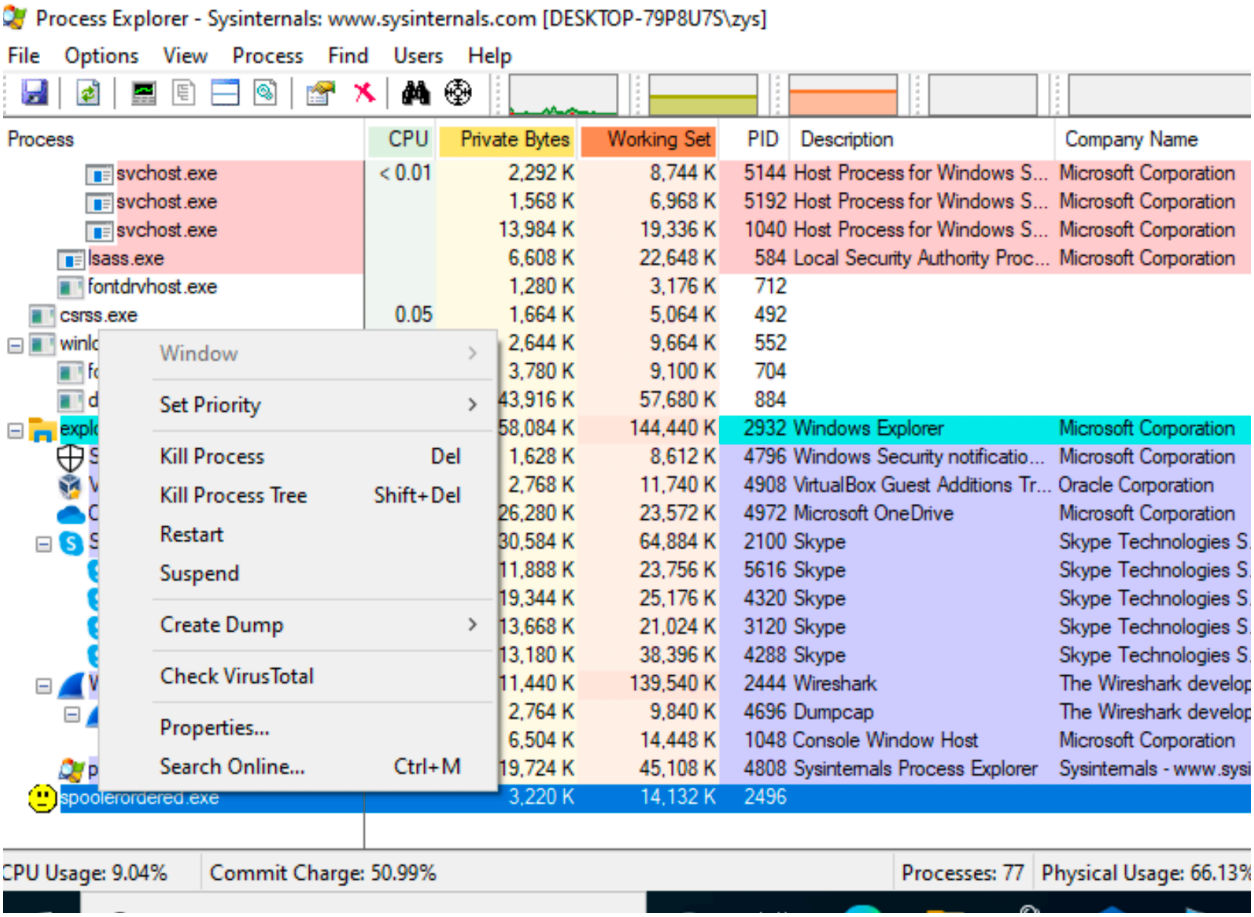
Then go to the path and check the box to show hidden items. Here after showing the hidden items, I can see the AppData directory.



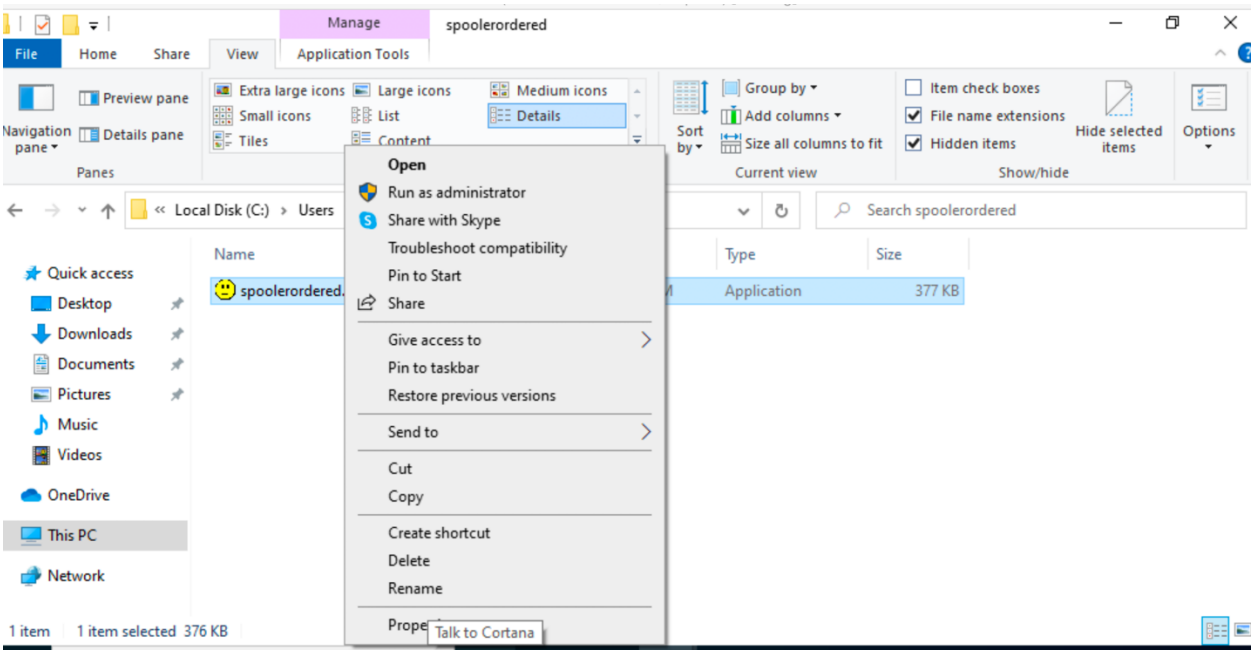
And here I get the actual position of the file now.

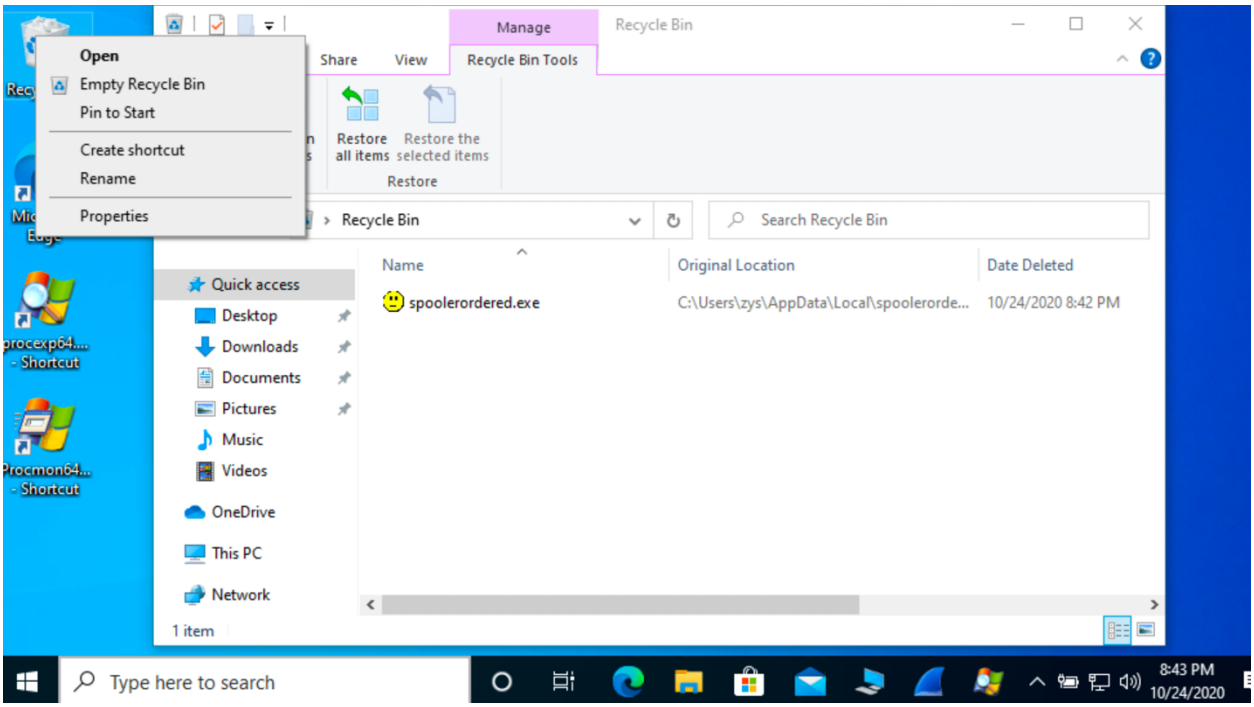


After that I use the process explorer to kill the malware process.



Then I just delete it and clean the recycle bin.





24. Once you are satisfied, revert to the snapshot again. This concludes the **second infection test**.

25. Now that the malware is apparently gone, how confident are you that you got it all? If you answered something like "100% confident", research the impact that hubris has had on human history (Icarus, Napoleon in Russia, the roaring '20s and great depression, the Vietnam War, the 2008 financial crisis, etc.), and revise your answer. How might some exceedingly clever malware have survived?
    I am 90% confident. They might make multi copies and store it inside places that we cannot find. As in the above solutions we only deleted the malware that is current executing, there might be other copies that are still remaining in the system.

26. At any point, did our analysis uncover what this malware was designed to do? Does that worry you?
    The malware might be designed to establish connection between the remote machine and the infected one and might take control of the infected PC. It is really scary as the private information might be leaked and at the same time the attacker might do something to lock your pc and ask you for money.

27. When you're done, shut down the VM.
28. To ensure the VM isn't accidentally used for something else later, after the homework deadline has passed, destroy the VM.

Background: The malware in question is a variant of Emotet, documented here. This specific variant is from here. It presumably is allowing remote control of infected machines for basically any purpose. This means that infected systems could easily be used as a botnet to conduct coordinated brute force login or DDOS attacks against internet sites, to retrieve any confidential files, to log user keystrokes, or to display any manner of advertising or malicious content to the users of the system.

~