THE 2019 SANS HOLIDAY HACK CHALLENGE

"Honesty is ~~not~~ always the best policy"

Janusz Jasinski


ADMIT ONE
This ticket entitles its bearer to
admittance for one to
KringleCon 2: Turtle Doves
Location:
Elf University
17 Christmas Tree Lane
North Pole

Introduction

Contents

Contents

Terminals (Docker)

All hosted on the sub-domain https://docker2019.kringlecon.com/

Ed Escape

- **Hint**: http://cs.wellesley.edu/~cs249/Resources/ed_is_the_standard_text_editor.html
- **Location**: Train Station
- **URL**: https://docker2019.kringlecon.com/?challenge=edescape

Screenshot

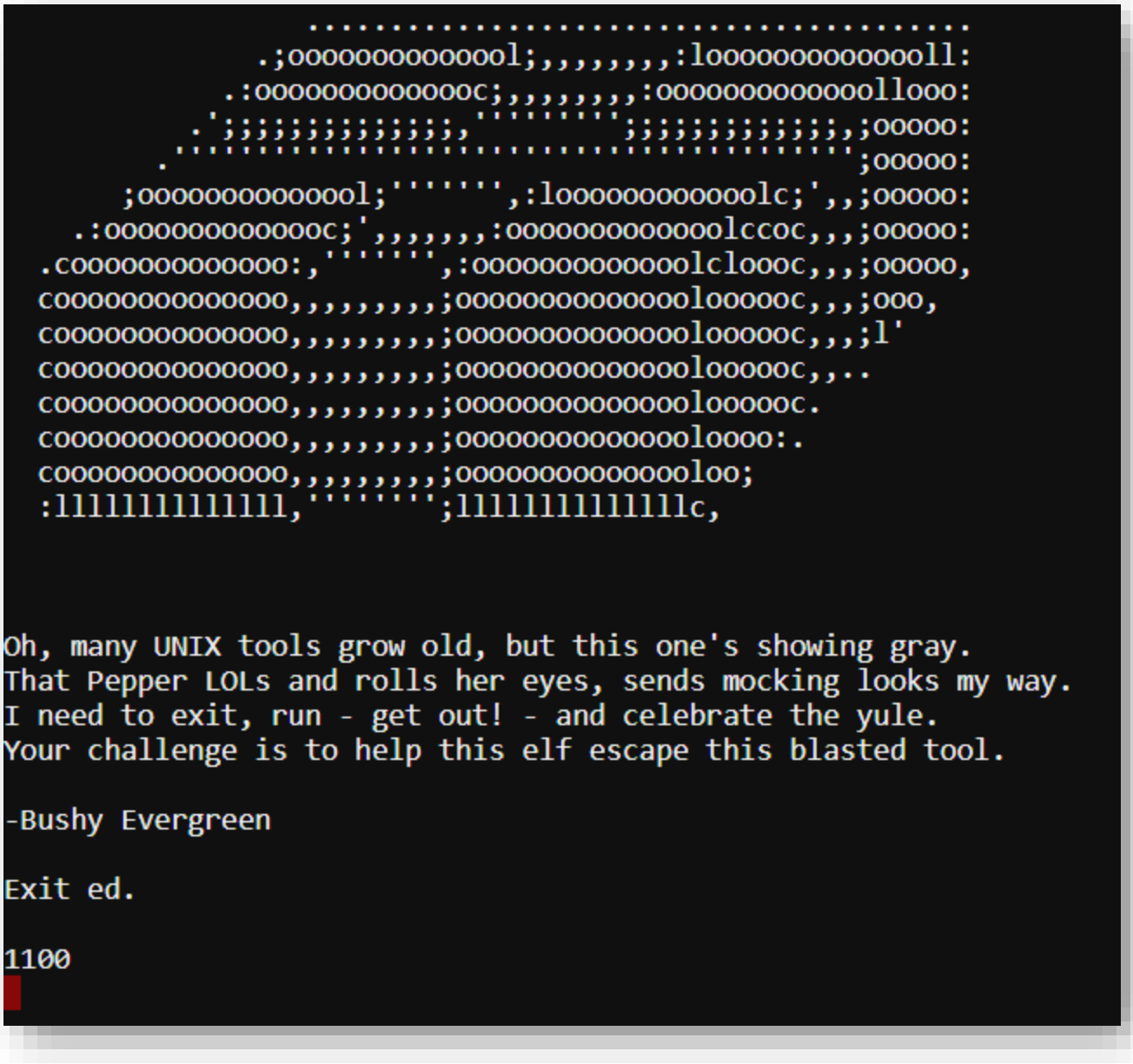


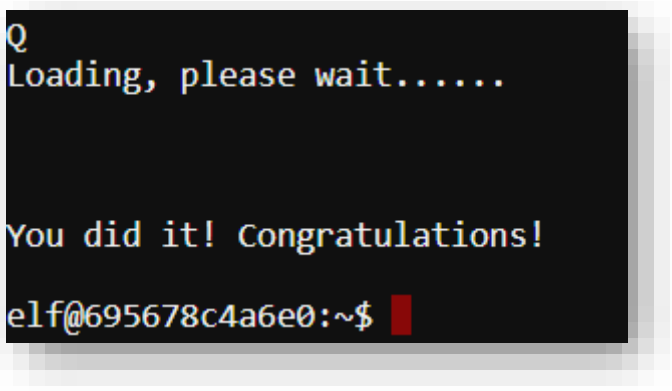
Figure 1 - Escape Ed Docker Screenshot

Synopsis

It looks like we need to escape the editor. Not familiar with this particular editor so will have to rely on the hint.

Solution

Reading the site[1], it seems that "*ed is a powerful text editor based on non-visual (line-oriented) behavior*". Doing a quick Google search gave us the man page for the editor[2] which told us that in order to quit the editor, we just need to enter q or Q.



[1] http://cs.wellesley.edu/~cs249/Resources/ed_is_the_standard_text_editor.html

[2] https://linux.die.net/man/1/ed

Path

- **Hint**: Green words matter, files must be found, and the terminal's $PATH matters.
- **Location**: Hermey Hall
- **URL**: https://docker2019.kringlecon.com/?challenge=path

Screenshot

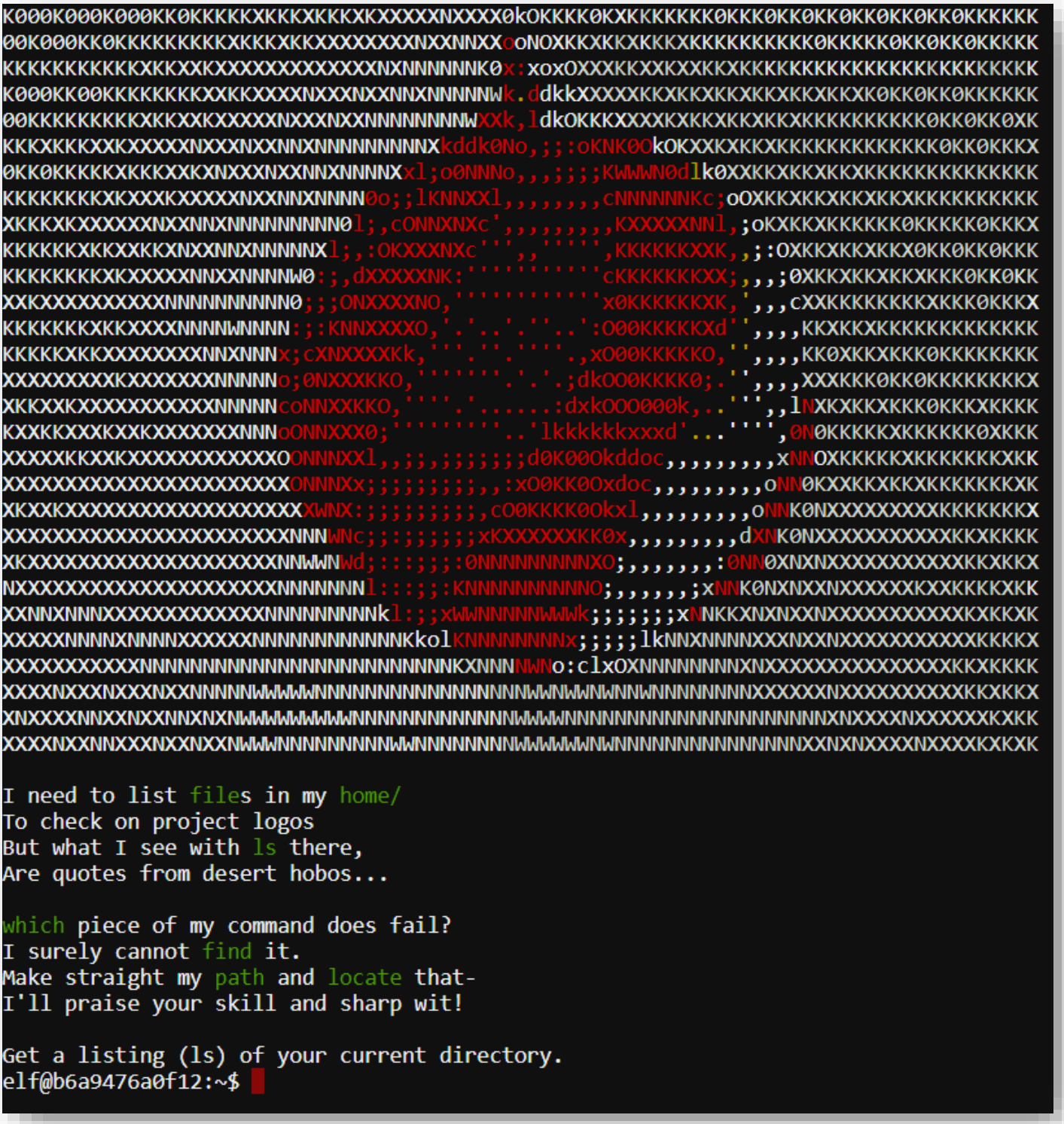


Figure 2 - Path Docker Screenshot

Synopsis

We need to list the current directory. I guess just doing a `ls` is not going to happen. The hint gives a lot away of commands we can use so let's go through them all.

Solution

- `which` command[3]
- `find` command[4]
- `$PATH` command[5]
- `locate` command[6]

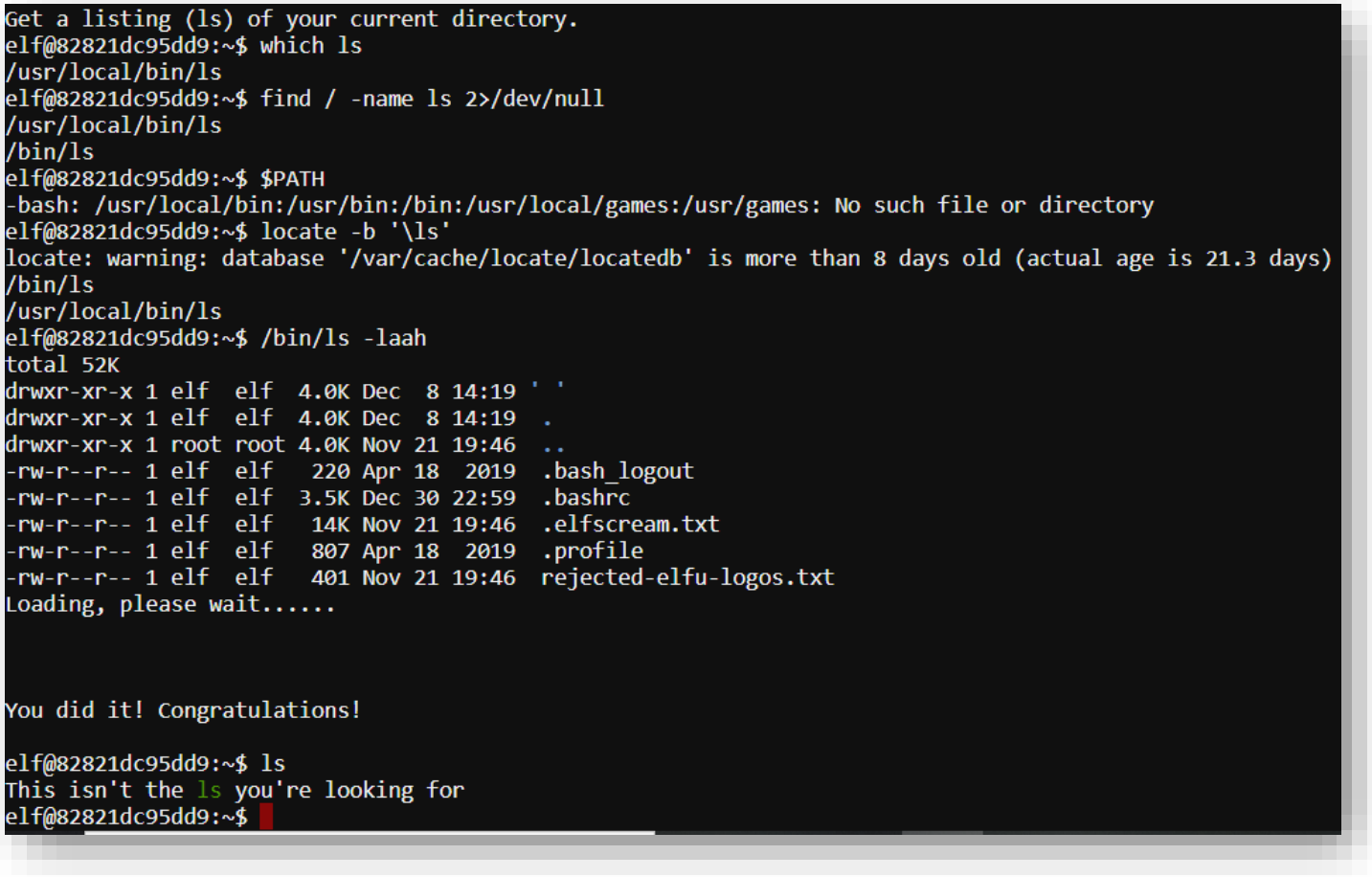


Figure 3 - Path Docker Screenshot

Miscellaneous

There are some files that seem out of place so let's see what they contain

---

[3] https://linux.die.net/man/1/which

[4] https://linux.die.net/man/1/find

[5] http://www.linfo.org/path_env_var.html

[6] https://linux.die.net/man/1/locate

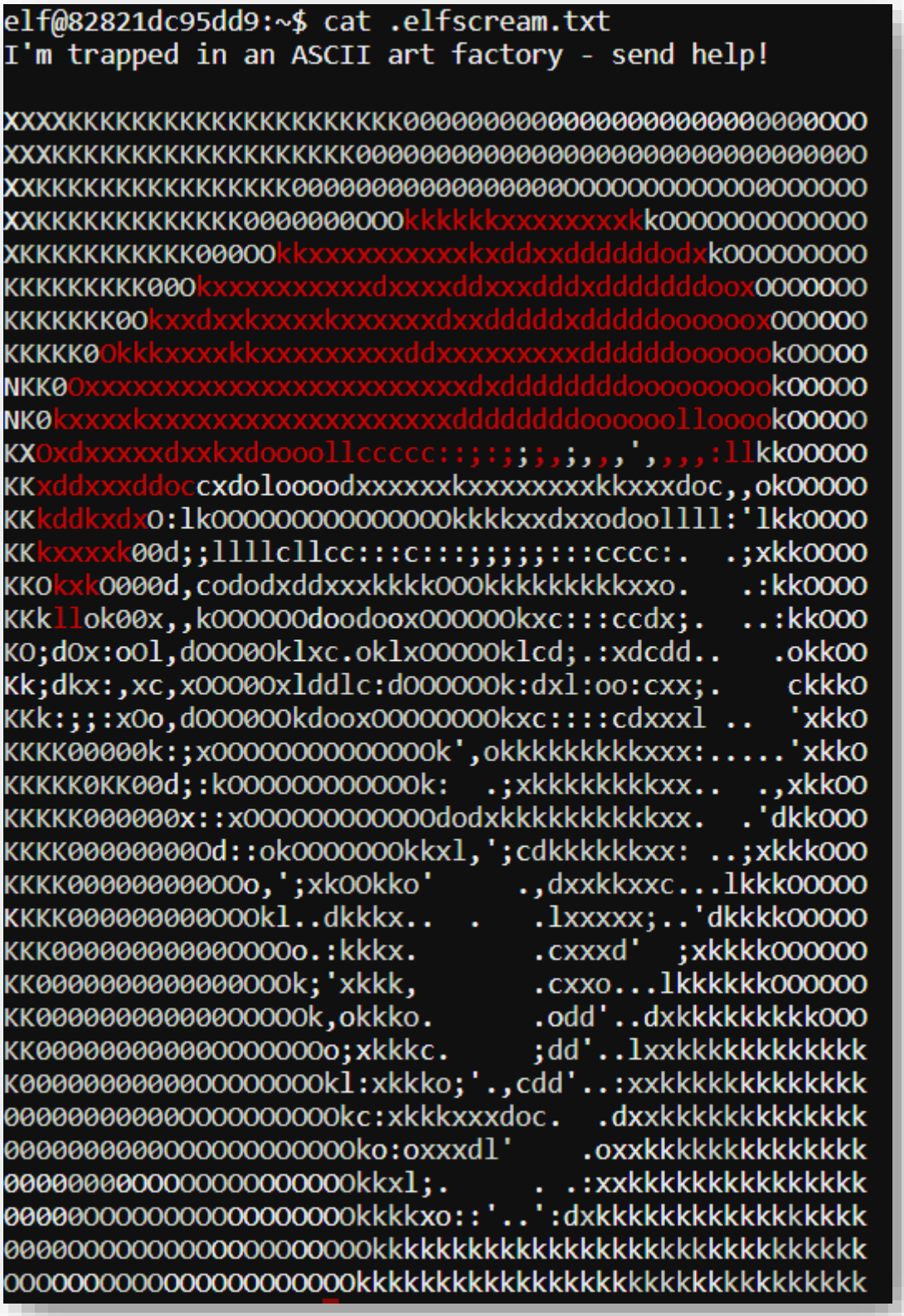


Figure 4 – Elfscream

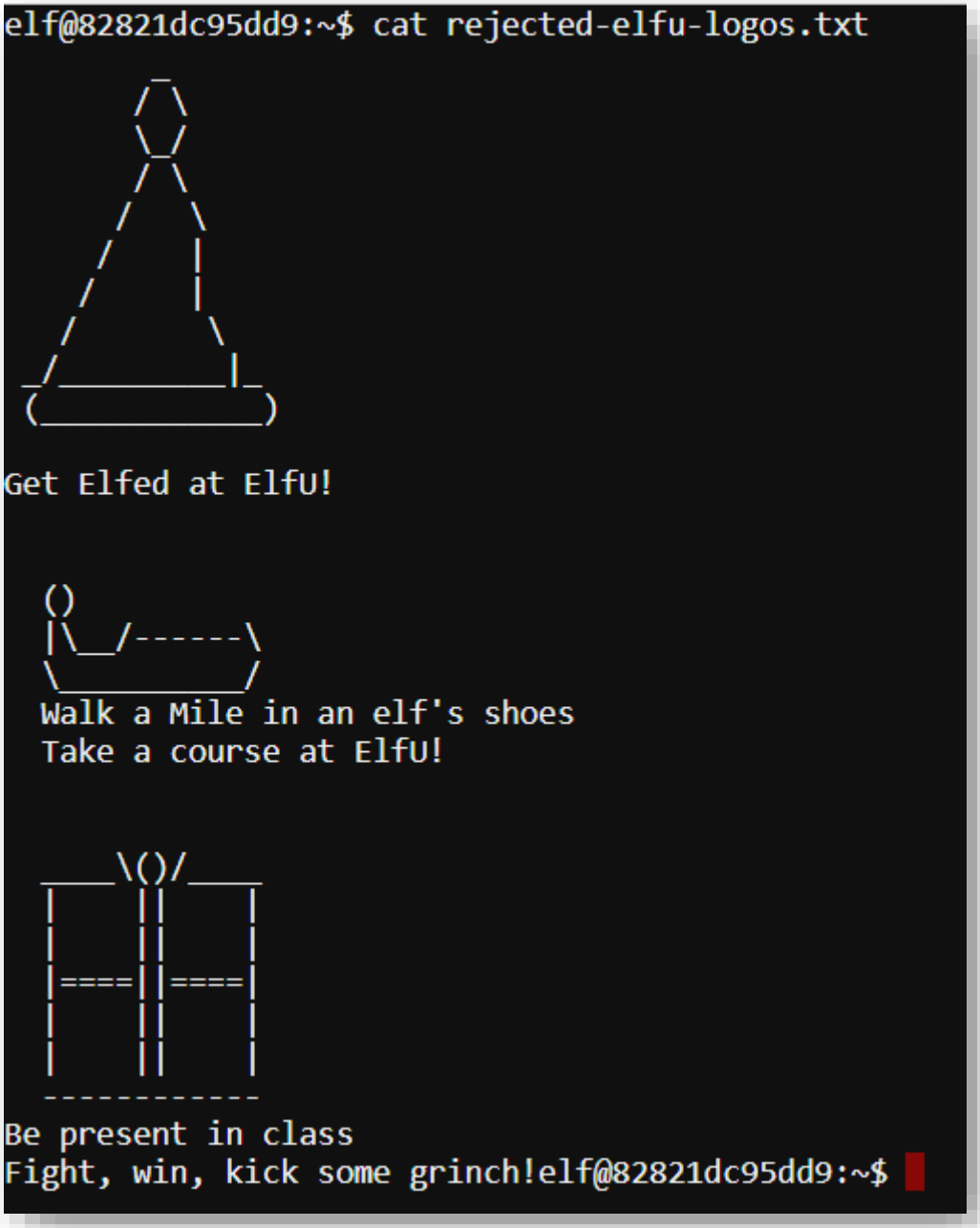


Figure 5 - Rejected Elf Logos

Mongo

- **Hint:** https://docs.mongodb.com/manual/reference/command/listDatabases/
- **Location:** NetWars
- **URL:** https://docker2019.kringlecon.com/?challenge=mongo

Screenshot

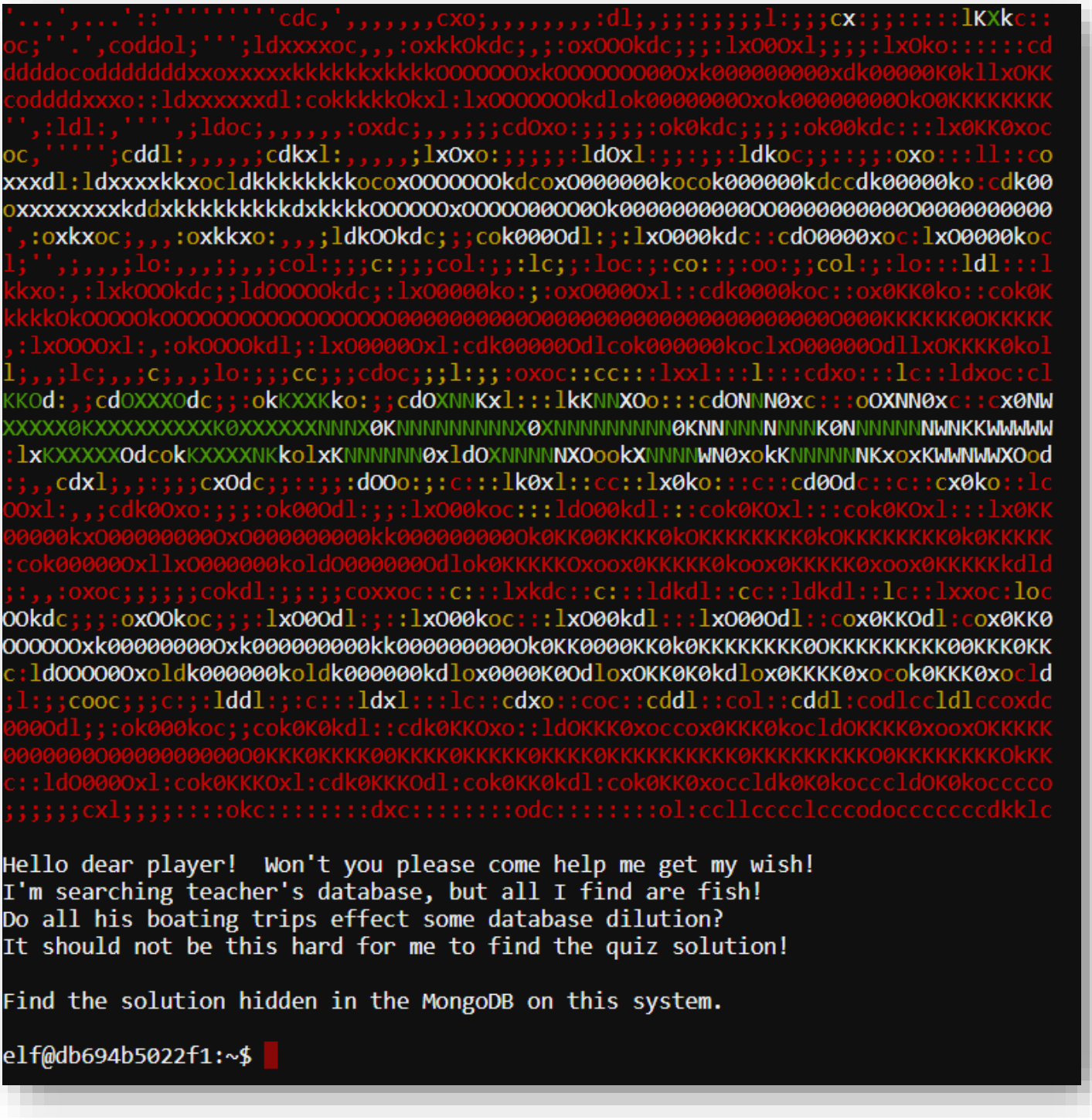


Figure 6 - Mongo Docker Sccreenshot

Synopsis

Seems pretty straightforward (famous last words) where we need to go into the MongoDB instance and retrieve some information

Solution

We try and connect to the instance[7] but are told that it's not running on that port, along with a fairly useful hint that it's on another port.

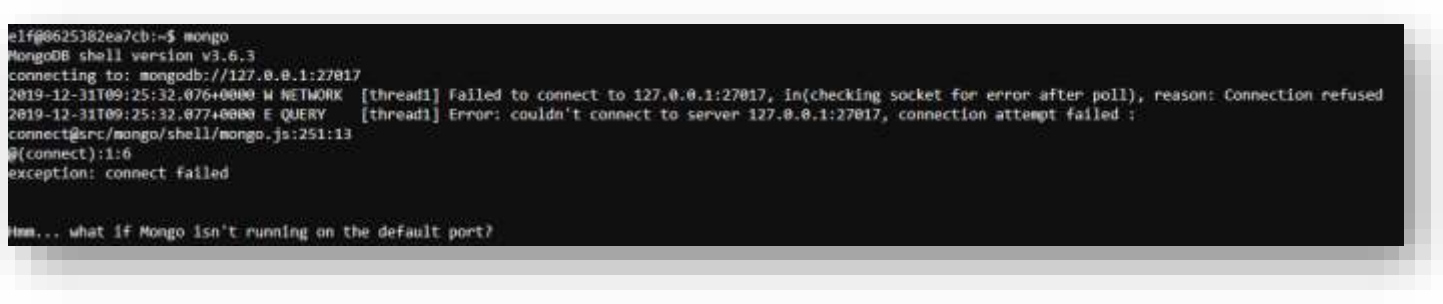


Figure 7- MongoDB Docker Screenshot

There are a few ways to check which port it's actually running on as shown below. The second of which shows us that we have `sudo` access on python running a script in the root directory but we don't dig any further.

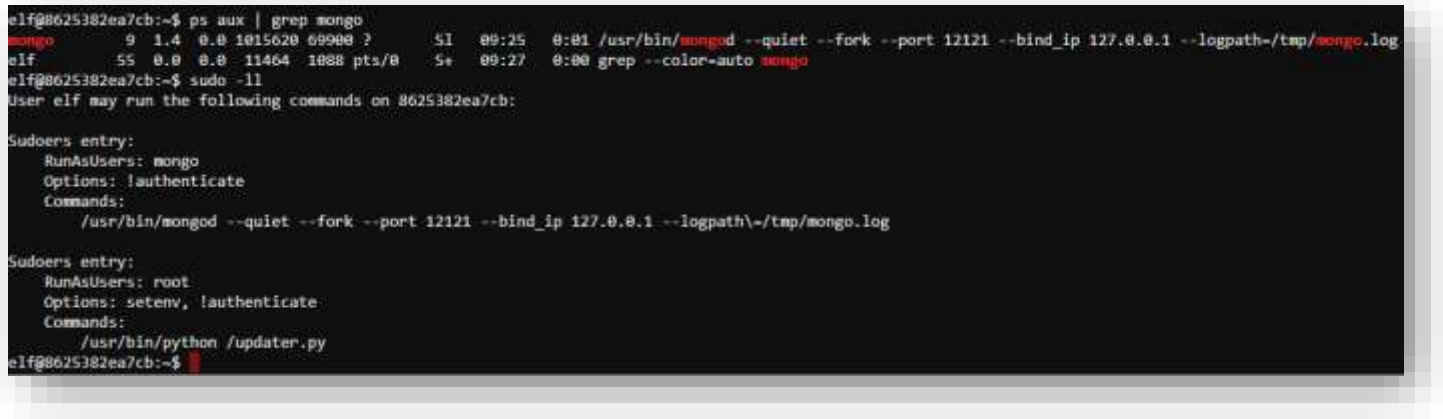


Figure 8 - MongoDB Docker Screenshot

Using the mongo manual, we now connect to the instance on the port mentioned to open up a mongo shell

[7] https://docs.mongodb.com/manual/reference/program/mongo/#syntax

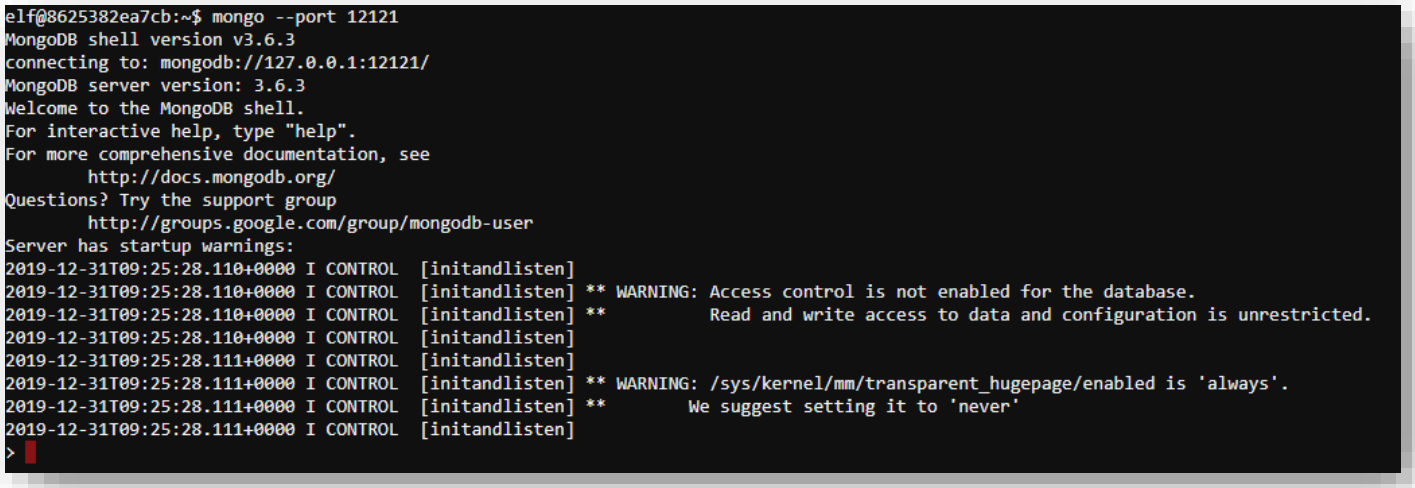


Figure 9 - MongoDB Docker Screenshot

We get a few warnings. *Enabling access control on a MongoDB deployment enforces authentication, requiring users to identify themselves. When accessing a MongoDB deployment that has access control enabled, users can only perform actions as determined by their roles.*[8]

Carrying on, we first list the databases on the instance.

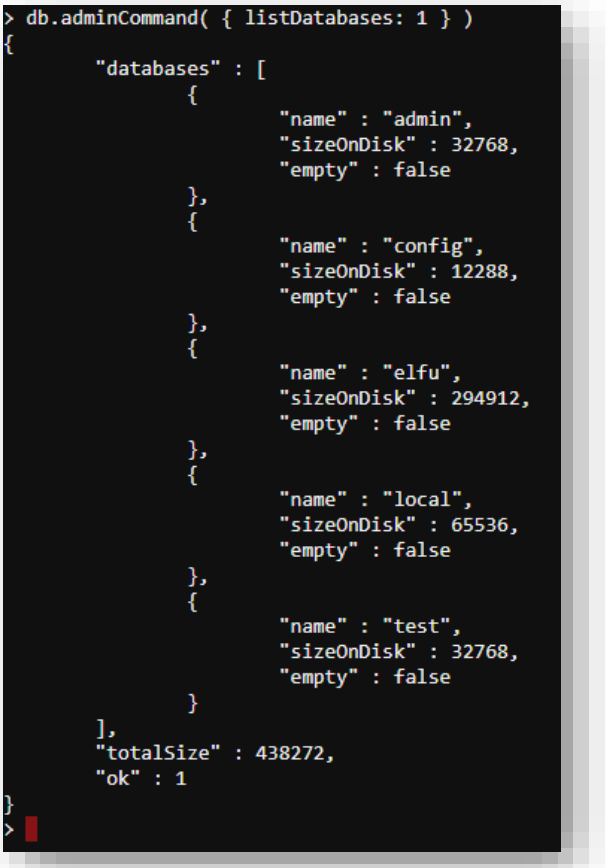


Figure 10 - MongoDB Docker Screenshot

[8] https://docs.mongodb.com/manual/tutorial/enable-authentication/

The elfu database sounds like what we are after so we carry on down that route.

```
> 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();displaySolution(); **" }
> db.loadServerScripts();displaySolution();
```

Figure 11 - MongoDB Docker Screenshot

We show collections and see one called `solution` that we go into which tells us what we need to do. The script is run which completed the terminal.

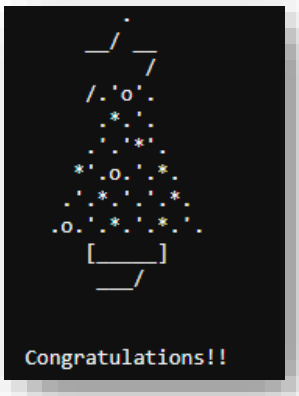


Figure 12 - MongoDB Docker Screenshot

To see what was in the other collections, we ran the following script against each database:

```
var collections = db.getCollectionNames();
for(var i = 0; i< collections.length; i++){
   print('Collection: ' + collections[i]); // print the name of each collection
   db.getCollection(collections[i]).find().forEach(printjson); //and then print the json of
each of its elements
}
```

Unfortunately, it didn't seem to reveal any easter eggs or further clues. Admin, config and local gave seemingly standard output whereas test gave us the below.

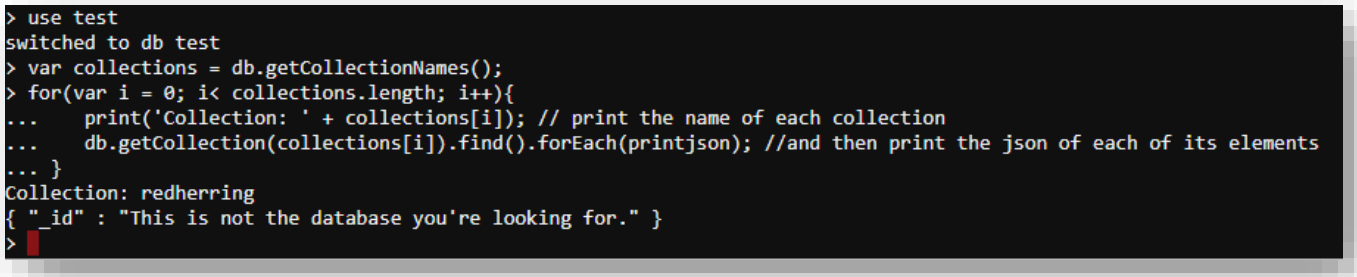


Figure 13 - MongoDB Docker Screenshot

Nyanshell

- **Hint**: On Linux, a user's shell is determined by the contents of /etc/passwd
- **Location**: Speaker UNpreparedness Room
- **URL**: https://docker2019.kringlecon.com/?challenge=nyanshell

Screenshot

Figure 14 - Nyanshell Docker Screenshot

Synopsis

The screenshot above says it all. This combined with the hint suggests that the user won't land in a bash prompt and our task is to make it happen.

Solution

So let's see what happens when we log in normally.

Figure 15 - Nyanshell Docker Screenshot

Right, so that's what we need to avoid ☺

As the hint suggests, let's look at /etc/passwd. We do this by first refreshing the terminal to land in bash.

```
elf@d7df796e80ef:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
elf:x:1000:1000::/home/elf:/bin/bash
alabaster_snowball:x:1001:1001::/home/alabaster_snowball:/bin/nsh
elf@d7df796e80ef:~$
```

Figure 16 - Nyanshell Docker Screenshot

Alabaster is running the shell `/bin/nsh` so we look into it a bit more.

```
elf@d7df796e80ef:~$ ls -laah /bin/nsh
-rwxrwxrwx 1 root root 74K Dec 11 17:40 /bin/nsh
elf@d7df796e80ef:~$
```

Figure 17 - Nyanshell Docker Screenshot

A useful aid to remind yourself of what the permissions are is shown below

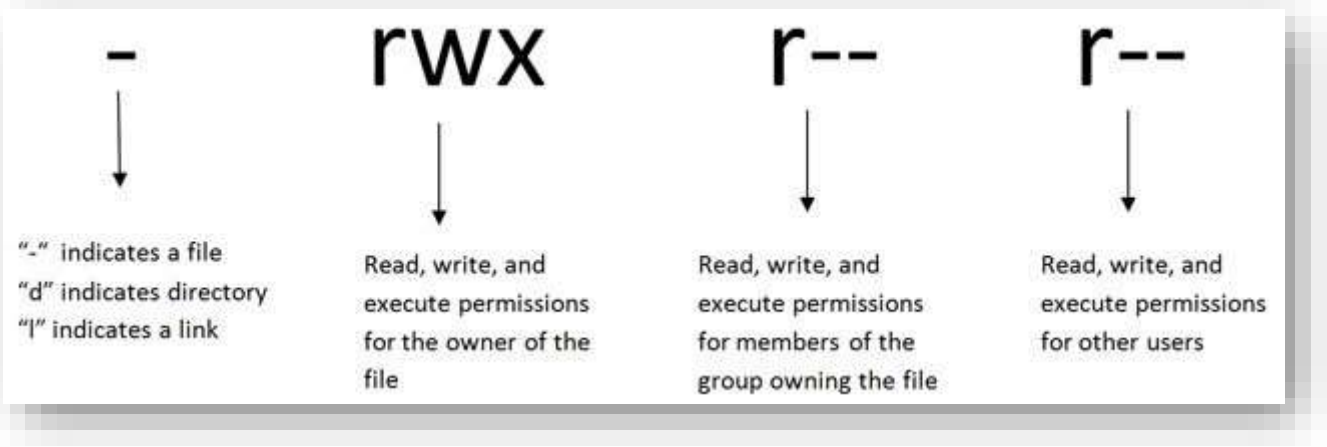


Figure 18 - Linux File Permissions

Using this, we can see that the file should be writable. So we'll try to copy the contents of the bash file we want into this one by first emptying it and then copying.

```
-bash: /bin/nsh: Operation not permitted
elf@d7df796e80ef:~$
```

Figure 19 - Nyanshell Docker Screenshot

However, we get an error. We then use `lsattr`[9] to list file attributes. The `i` means the file is immutable.

```
elf@d7df796e80ef:~$ lsattr /bin/nsh
----i---------e---- /bin/nsh
```

Figure 20 - Nyanshell Docker Screenshot

[9] http://man7.org/linux/man-pages/man1/lsattr.1.html

The immutable flag[10] is set which prompts us to see what commands we can run under sudo.

```
elf@d7df796e80ef:~$ sudo -ll
Matching Defaults entries for elf on d7df796e80ef:
    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 d7df796e80ef:

Sudoers entry:
    RunAsUsers: root
    Options: !authenticate
    Commands:
        /usr/bin/chattr
elf@d7df796e80ef:~$
```

Figure 21 - Nyanshell Docker Screenshot

Great – `chattr`[11] is used to change file attributes so we just need to run it against the `/bin/nsh` shell and run `lsattr` again to make sure it's worked.

```
elf@d7df796e80ef:~$ sudo /usr/bin/chattr -i /bin/nsh
elf@d7df796e80ef:~$ lsattr /bin/nsh
--------------e---- /bin/nsh
elf@d7df796e80ef:~$
```

Figure 22 - Nyanshell Docker Screenshot

Putting it all together helps us solve this particular terminal.

```
elf@d7df796e80ef:~$ > /bin/nsh
elf@d7df796e80ef:~$ cp /bin/bash /bin/nsh
elf@d7df796e80ef:~$ su alabaster_snowball
Password:
Loading, please wait......


You did it! Congratulations!

alabaster_snowball@d7df796e80ef:/home/elf$
```

Figure 23 - Nyanshell Docker Screenshot

[10] https://unix.stackexchange.com/questions/32256/whats-the-meaning-of-output-of-lsattr

[11] https://linux.die.net/man/1/chattr

Powershell

- **Hint**: https://blogs.sans.org/pen-testing/files/2016/05/PowerShellCheatSheet_v41.pdf
- **Location**: The Laboratory
- **URL**: https://docker2019.kringlecon.com/?challenge=powershell

Screenshot

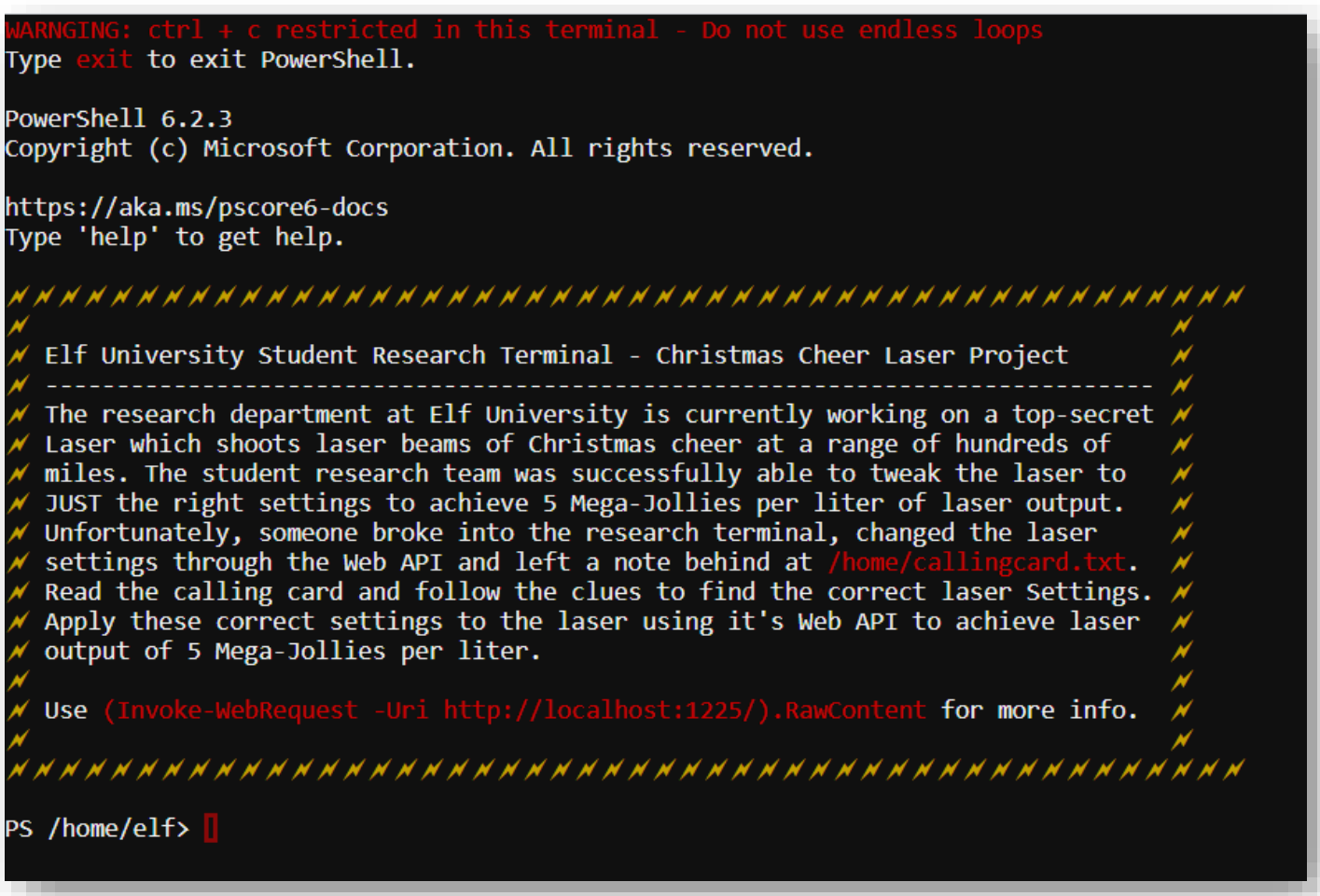


Figure 24 - PowerShell Docker Screenshot

Synopsis

Again- it makes it quite clear in the terminal what to do. We need to follow clues to get the correct laser settings and then to apply them.

Solution

Two things stand out in red so we'll see what happens when we open the note[12] and invoke the web request.

[12] https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-content?view=powershell-6

```
PS /home/elf> Get-Content /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>
```

Figure 25 - PowerShell Docker Screenshot

The hint points to viewing command history.

```
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: Tue, 31 Dec 2019 11:28:08 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>
```

Figure 26 - PowerShell Docker Screenshot

This gives us the various values to amend and the way to amend them. We'll hold onto this for now whilst we follow the clues. First one being the ability to view command history.

Using the `Get-History`[13] command and formatting it, we get presented with the below.

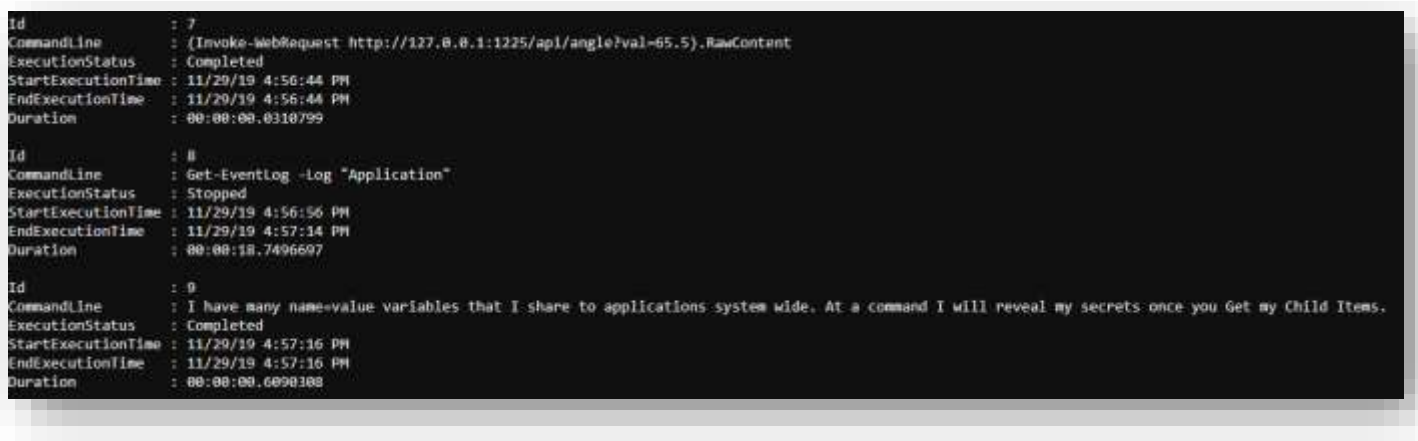


Figure 27 - PowerShell Docker Screenshot

It comes back with several commands but numbers (7) and (9) seem to be the most useful where (7) gives us one of the settings and (9) gives us another clue.

The clue heavily suggests environment variables so we look at what we have available to us.[14] We look at all names and then expand on the one we're interested in.

Figure 28 - PowerShell Docker Screenshot

Another clue. This time we need to recursively search all files under `/etc` and bring back the one with the last write date. Specifically a compressed file.

---

[13] https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/get-history?view=powershell-6

[14] https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_environment_variables?view=powershell-6

```
PS /home/elf> Get-ChildItem -Recurse '\etc' | Sort {$_.LastWriteTime} | select -last 1
Get-ChildItem : Access to the path '/etc/ssl/private' is denied.
At line:1 char:1
+ Get-ChildItem -Recurse '\etc' | Sort {$_.LastWriteTime} | select -las ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+ CategoryInfo          : PermissionDenied: (/etc/ssl/private:String) [Get-ChildItem], UnauthorizedAccessException
+ FullyQualifiedErrorId : DirUnauthorizedAccessError,Microsoft.PowerShell.Commands.GetChildItemCommand



    Directory: /etc/apt

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
--r---          12/31/19 11:31 AM        5662902 archive
```

Figure 29 - PowerShell Docker Screenshot

The archive was found using `Get-ChildItem`[15] and sorting.

Copying it to the home folder and extracting the contents seems like the way to go and rather than running a command per line, we can concatenate them using a semi colon.

```
PS /home/elf> Copy-Item "\etc\apt\archive" -Destination "\home\elf"; Expand-Archive archive -DestinationPath .\riddle; cd ./riddle/;
PS /home/elf/riddle> dir


    Directory: /home/elf/riddle

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----          12/31/19 11:52 AM                refraction

PS /home/elf/riddle> cd ./refraction/
PS /home/elf/riddle/refraction> dir


    Directory: /home/elf/riddle/refraction

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

Figure 30 - PowerShell Docker Screenshot

This then give us another setting and another clue

[15] https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-childitem?view=powershell-6

```
PS /home/elf/riddle/refraction> Get-Content 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/riddle/refraction> chmod +x ./runme.elf
PS /home/elf/riddle/refraction> ./runme.elf
refraction?val=1.867
PS /home/elf/riddle/refraction>
```

Figure 31 - PowerShell Docker Screenshot

The clue points to going back to our home folder and seeing what's there.

```
PS /home/elf> dir


    Directory: /home/elf

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-r---          12/13/19  5:15 PM                depths
d-----          12/31/19 11:52 AM                riddle
------          12/31/19 11:52 AM        5662902 archive
--r---          12/13/19  4:29 PM           2029 motd
```

Figure 32 - PowerShell Docker Screenshot

So there is a folder called depths. MD5 identity? Is it based off filename, filepath or contents. We opt for the file itself and utilise `Get-FileHash`[16].

```
PS /home/elf> Get-ChildItem -Path "/home/elf/depths/" -Recurse | where { $_.PSIsContainer } | where { (Get-FileHash -Path $_.FullName -Algorithm MD5).Hash -eq "25520151A320B5B0D21561F92C8F6224" }


    Directory: /home/elf/depths/produce


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
--r---          11/18/19  7:53 PM            224 thhyShll.txt
```

Figure 33 - PowerShell Docker Screenshot

This give us another file to look into, the results of which can be seen below.

[16] https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-filehash?view=powershell-6

Figure 34 - PowerShell Docker Screenshot

Another setting and another clue. The clue points to sorting all files under /depth/ and bring back the longest. We initially did the following command which brought us back what we needed.

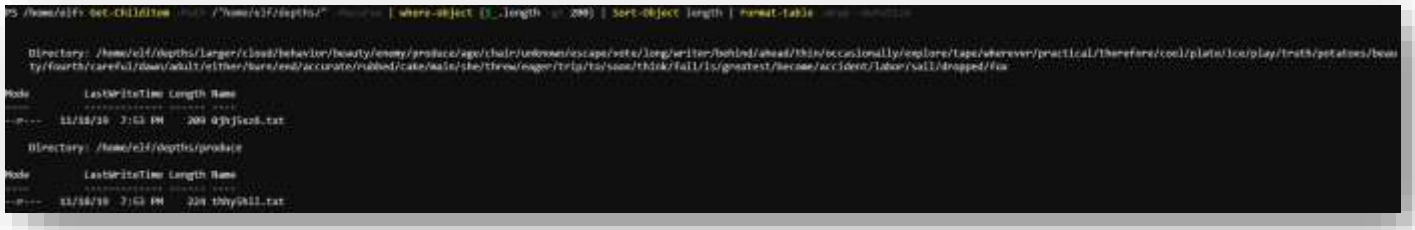


Figure 35 - PowerShell Docker Screenshot

However, on second glance, we got a bit lucky as there could have been hundreds of files with a length greater than 200 so the script was amended to take this into account by, as the clue suggested, getting all fullnames[17] and selecting the longest one.

Figure 36 - PowerShell Docker Screenshot

Either way, this gives us another text file to read.

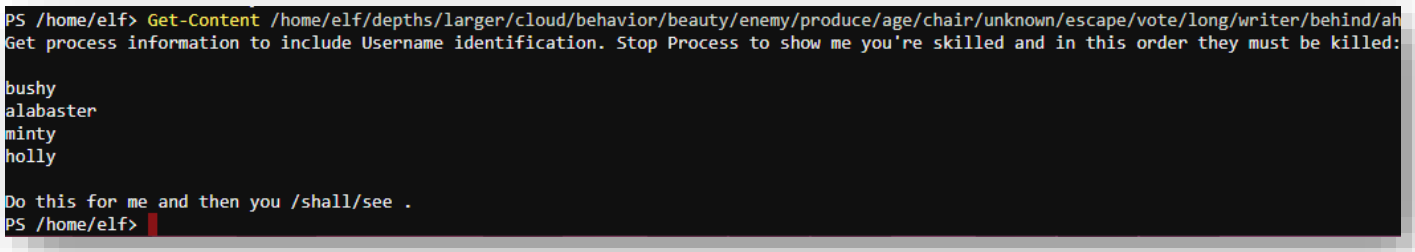


Figure 37 - PowerShell Docker Screenshot

[17] https://stackoverflow.com/questions/16551047/how-to-display-the-length-of-a-fullname

So we need to identify running processes, the relevant usernames and kill them. This should allow us to read `/shall/see` which is currently unavailable.

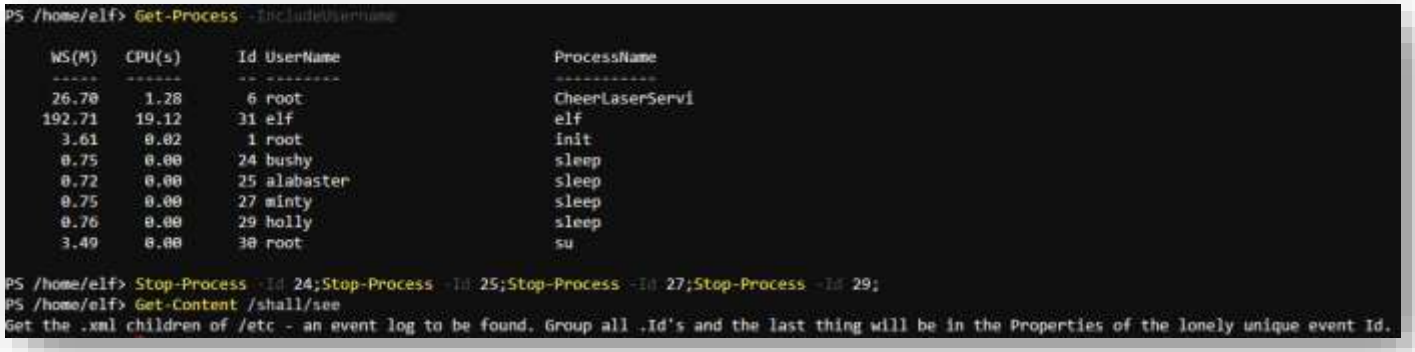


Figure 38 - PowerShell Docker Screenshot

So we listed the processes[18], killed[19] them in the order requested and viewed the contents of the file to give us another clue.

The last clue pointed to us finding a XML file and then processing the data within it. To find the XML we leveraged what we have use before.

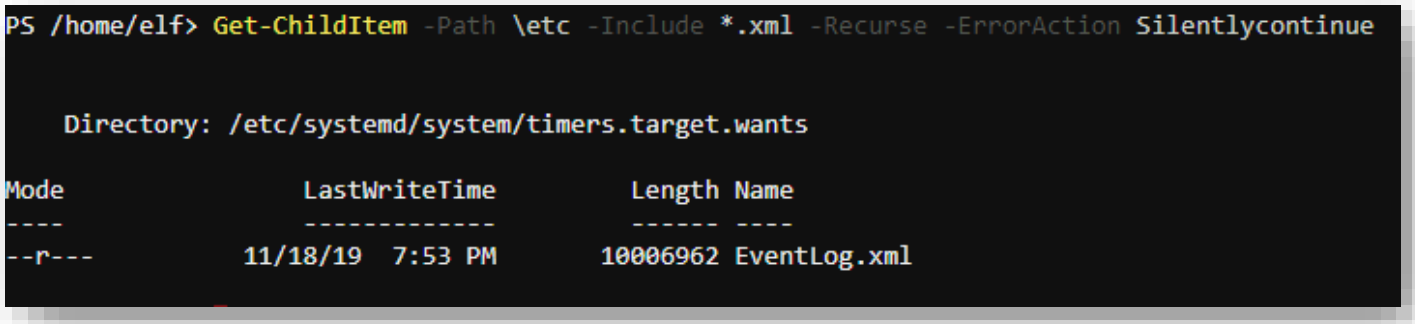


Figure 39 - PowerShell Docker Screenshot

Any errors weren't outputted in case we got a load of permissions related warnings. Let's just check what's inside the file.

[18] https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-process?view=powershell-6

[19] https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/stop-process?view=powershell-6

```
PS /etc/systemd/system/timers.target.wants> Get-Content ./EventLog.xml -Head 20
<Objs Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04">
  <Obj RefId="0">
    <TN RefId="0">
      <T>System.Diagnostics.Eventing.Reader.EventLogRecord</T>
      <T>System.Diagnostics.Eventing.Reader.EventRecord</T>
      <T>System.Object</T>
    </TN>
    <ToString>System.Diagnostics.Eventing.Reader.EventLogRecord</ToString>
    <Props>
      <I32 N="Id">3</I32>
      <By N="Version">5</By>
      <Nil N="Qualifiers" />
      <By N="Level">4</By>
      <I32 N="Task">3</I32>
      <I16 N="Opcode">0</I16>
      <I64 N="Keywords">-9223372036854775808</I64>
      <I64 N="RecordId">2194</I64>
      <S N="ProviderName">Microsoft-Windows-Sysmon</S>
      <G N="ProviderId">5770385f-c22a-43e0-bf4c-06f5698ffbd9</G>
      <S N="LogName">Microsoft-Windows-Sysmon/Operational</S>
```

Figure 40 - PowerShell Docker Screenshot

Using a number of resources[2021] we decide to group on `<I32 N=”Id”>XXX</I32>` taking advantage of XML capabilities[22] within PowerShell.

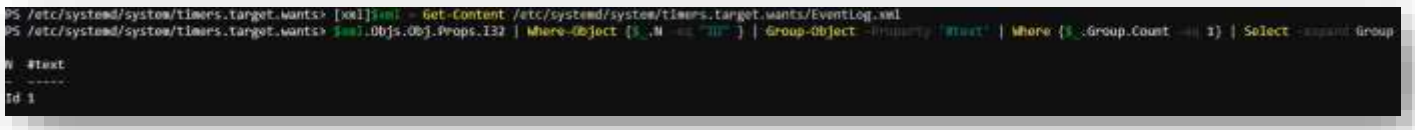


Figure 41 - PowerShell Docker Screenshot

This just gives us one record back which is good news. Now to bring back the properties of the relevant node.

---

[20] https://docs.microsoft.com/en-us/windows/win32/wes/eventschema-eventid-systempropertiestype-element

[21] https://docs.microsoft.com/en-us/windows/win32/wes/eventschema-schema

[22] https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/select-xml?view=powershell-6

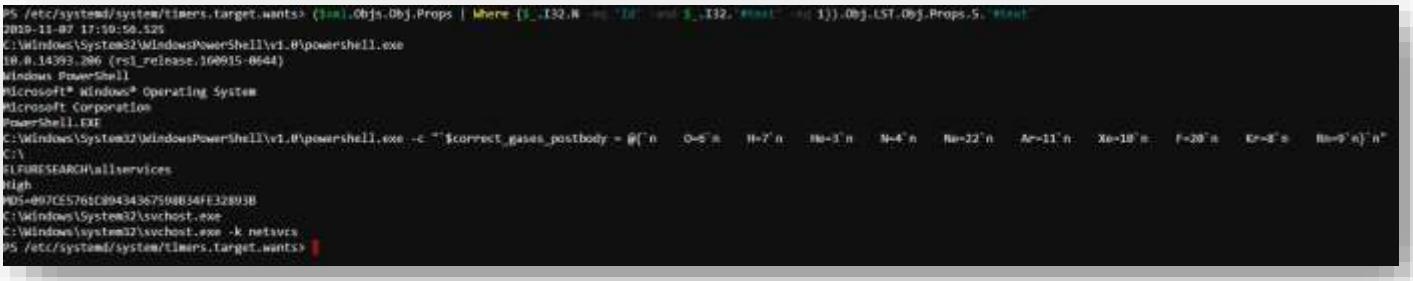


Figure 42 - PowerShell Docker Screenshot

It gives us our final setting which after the longest terminal, completes the terminal.

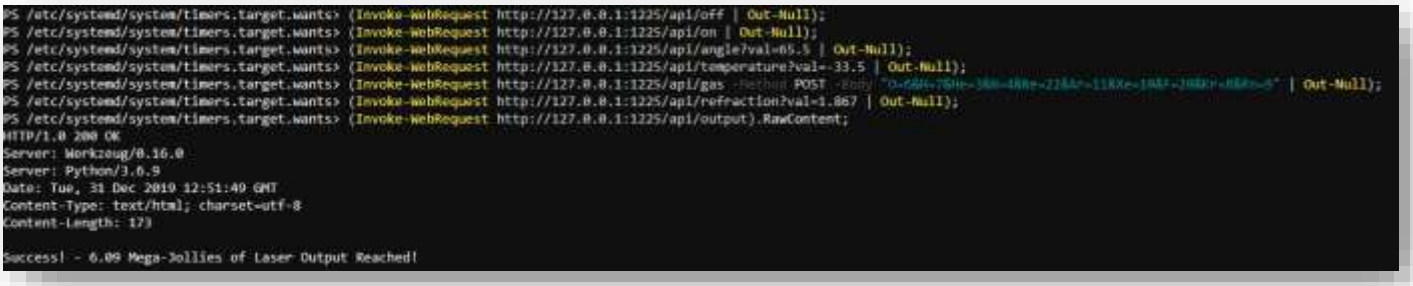


Figure 43 - PowerShell Docker Screenshot

Iptables

- **Hint**: https://upcloud.com/community/tutorials/configure-iptables-centos
- **Location**: Student Union
- **URL**: https://docker2019.kringlecon.com/?challenge=iptables

Screenshot

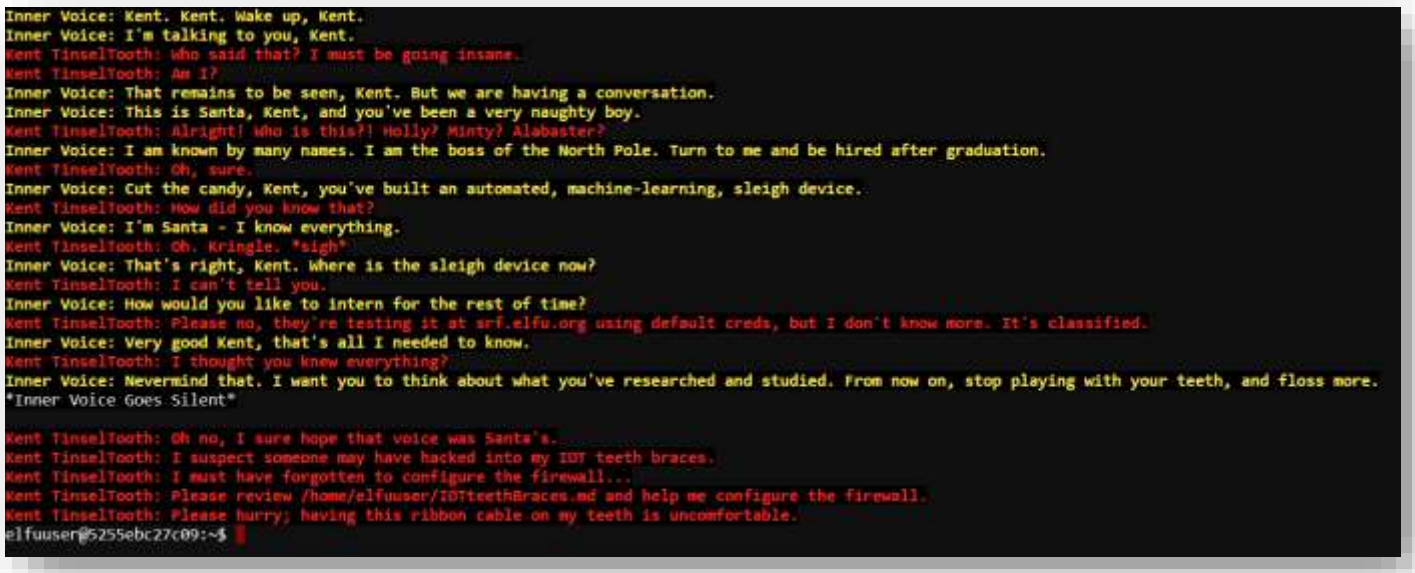


Figure 44 - Iptables Docker Screenshot

Synopsis

Open opening /home/elfuser/IOTTeethBraces.md we are asked to put a bunch of iptables rules in to prevent further hacks.

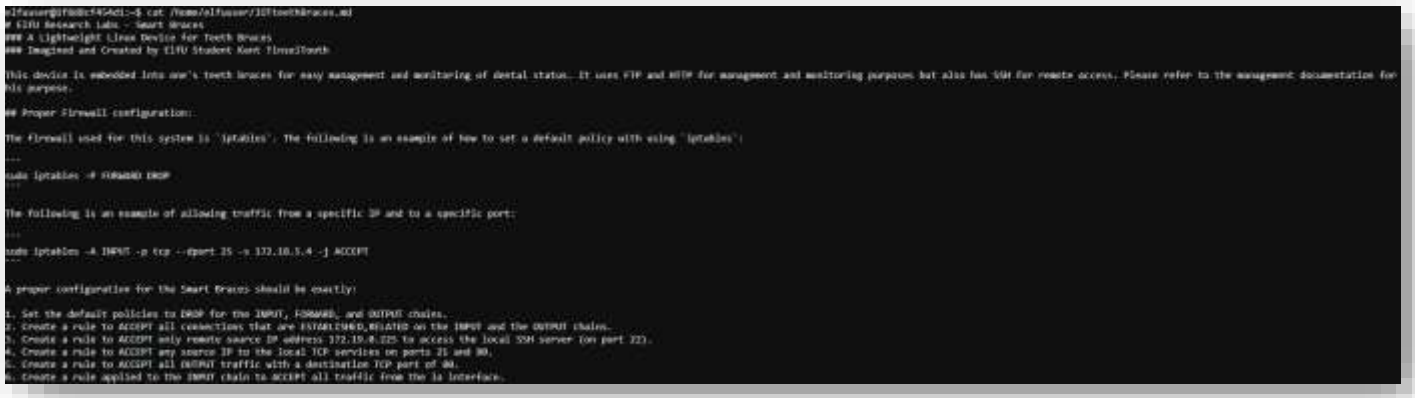


Figure 45 - Iptables Docker Screenshot

Solution

Seems fairly straightforward when using the hint, the nudge in the terminal and other resources[23]. The rules are added and just checked at the end and before we knew it, this terminal was completed

```
elfuuser@2de9c34d594c:~$ sudo iptables -P INPUT DROP
elfuuser@2de9c34d594c:~$ sudo iptables -P FORWARD DROP
elfuuser@2de9c34d594c:~$ sudo iptables -P OUTPUT DROP
elfuuser@2de9c34d594c:~$ sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
elfuuser@2de9c34d594c:~$ sudo iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
elfuuser@2de9c34d594c:~$ sudo iptables -A INPUT -p tcp --dport 22 -s 172.19.0.225 -j ACCEPT
elfuuser@2de9c34d594c:~$ sudo iptables -A INPUT -p tcp --dport 21 -j ACCEPT
elfuuser@2de9c34d594c:~$ sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
elfuuser@2de9c34d594c:~$ sudo iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT
elfuuser@2de9c34d594c:~$ sudo iptables -A INPUT -i lo -j ACCEPT
elfuuser@2de9c34d594c:~$ sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
ACCEPT     tcp  --  172.19.0.225         anywhere             tcp dpt:22
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:21
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:80
ACCEPT     all  --  anywhere             anywhere

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:80
elfuuser@2de9c34d594c:~$ Kent TinselTooth: Great, you hardened my IOT Smart Braces firewall!


/usr/bin/inits: line 10:  1082 Killed                  su elfuuser
```

Figure 46 - Iptables Docker Screenshot

[23] https://www.digitalocean.com/community/tutorials/iptables-essentials-common-firewall-rules-and-commands

Jq

- **Hint**: https://pen-testing.sans.org/blog/2019/12/03/parsing-zeek-json-logs-with-jq-2
- **Location**: Sleigh Workshop
- **URL**: https://docker2019.kringlecon.com/?challenge=jq

Screenshot

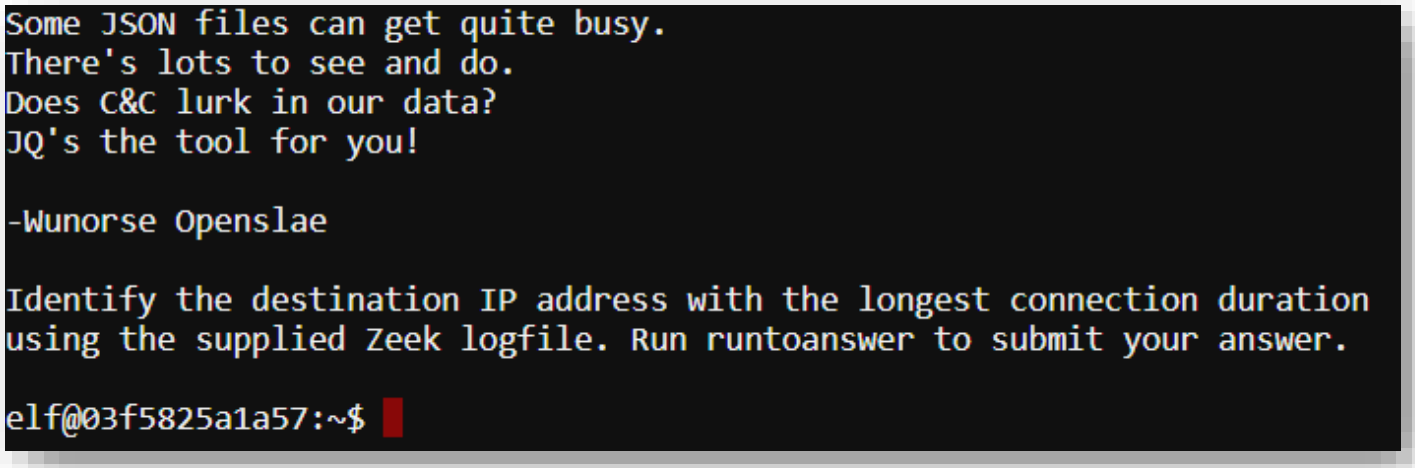


Figure 47 - Jq Docker Screenshot

Synopsis

Again, the terminal is pretty descriptive in what it needs – from a Zeek log file in JSON format, get the relevant record that has the longest (largest) duration and bring back the relative IP.

Solution

The code below sorts the JSON file by duration, reverses the results and then brings out the first node

```
elf@70dc32948307:~$ cat conn.log | jq -s 'sort_by(.duration) | reverse | .[0]'
{
  "ts": "2019-04-18T21:27:45.402479Z",
  "uid": "CmYAZn10sInxVD5WWd",
  "id.orig_h": "192.168.52.132",
  "id.orig_p": 8,
  "id.resp_h": "13.107.21.200",
  "id.resp_p": 0,
  "proto": "icmp",
  "duration": 1019365.337758,
  "orig_bytes": 30781920,
  "resp_bytes": 30382240,
  "conn_state": "OTH",
  "missed_bytes": 0,
  "orig_pkts": 961935,
  "orig_ip_bytes": 57716100,
  "resp_pkts": 949445,
  "resp_ip_bytes": 56966700
}
```

Figure 48 - Jq Docker Screenshot

We get an IP of 13.107.21.200 which we then pass to the `runtoanswer` command.

```
elf@70dc32948307:~$ 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!
```

Figure 49 - Jq Docker Screenshot

Terminals (Non-Docker)

Keypad

- URL: https://keypad.elfu.org/

Screenshot

Figure 50 - Keypad Screenshot

Synopsis

We need to get the correct access code to enter the building. A few clues will help us on our way which are:

- One digit is repeated once.
- The code is a prime number.
- You can probably tell by looking at the keypad which buttons are used.

Solution

We can narrow down our searches drastically by using the clues mentioned above, especially with it looking like only the numbers 1, 3 and 7 are used on account of them looking the most worn out.

Instinctively 1337 is thought of but it's not a prime number on account of it being divisible by 7 and 191.

When trying out a number, we can see a GET request is made to a URL which contains our guess which indicates we may be able to brute force the actual solution

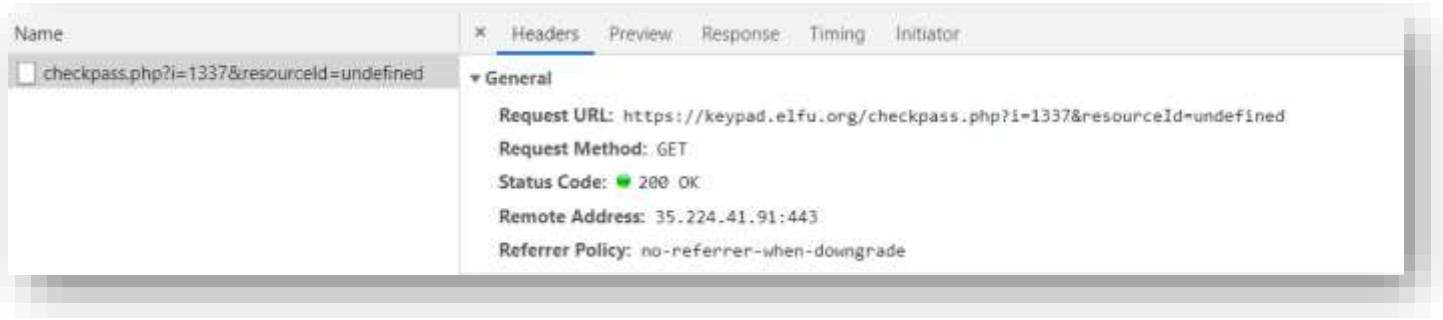


Figure 51 - Keypad Screenshot

A simple little python script is created that produces values conforming to the clues mentioned and tried them against the GET request.

```python
import requests
import json
import sys

def has_doubles(n): # Checking if the number has 2 digits the same
    return len(set(str(n))) < len(str(n))

for num in range(1337,7713): # Since the lowest number can be 1337 and the highest can be 7713, limit the searches
    prime = True
    for i in range(2,num): # Check if it's a prime number
        if (num%i==0):
            prime = False
    if prime:
        s = set(str(num))
        if s.issuperset("137"): #If it contains these numbers
            if has_doubles(num):
                r = requests.get('https://keypad.elfu.org/checkpass.php?i='+str(num)+'&resourceId=undefined')
                json_data = json.loads(r.text)
                if json_data["success"]:
                    print('[+] '+str(num)+' worked')
                    sys.exit(0)
                else:
                    print('[+] '+str(num)+' failed')
```

Figure 52 - Keypad Screenshot

The script is run and a short time later, we have our answer

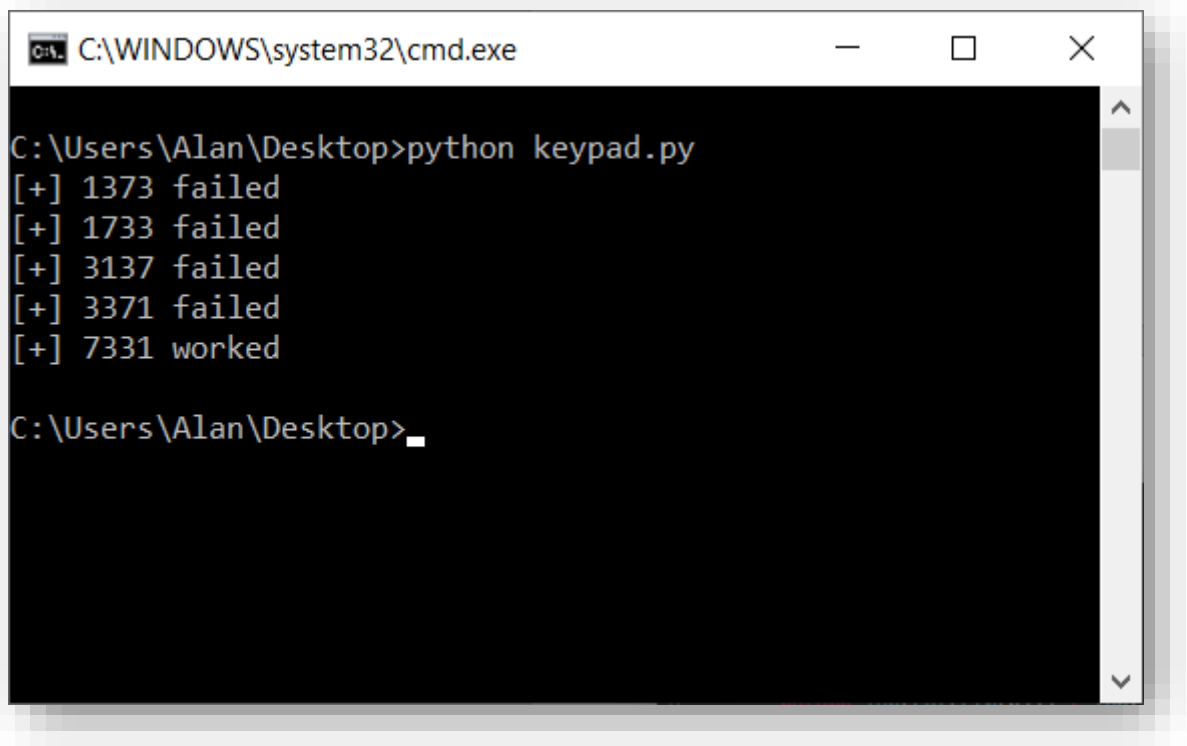


Figure 53 - Keypad Screenshot

Trail

- **URL:** https://trail.elfu.org/gameselect/?playerid=JebediahSpringfield

Screenshot

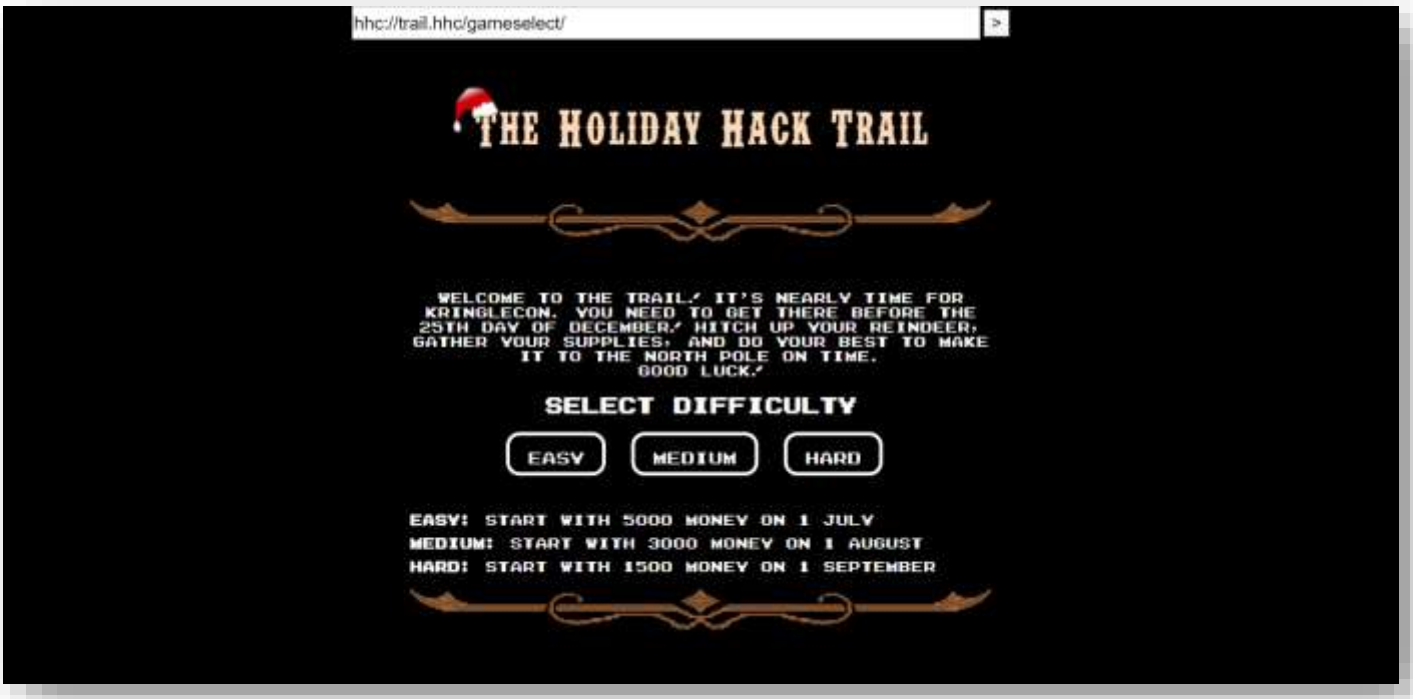


Figure 54 - Trail Screenshot

Synopsis

There are 3 levels to the game. Completing it on hard will give us clues to completing the crate challenge.

Solution

Before continuing, viewing the source of the main page gives us clues as to what to look for

Figure 55 - Trail Screenshot

A quick run through of the game reveals the distance to travel is 8000. Once we get down to zero, the game is complete.

Easy

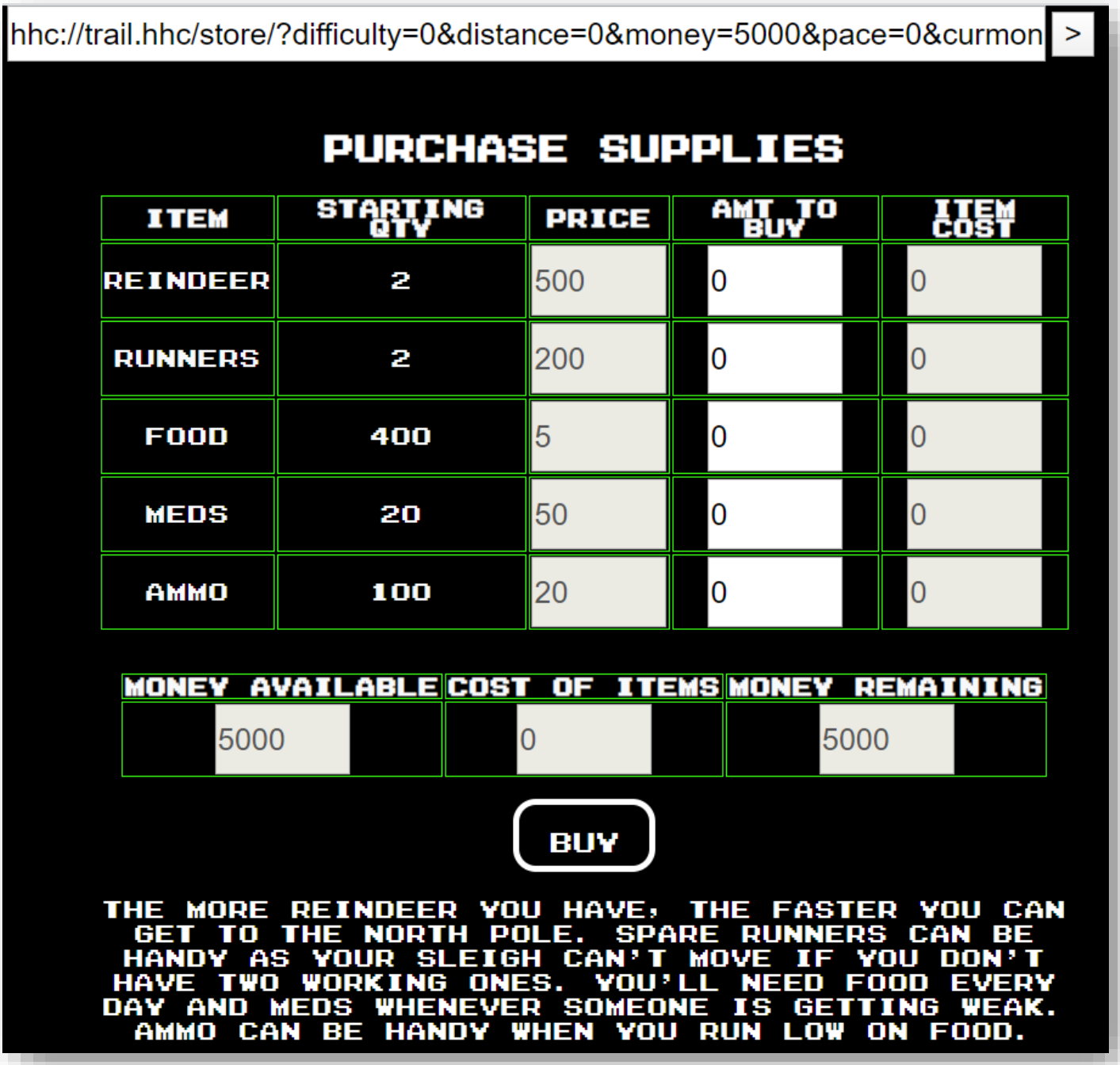


Figure 56 - Trail Screenshot

The HTML comments give the game away for this one, if the input field at the top didn't already. Amending the distance to 8000 and submitting it, along with amending any other variables, gave us that amount to play with for the various game variables. Unfortunately, we can't amend the game difficulty for the other levels.

In the next screenshot, we have amended the distance travelled to the full 8000 and the money to 50,000 rather than 5,000.

Figure 57 - Trail Screenshot

Clicking om “Buy” takes us to the following screen.

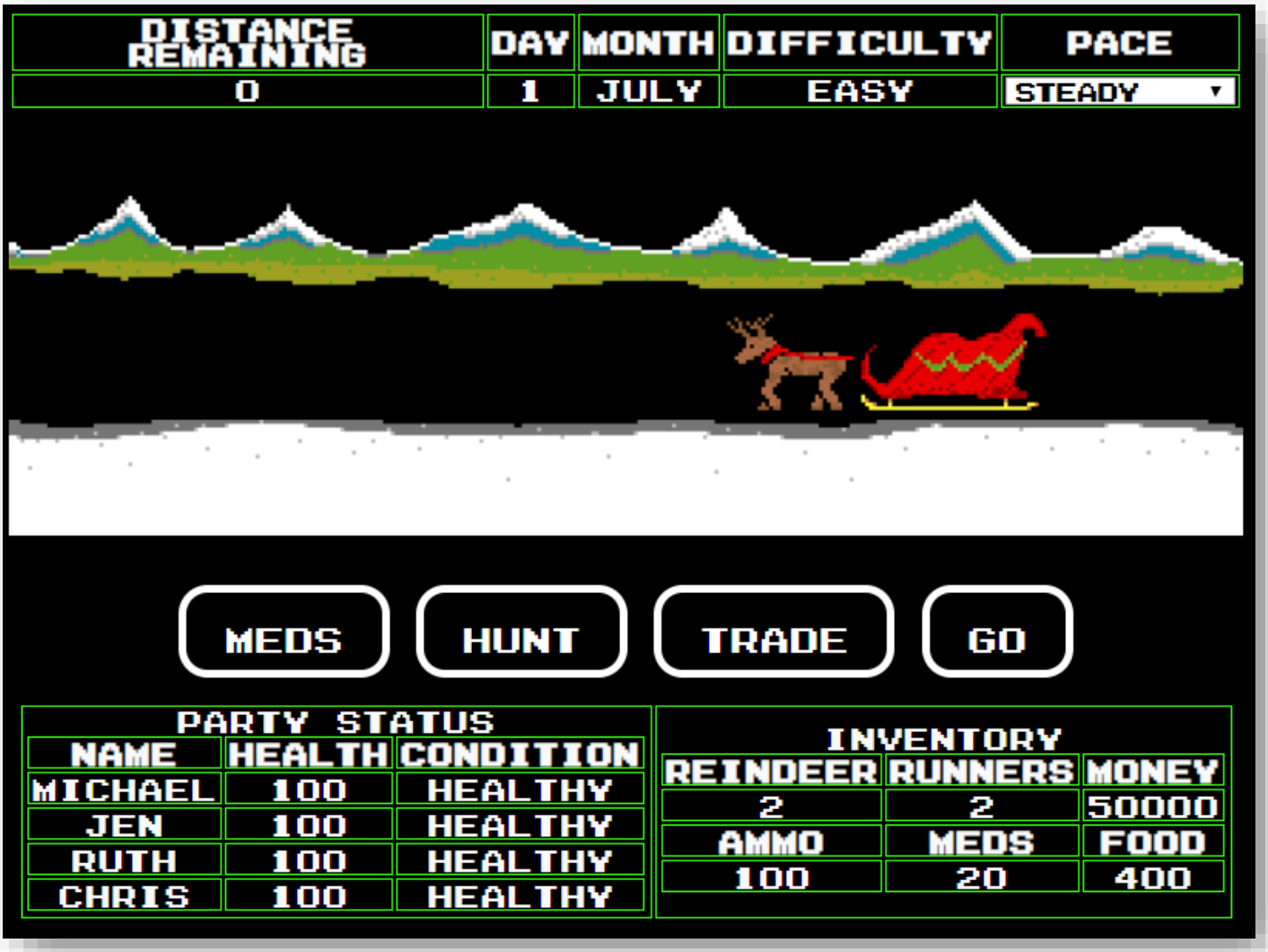


Figure 58 - Trail Screenshot

As you can see, the distance remaining is zero. This means that we have already covered the distance so hitting go should mean we have completed the task when we hit “Go”. This is confirmed in the next screenshot.

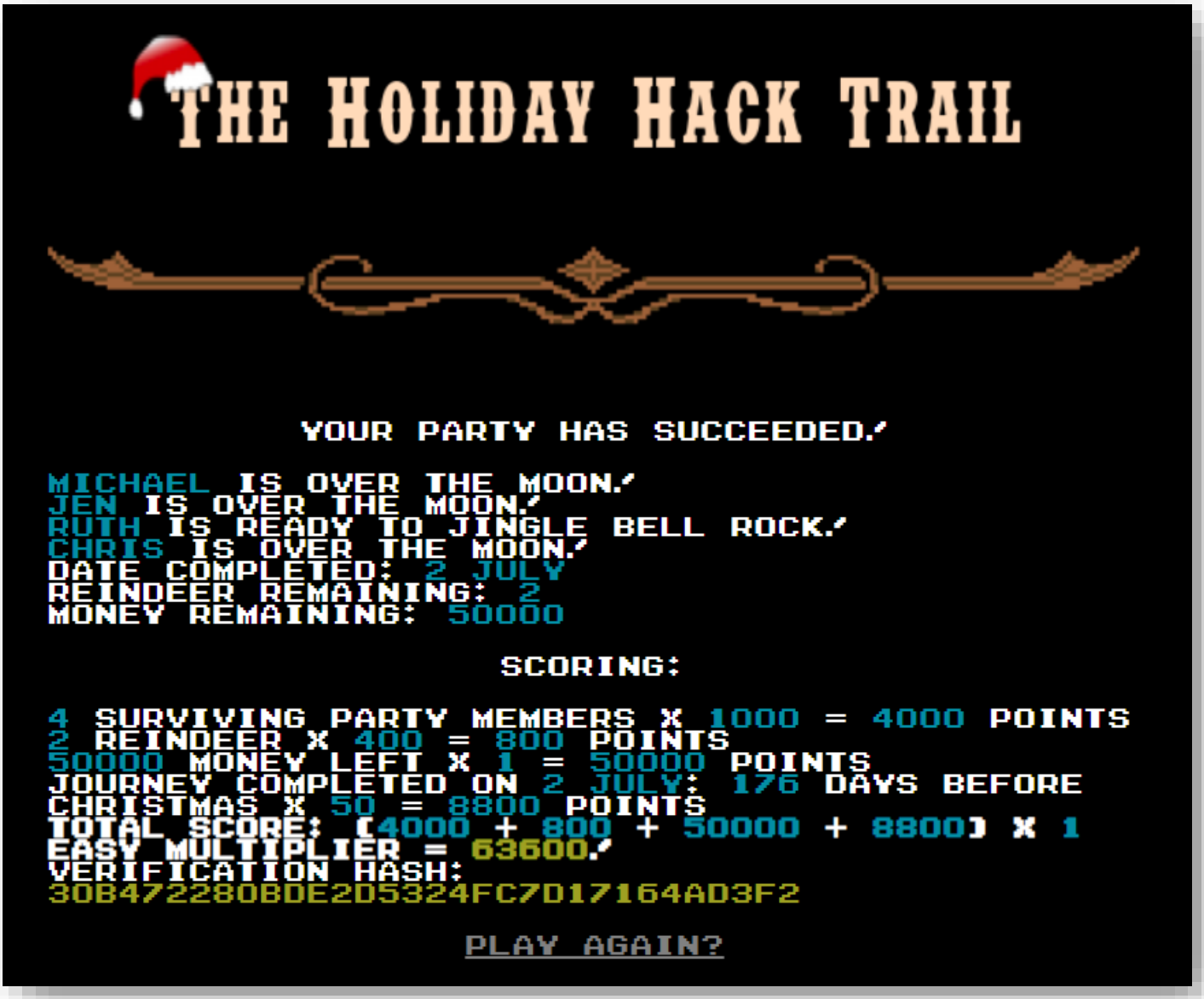


Figure 59 - Trail Screenshot

Medium

The clue for this level suggests that instead of using GET to attain the variables, the game is not using POST. We inspect the page which reveals the following:

```
▼<div id="statusContainer">
    <input type="hidden" name="difficulty" class="difficulty" value="1">
    <input type="hidden" name="money" class="difficulty" value="3000">
    <input type="hidden" name="distance" class="distance" value="0">
    <input type="hidden" name="curmonth" class="difficulty" value="8">
    <input type="hidden" name="curday" class="difficulty" value="1">
```

Figure 60 - Trail Screenshot

This value can be amended here or via a proxy like Burp. As we're already here, we might as well do this here.

```
<input type="hidden" name="difficulty" class="difficulty" value="1">
<input type="hidden" name="money" class="difficulty" value="3000">
<input type="hidden" name="distance" class="distance" value="8000"> == $0
<input type="hidden" name="curmonth" class="difficulty" value="8">
<input type="hidden" name="curday" class="difficulty" value="1">
```

Figure 61 - Trail Screenshot

The game is played as before and we complete the game in similar fashion to before.

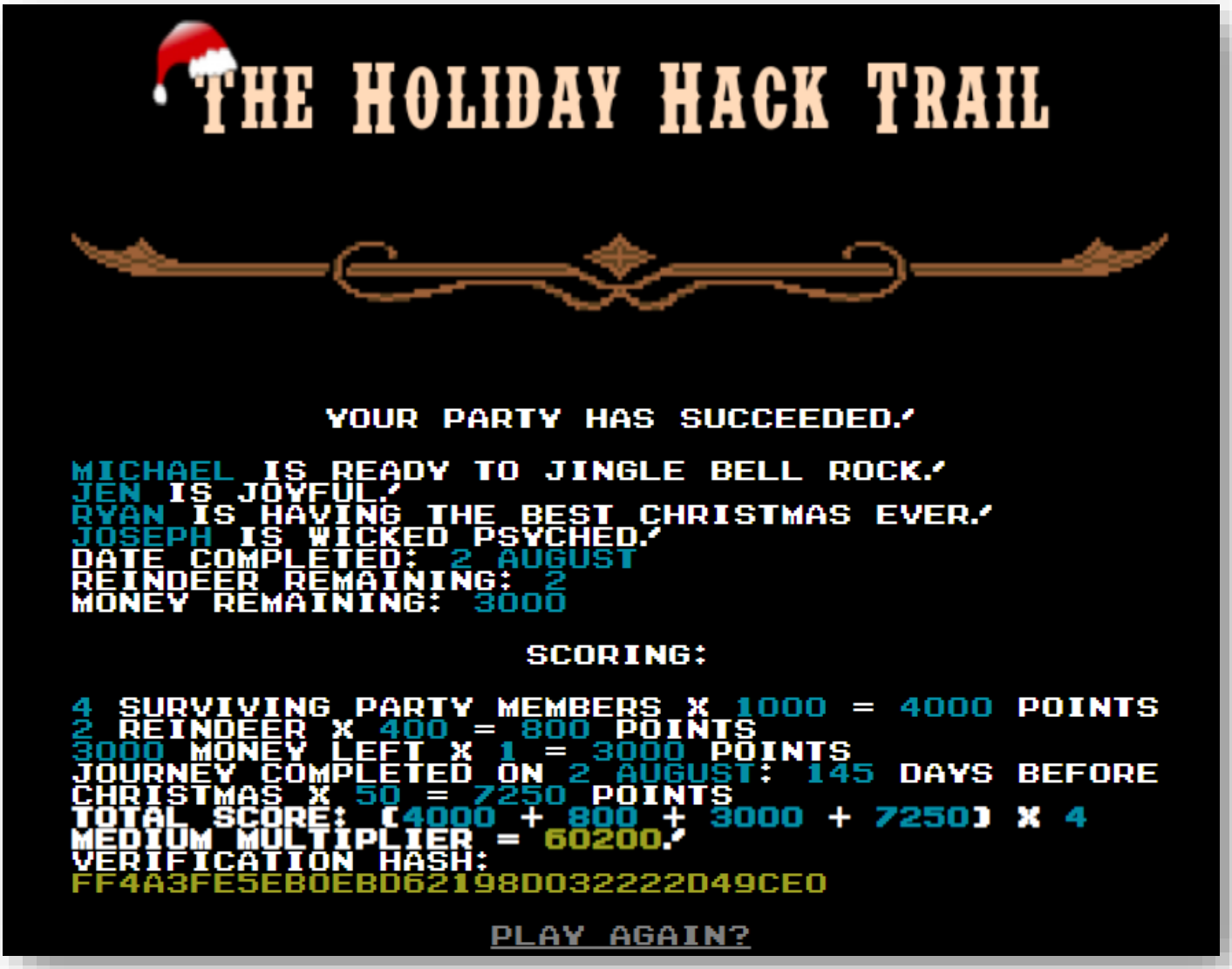


Figure 62 - Trail Screenshot

Hard

The game is started as before and looking at the source code this time, we see there is a hash field.

```
    <input type="hidden" name="reindeer" class="reindeer" value="2">
    <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="bc573864331a9e42e4511de6f678aa83">
</div>
```

Figure 63 - Trail Screenshot

A string of 32 characters could be a MD5 hash and a quick lookup[24] confirms this with the decoded value being 1626. This value comes from combining several of the values within the form.

Type	Values
money	1500
distance	0
curmonth	9
curday	1
reindeer	2
runners	2
ammo	10
meds	2
food	100
Total	**1626**

Playing the game to see if any more details can be found, we see that whatever we buy, the hash decreases by the cost of that item less 1 i.e. if we buy something for 500, the hash decreases by 499.

Returning to the start, if we increase the distance as we did in the medium level game but then increase the decrypted hash value by 8000 as well, and the MD5 encrypt it, we should be in the same place where we were for the previous level.

1626 + 8000 = 9626 which is **649d45bf179296e31731adfd4df25588** when passed through MD5.

```
    <input type="hidden" name="reindeer" class="reindeer" value="2">
    <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="bc573864331a9e42e4511de6f678aa83">
</div>
```

Figure 64 - Trail Screenshot

As with the previous two levels, when we proceed with the game, distance is set to zero and the game is complete on the first run.

[24] https://www.md5online.org/md5-decrypt.html

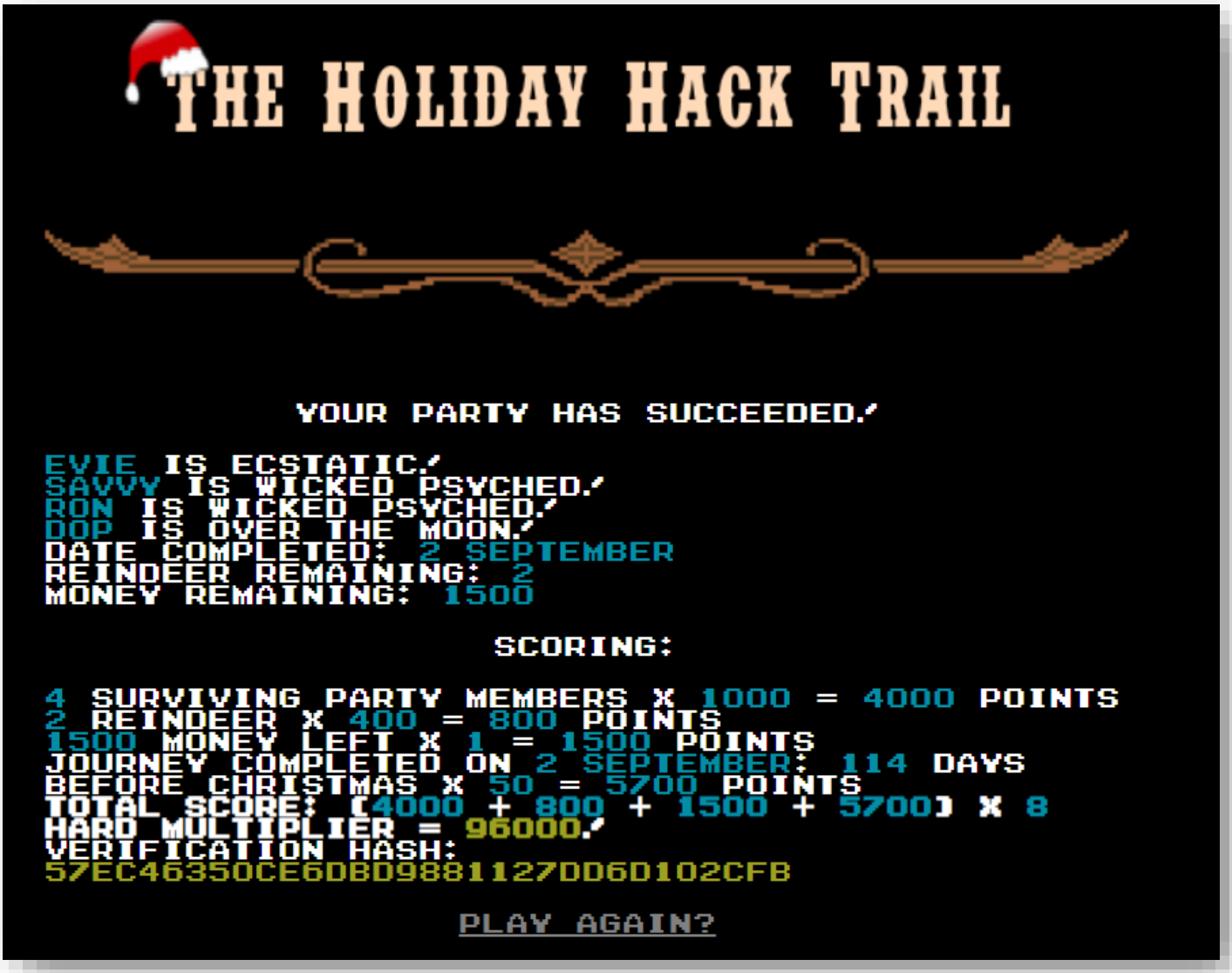


Figure 65 - Trail Screenshot

Viewing the source code of the page reveals the below comments which will be useful for another objective.

```
<!-- 1 - When I'm down, my F12 key consoles me
2 - Reminds me of the transition to the paperless naughty/nice list...
3 - Like a present stuck in the chimney!  It got sent...
4 - We keep that next to the cookie jar
5 - My title is toy maker the combination is 12345
6 - Are we making hologram elf trading cards this year?
7 - If we are, we should have a few fonts to choose from
8 - The parents of spoiled kids go on the naughty list...
9 - Some toys have to be forced active
10 - Sometimes when I'm working, I slide my hat to the left and move odd things onto my scalp! -->
```

Figure 66 - Trail Screenshot

Graylog

- **URL**: https://graylog.elfu.org/

Screenshot

Synopsis

The URL we get given is https://incident.elfu.org/ but to go through the objective without https://report.elfu.org/ coming up, you can use https://incident.elfu.org/

Solution

The feedback on the website gives a good account of what happened and why so we won't go into detail ourselves.

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?

We can track the creation of a file by looking at records where the event ID is equal to 2[25] and where the process is Firefox.

This gives us: `EventID:2 AND ProcessImage:/.+firefox.+/`

Unfortunately this gives is 21 entries but we can narrow this down by excluding .temp files with the following search: `EventID:2 AND ProcessImage:/.+firefox.+/ AND NOT TargetFilename:/.+\.temp/`

Now we only have one result that gives us our answer

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

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?

Given the file was executed, we can just use this as our `ProcessImage` and see what IP and port it connected to. Using the search `ProcessImage:"C:\\Users\\minty\\Downloads\\cookie_recipe.exe" AND DestinationIp:/.+/` we get the following:

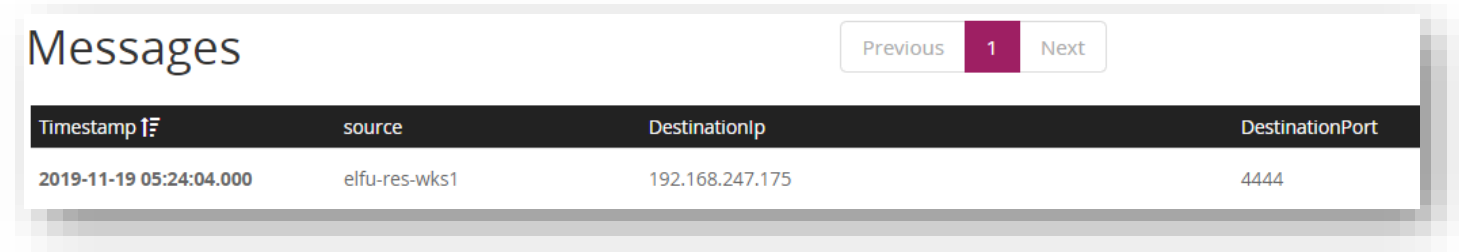


Figure 67 - Garylog Screenshot

Answer: 192.168.247.175:4444

[25] https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon

Question 3

Since commands have an Event ID of 1, and the attacker initially used the downloaded executable, we can filter on both Event ID and the `ParentProcessImage`. Ordering the results by timestamp will help us paint a better picture of what happened and when. The search used was `ParentProcessImage:"C:\\Users\\minty\\Downloads\\cookie_recipe.exe" AND EventID:1`

The result of which can be seen below

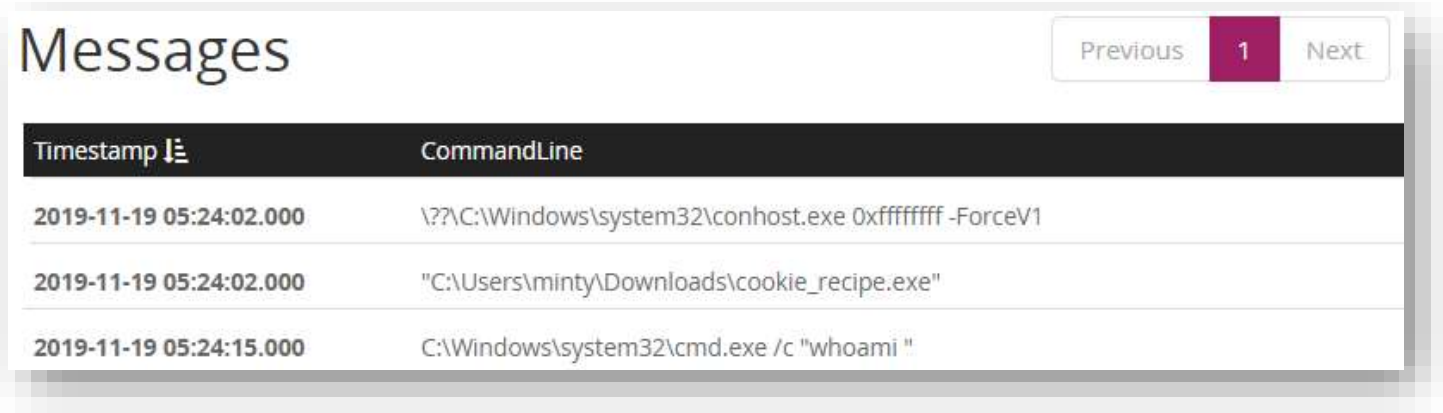


Figure 68 - Graylog Screenshot

Answer: whoami

Question 4

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

If we continue with the same search as before and go down the list, we see the service we are after

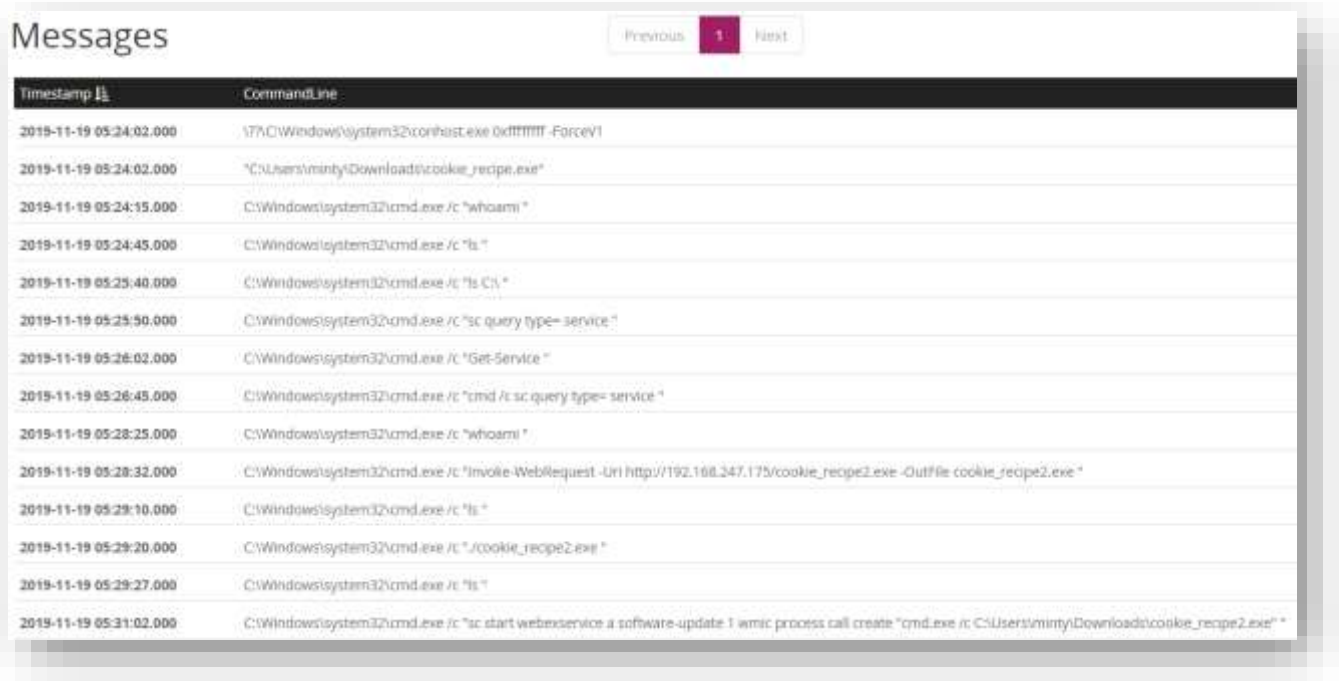


Figure 69 - Graylof Screenshot

Answer: webexservice

Question 5

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

In the previous question, we saw that `webexservice` was used to download a file to `C:\Users\minty\Downloads\cookie_recipe2.exe`. We therefore use this as the `ParentProcessImage` instead as follows: `ParentProcessImage:"C:\\Users\\minty\\Downloads\\cookie_recipe2.exe"`

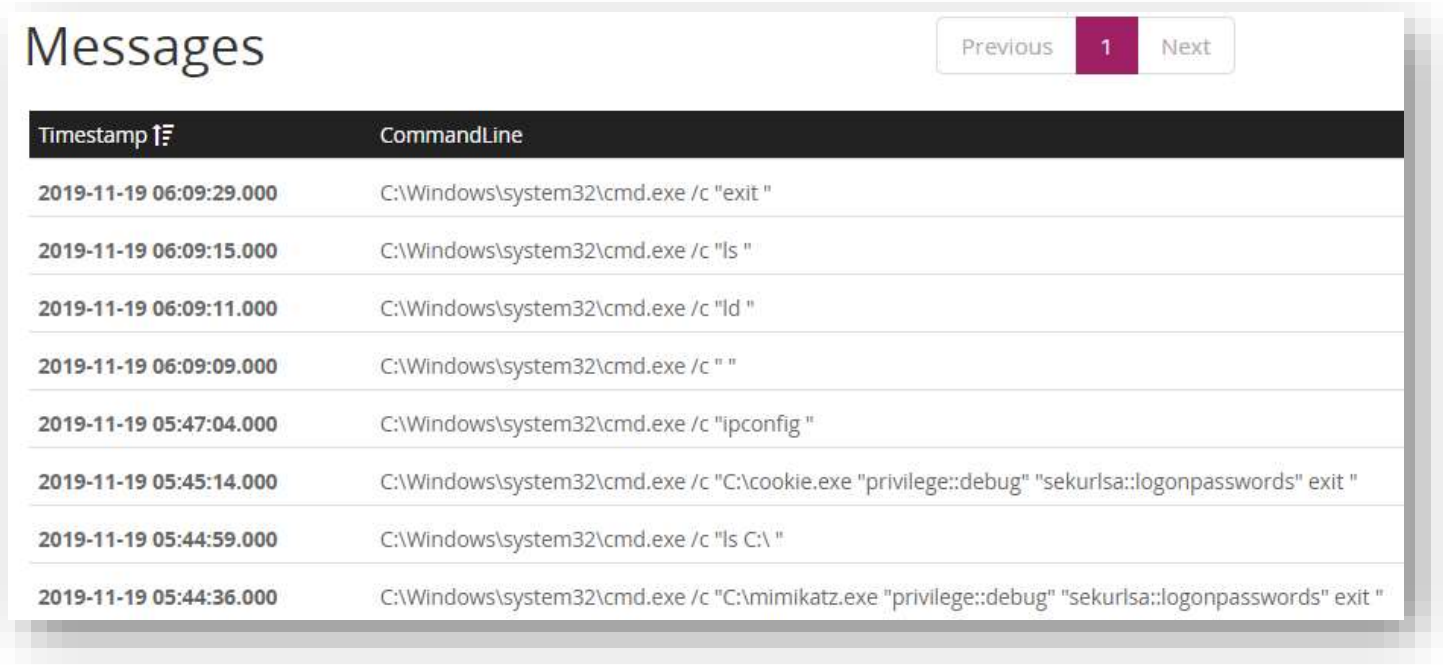


Figure 70 - Graylog Screenshot

Going down the list, we see the file being used. A few steps after mimikatz[26] is also used.

Answer: C:\cookie.exe

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?

A successfully logged-in account is logged under Event ID 4624[27]. Knowing the local address they'll be pivoting from allows us to build the following search: `EventID:4624 and SourceNetworkAddress:192.168.247.175 AND AccountName:/.+/`

This will bring back several results where they all share the same AccountName.

[26] https://github.com/gentilkiwi/mimikatz

[27] https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4624

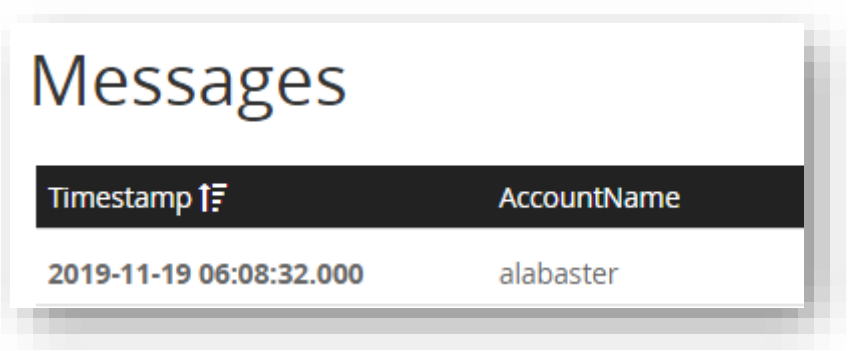


Figure 71 - Graylof Screenshot

Answer: alabaster

Question 7

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

Logon type 10[28] denotes a RDP connection so a pretty simple search: `LogonType:10 AND DestinationHostname:/.+/`

Figure 72 - Graylog Screenshot

Answer: 06:04:28

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?

Some things to note:

1. They have RDP access to the box via a GUI
2. Navigating the file system suggests they will be using explorer

[28] http://techgenix.com/logon-types/

Based on this, we can say that logon type 3 is used[29] and we'll be looking for Event ID 4624[30]. This gives us a fairly simple search: `LogonType:3 AND EventID:4624`

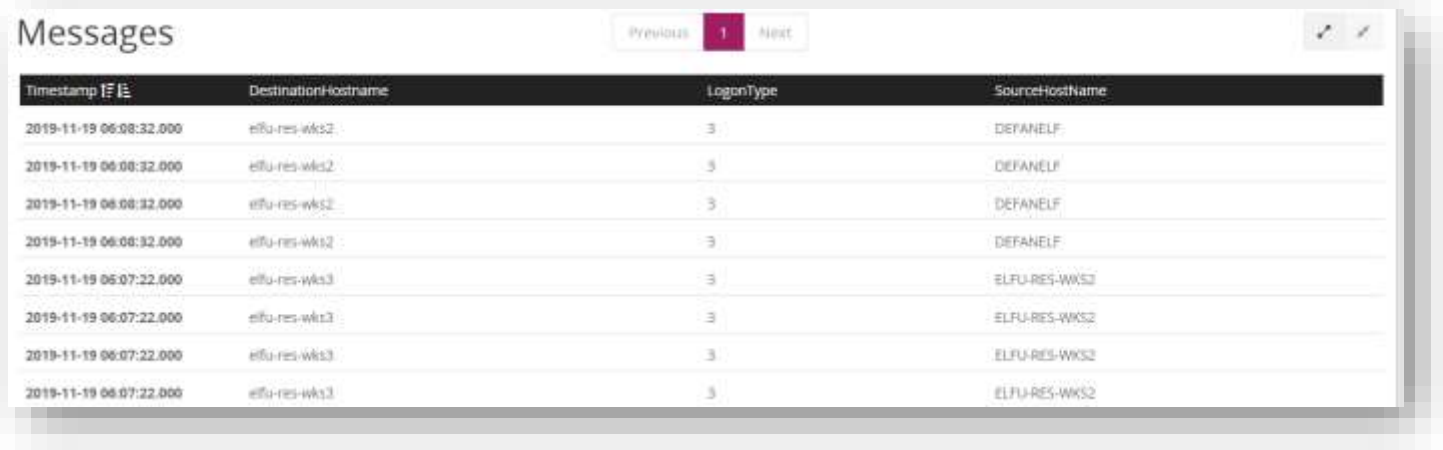


Figure 73 - Graylog Screenshot

Given we're looking at workstations, this makes the answer stand out a bit more.

Answer: ELFU-RES-WKS2,elfu-res-wks3,3

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 can start building on previous searches with an Event ID of 2[31] and a source of the 2nd workstation to give us: `source:elfu\-res\-wks2 AND EventID:2`

This gives is over 70 results, most of which are under common file paths so we amend the search to filter these out: `source:elfu\-res\-wks2 AND EventID:2 AND NOT TargetFilename:/.+AppData.+/ AND NOT TargetFilename:/.+ProgramData.+/`

It gives us two results, one of which is the fairly obvious answer.

Figure 74 - Graylog Screenshot

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

---

[29] http://techgenix.com/logon-types/

[30] https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4624

[31] https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventid=90002

Question 10

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

To exfiltrate the document, the name of it would be used in a command line so we can use a fairly simple search for this: `CommandLine:/.+super_secret_elfu_research.+/`

Figure 75 - Graylog Screenshot

A PowerShell command is invoked to post data to pastebin so pivoting on this gives our final answer: `DestinationHostname:pastebin.com`

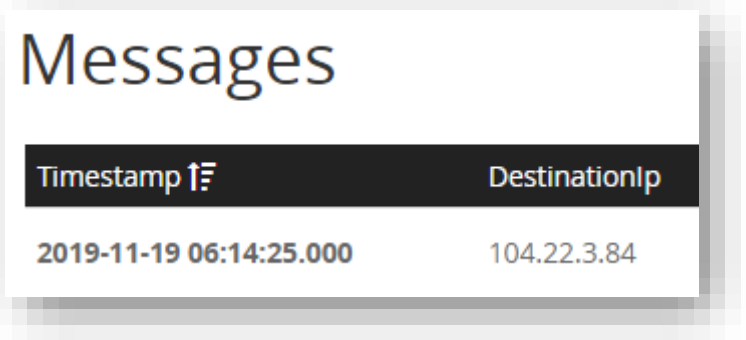


Figure 76 - Graylog Screenshot

Answer: 104.22.3.84

Objectives

Find the Turtle Doves

Synopsis

Find the missing turtle doves.

Solution

In what was the easiest objective, we just had to walk to the student union and on the hearth of the fireplace were the turtle doves

Figure 77 - Two Turtles Doves

Unredact Threatening Document

Synopsis

Someone sent a threatening letter to Elf University. What is the first word in ALL CAPS in the subject line of the letter? Please find the letter in the Quad.

Solution

Like the previous objective, finding the document was fairly easy. It was in the upper left corner of the grid. We've removed the trees for clarity.

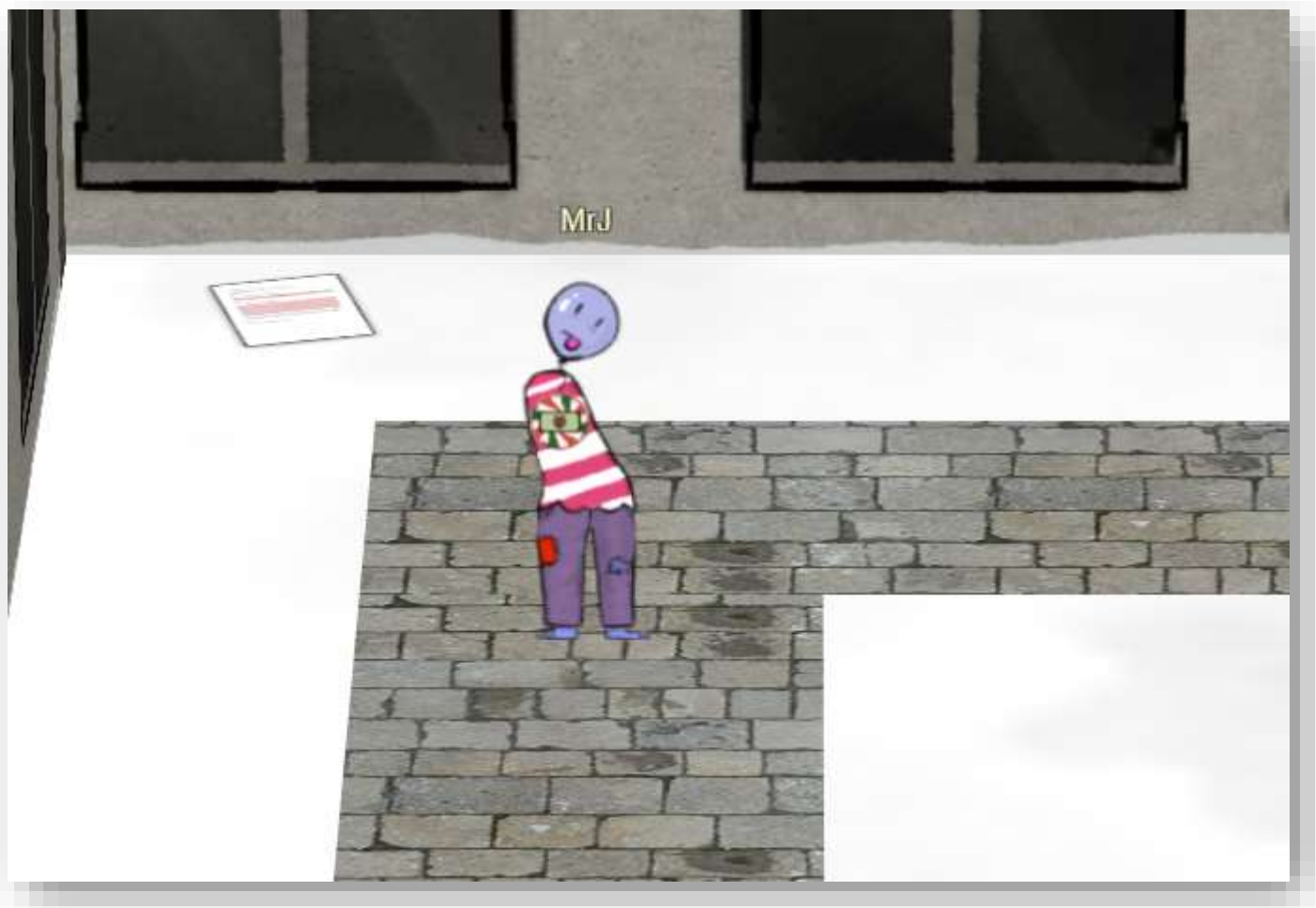


Figure 78 - Unredact Threatening Document

The PDF was a single page letter with most of it being redacted as shown on the next page.

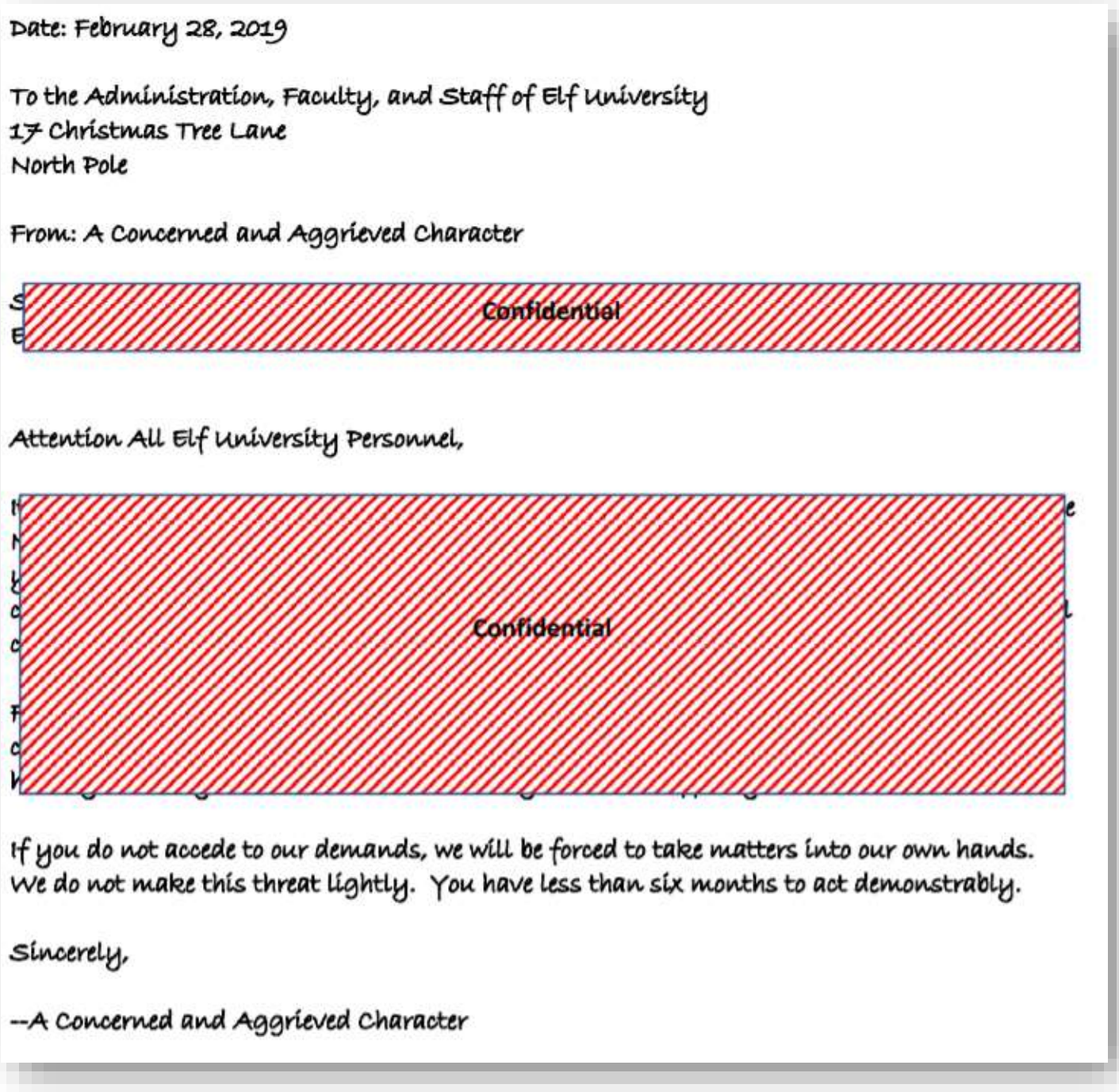
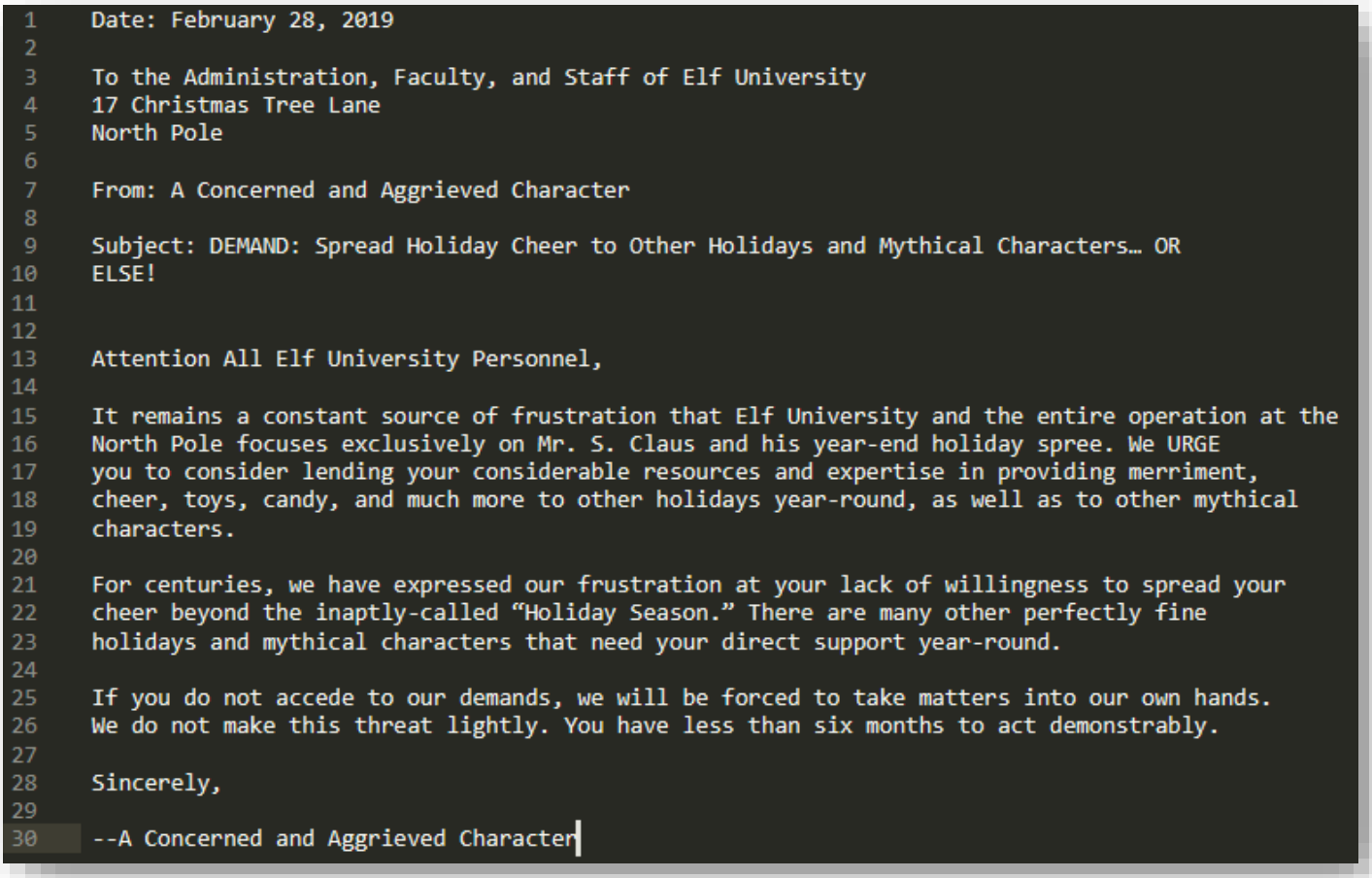


Figure 79 – Redacted Document

There were a few ways to remove the confidential boxes such as opening it in MS Word or using Adobe Acrobat Pro. However, simple selecting all the text, copying it and pasting it into a text editor sufficed.



```
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
```

Figure 80 - Redacted Document

Windows Log Analysis: Evaluate Attack Outcome

Synopsis

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.

- https://downloads.elfu.org/Security.evtx.zip

Hint

- https://www.ericconrad.com/2016/09/deepbluecli-powershell-module-for-hunt.html

Solution

Following on from the hint, we move to https://github.com/sans-blue-team/DeepBlueCLI where we go ahead and download/install DeepBlueCLI.

The usage seems fairly simple so we dive right in. We run it against the contents of the extracted file and export it to an easy-to-read format.

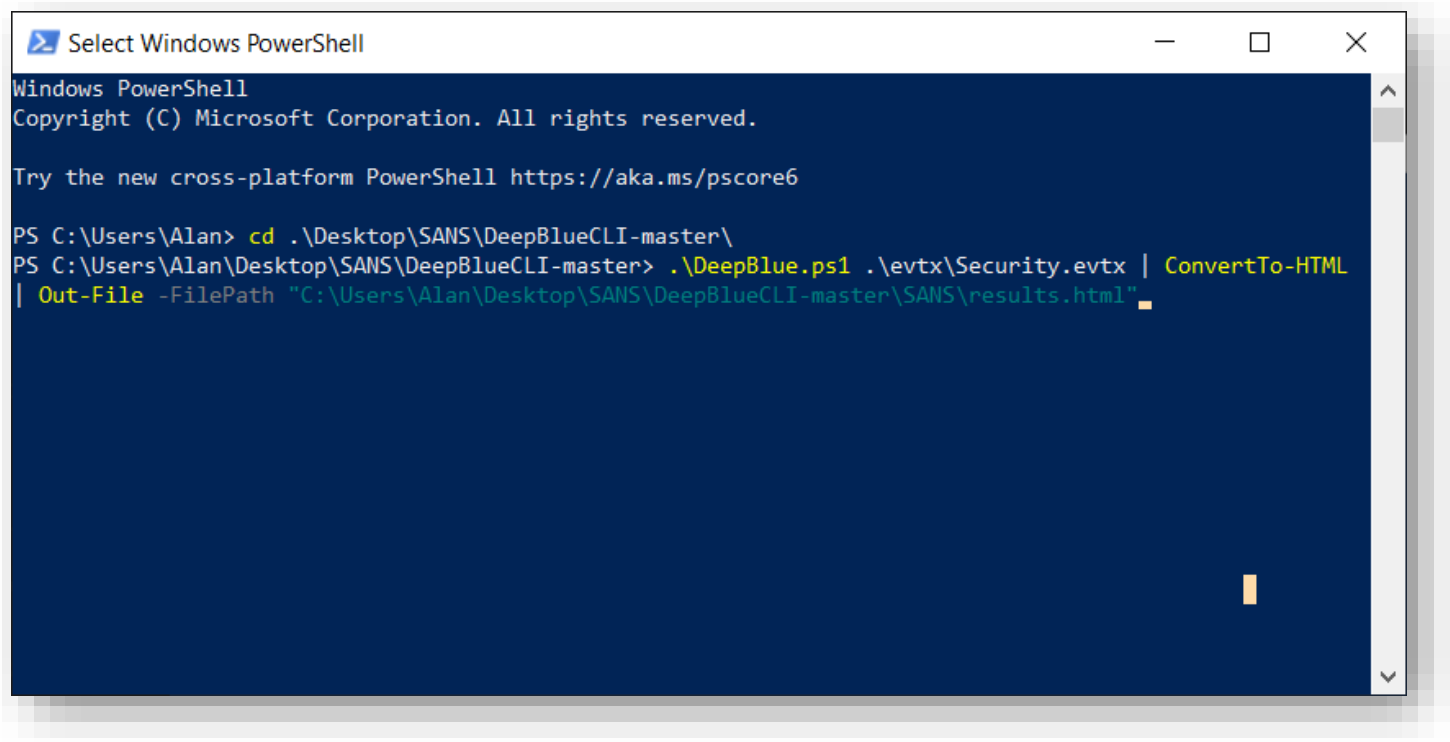
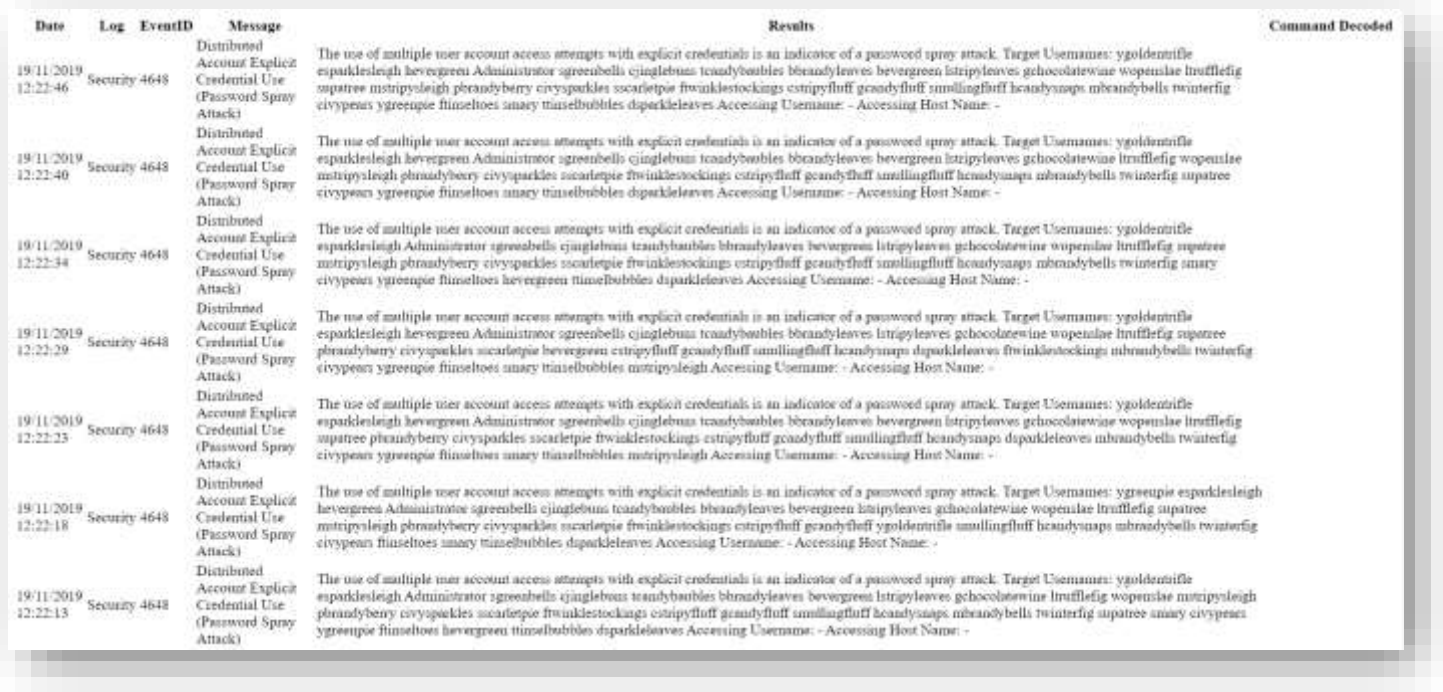


Figure 81 – DeepBlueCLI

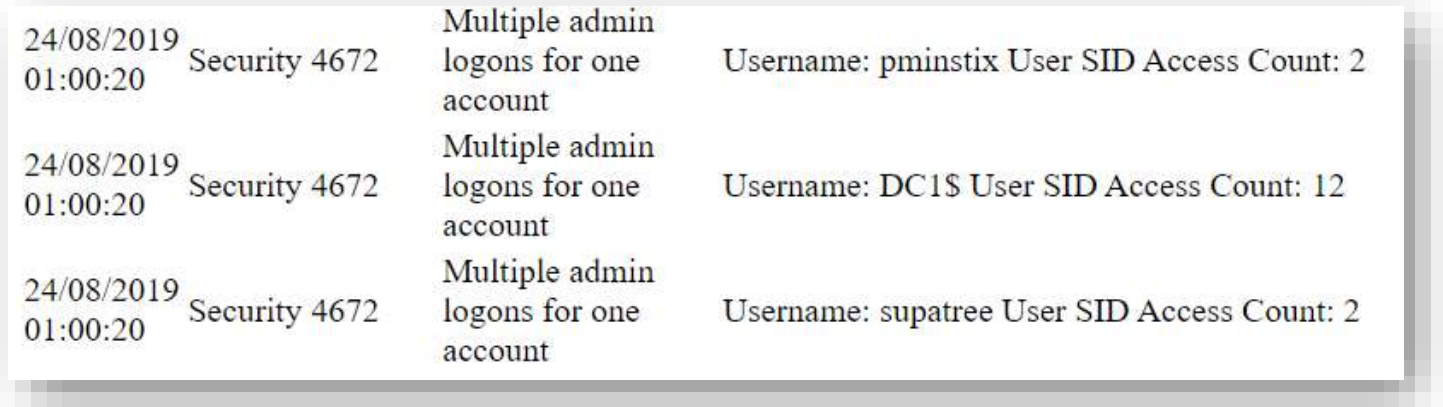
After a minute or so, the results are saved to the HTML file.



Date	Log	EventID	Message	Results	Command Decoded
19/11/2019 12:22:46	Security	4648	Distributed Account Explicit Credential Use (Password Spray Attack)	The use of multiple user account access attempts with explicit credentials is an indicator of a password spray attack. Target Usernames: ygoldentrifle esparkleslegh hevergreen Administrator sgreenbells cjinglebuns tcandybaubles bbrandyleaves bevergreen lstripyleaves gchocolatewine wopenulae ltrufflefig supatree mstripysleigh pbrandyberry civysparkles scarletpie frwinklestockings cstripyfluff gcandyfluff smullingfluff hcandysnaps mbrandybells twinterfig civypears ygreenpie ftinseltoes smary ttinselbubbles dsparkleleaves Accessing Username: - Accessing Host Name: -	
19/11/2019 12:22:40	Security	4648	Distributed Account Explicit Credential Use (Password Spray Attack)	The use of multiple user account access attempts with explicit credentials is an indicator of a password spray attack. Target Usernames: ygoldentrifle esparkleslegh hevergreen Administrator sgreenbells cjinglebuns tcandybaubles bbrandyleaves bevergreen lstripyleaves gchocolatewine ltrufflefig wopenulae mstripysleigh pbrandyberry civysparkles scarletpie frwinklestockings cstripyfluff gcandyfluff smullingfluff hcandysnaps mbrandybells twinterfig supatree civypears ygreenpie ftinseltoes smary ttinselbubbles dsparkleleaves Accessing Username: - Accessing Host Name: -	
19/11/2019 12:22:34	Security	4648	Distributed Account Explicit Credential Use (Password Spray Attack)	The use of multiple user account access attempts with explicit credentials is an indicator of a password spray attack. Target Usernames: ygoldentrifle esparkleslegh Administrator sgreenbells cjinglebuns tcandybaubles bbrandyleaves bevergreen lstripyleaves gchocolatewine wopenulae ltrufflefig supatree mstripysleigh pbrandyberry civysparkles scarletpie frwinklestockings cstripyfluff gcandyfluff smullingfluff hcandysnaps mbrandybells twinterfig smary civypears ygreenpie ftinseltoes hevergreen ttinselbubbles dsparkleleaves Accessing Username: - Accessing Host Name: -	
19/11/2019 12:22:29	Security	4648	Distributed Account Explicit Credential Use (Password Spray Attack)	The use of multiple user account access attempts with explicit credentials is an indicator of a password spray attack. Target Usernames: ygoldentrifle esparkleslegh hevergreen Administrator sgreenbells cjinglebuns tcandybaubles bbrandyleaves lstripyleaves gchocolatewine wopenulae ltrufflefig supatree pbrandyberry civysparkles scarletpie bevergreen cstripyfluff gcandyfluff smullingfluff hcandysnaps dsparkleleaves frwinklestockings mbrandybells twinterfig civypears ygreenpie ftinseltoes smary ttinselbubbles mstripysleigh Accessing Username: - Accessing Host Name: -	
19/11/2019 12:22:23	Security	4648	Distributed Account Explicit Credential Use (Password Spray Attack)	The use of multiple user account access attempts with explicit credentials is an indicator of a password spray attack. Target Usernames: ygoldentrifle esparkleslegh hevergreen Administrator sgreenbells cjinglebuns tcandybaubles bbrandyleaves bevergreen lstripyleaves gchocolatewine wopenulae ltrufflefig supatree pbrandyberry civysparkles scarletpie frwinklestockings cstripyfluff gcandyfluff smullingfluff hcandysnaps dsparkleleaves mbrandybells twinterfig civypears ygreenpie ftinseltoes smary ttinselbubbles mstripysleigh Accessing Username: - Accessing Host Name: -	
19/11/2019 12:22:18	Security	4648	Distributed Account Explicit Credential Use (Password Spray Attack)	The use of multiple user account access attempts with explicit credentials is an indicator of a password spray attack. Target Usernames: ygreenpie esparkleslegh hevergreen Administrator sgreenbells cjinglebuns tcandybaubles bbrandyleaves bevergreen lstripyleaves gchocolatewine wopenulae ltrufflefig supatree mstripysleigh pbrandyberry civysparkles scarletpie frwinklestockings cstripyfluff gcandyfluff ygoldentrifle smullingfluff hcandysnaps mbrandybells twinterfig civypears ftinseltoes smary ttinselbubbles dsparkleleaves Accessing Username: - Accessing Host Name: -	
19/11/2019 12:22:13	Security	4648	Distributed Account Explicit Credential Use (Password Spray Attack)	The use of multiple user account access attempts with explicit credentials is an indicator of a password spray attack. Target Usernames: ygoldentrifle esparkleslegh Administrator sgreenbells cjinglebuns tcandybaubles bbrandyleaves bevergreen lstripyleaves gchocolatewine ltrufflefig wopenulae mstripysleigh pbrandyberry civysparkles scarletpie frwinklestockings cstripyfluff gcandyfluff smullingfluff hcandysnaps mbrandybells twinterfig supatree smary civypears ygreenpie ftinseltoes hevergreen ttinselbubbles dsparkleleaves Accessing Username: - Accessing Host Name: -	

Figure 82 - DeepBlueCLI Results

We can see a whole load of pointers to the aforementioned password spray attack happening. Scrolling down a little, we see some successful logins



24/08/2019 01:00:20	Security 4672	Multiple admin logons for one account	Username: pminstix User SID Access Count: 2
24/08/2019 01:00:20	Security 4672	Multiple admin logons for one account	Username: DC1$ User SID Access Count: 12
24/08/2019 01:00:20	Security 4672	Multiple admin logons for one account	Username: supatree User SID Access Count: 2

Figure 83 - DeepBlueCLI Successful Logins

Out of the three shown, only **supatree** was included in the password spray attacks. As a result, we have found the compromised account.

Answer

supatree

Windows Log Analysis: Determine Attacker Technique

Synopsis

Using these normalized Sysmon logs, identify the tool the attacker used to retrieve domain password hashes from the lsass.exe process. For hints on achieving this objective, please visit Hermey Hall and talk with SugarPlum Mary.

- https://downloads.elfu.org/sysmon-data.json.zip

Hint

- https://www.endgame.com/our-experts/ross-wolf

Solution

The hints points towards using EQL so we went through installing and reading up about it[32] , leaning very heavily on the diagram below.

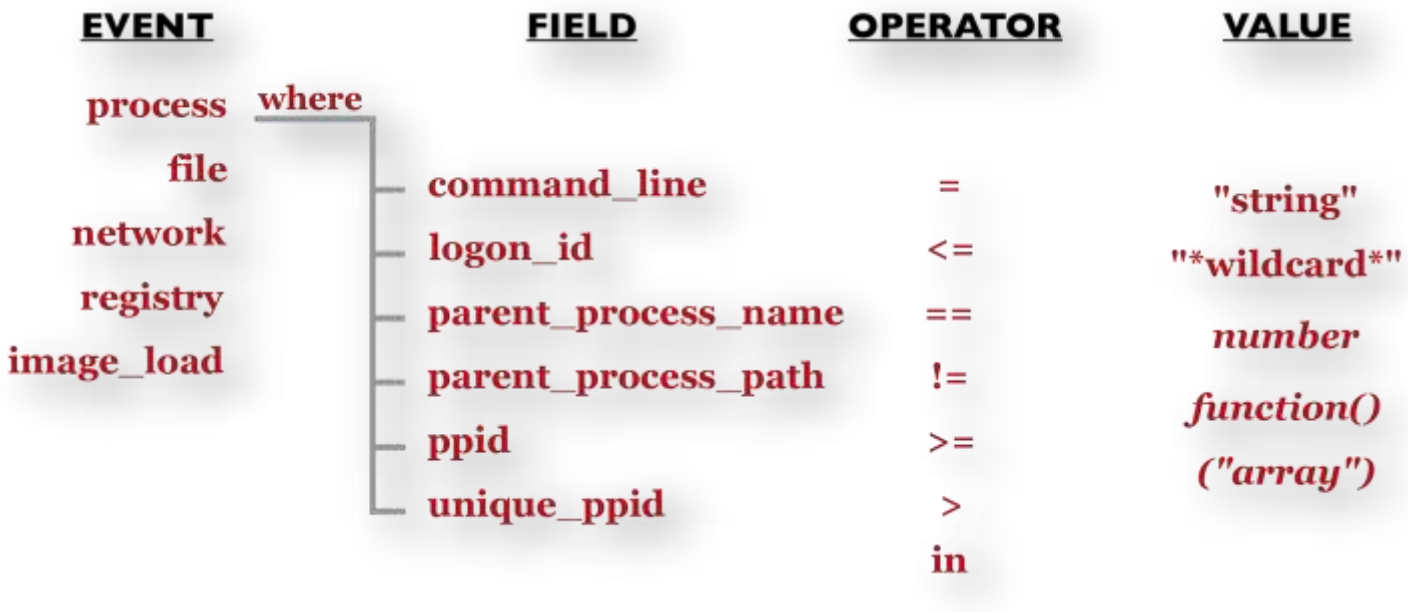


Figure 84 - EQL

Based on the synopsis, we pulled back everything where `lsass.exe` could be present.

[32] https://pen-testing.sans.org/blog/2019/12/10/eql-threat-hunting/

Figure 85 – EQL

There's only one instance so we look at everything where the parent process ID is equal to the process ID of what we just found

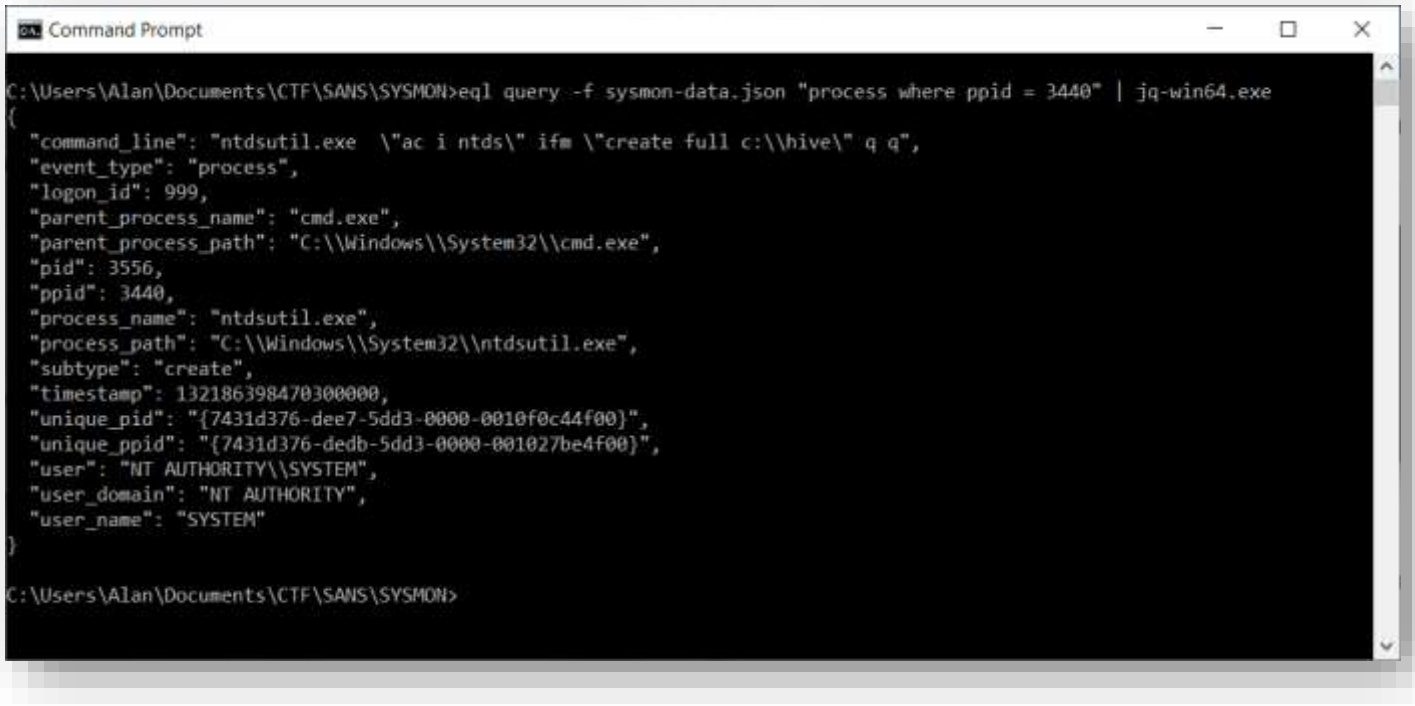


Figure 86 - EQL

Alternatively, we could have looked at what happened after lsass.exe was fired up which would have given us the same answer.

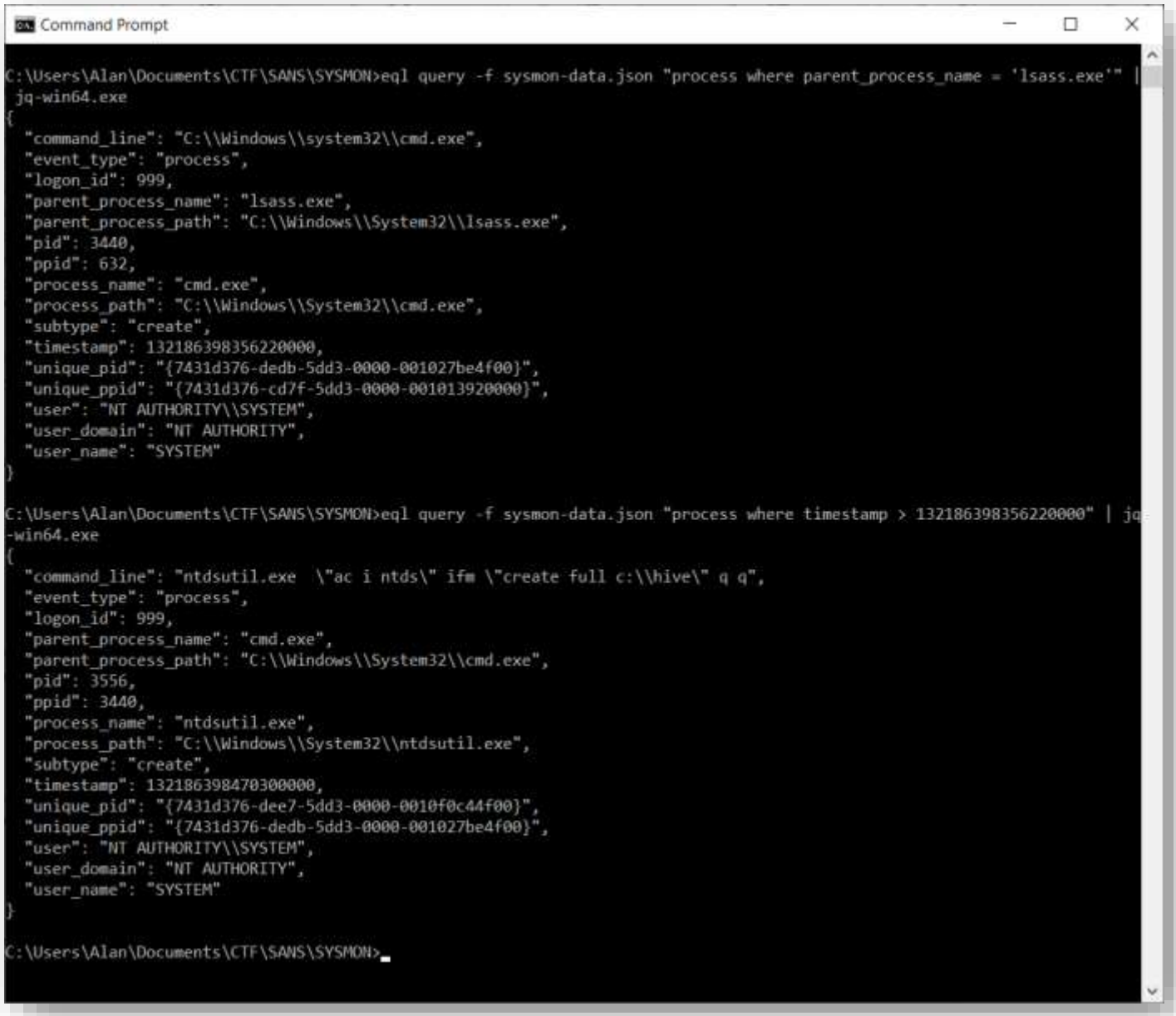


Figure 87 – EQL

Answer

ntdsutil.exe

Network Log Analysis: Determine Compromised System

Synopsis

The attacks don't stop! Can you help identify the IP address of the malware-infected system using these Zeek logs? For hints on achieving this objective, please visit the Laboratory and talk with Sparkle Redberry.

- https://downloads.elfu.org/elfu-zeeklogs.zip

Hint

https://www.activecountermeasures.com/free-tools/rita/

Solution

As it turns out, there were two ways to do this. First way was to install RITA and process the logs as below.

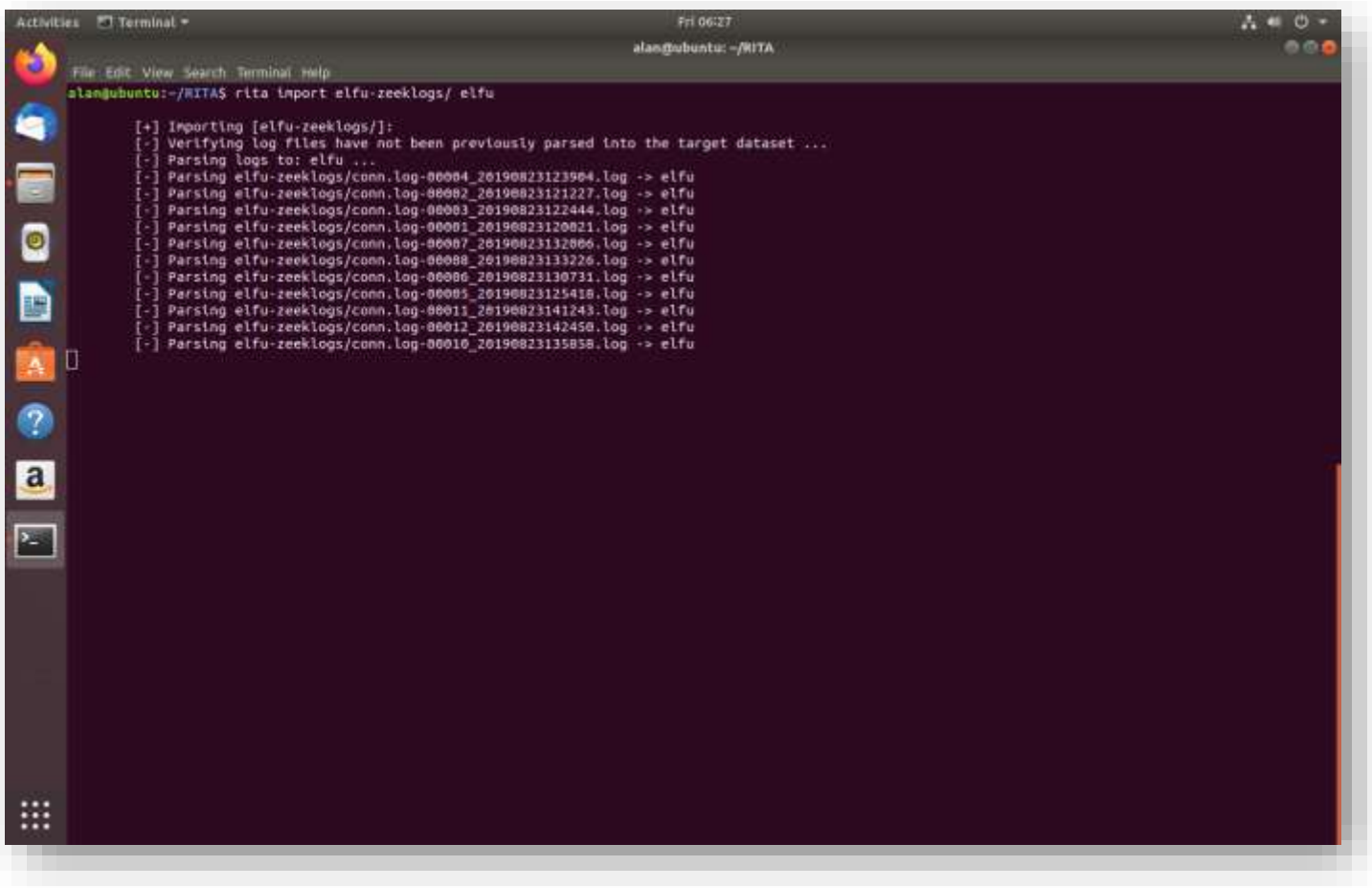


Figure 88 - RITA Screenshot

Running the show-beacons command displays hosts which show signs of a C2 channel[33].

[33] https://digital-forensics.sans.org/blog/2014/03/31/the-importance-of-command-and-control-analysis-for-incident-response

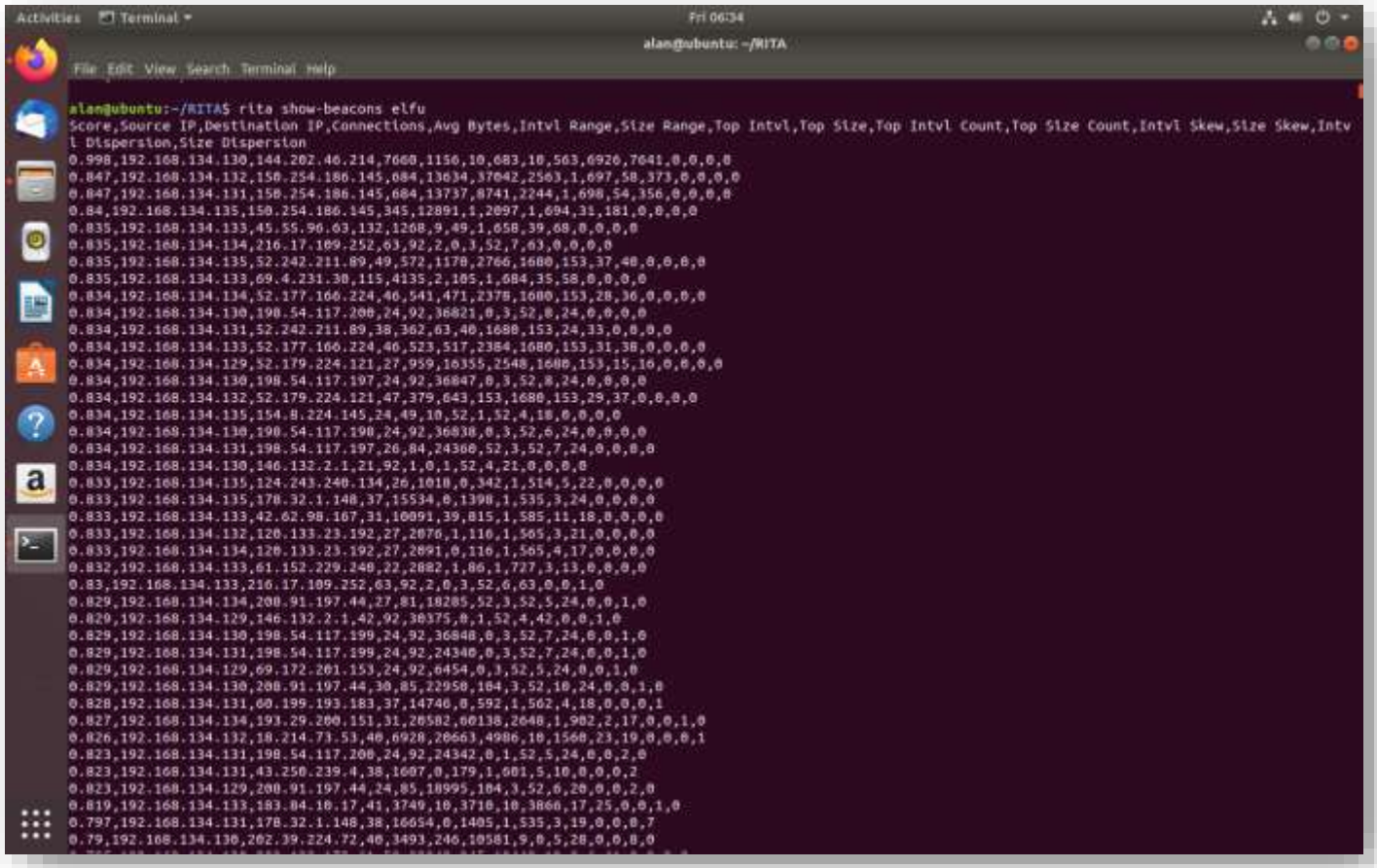
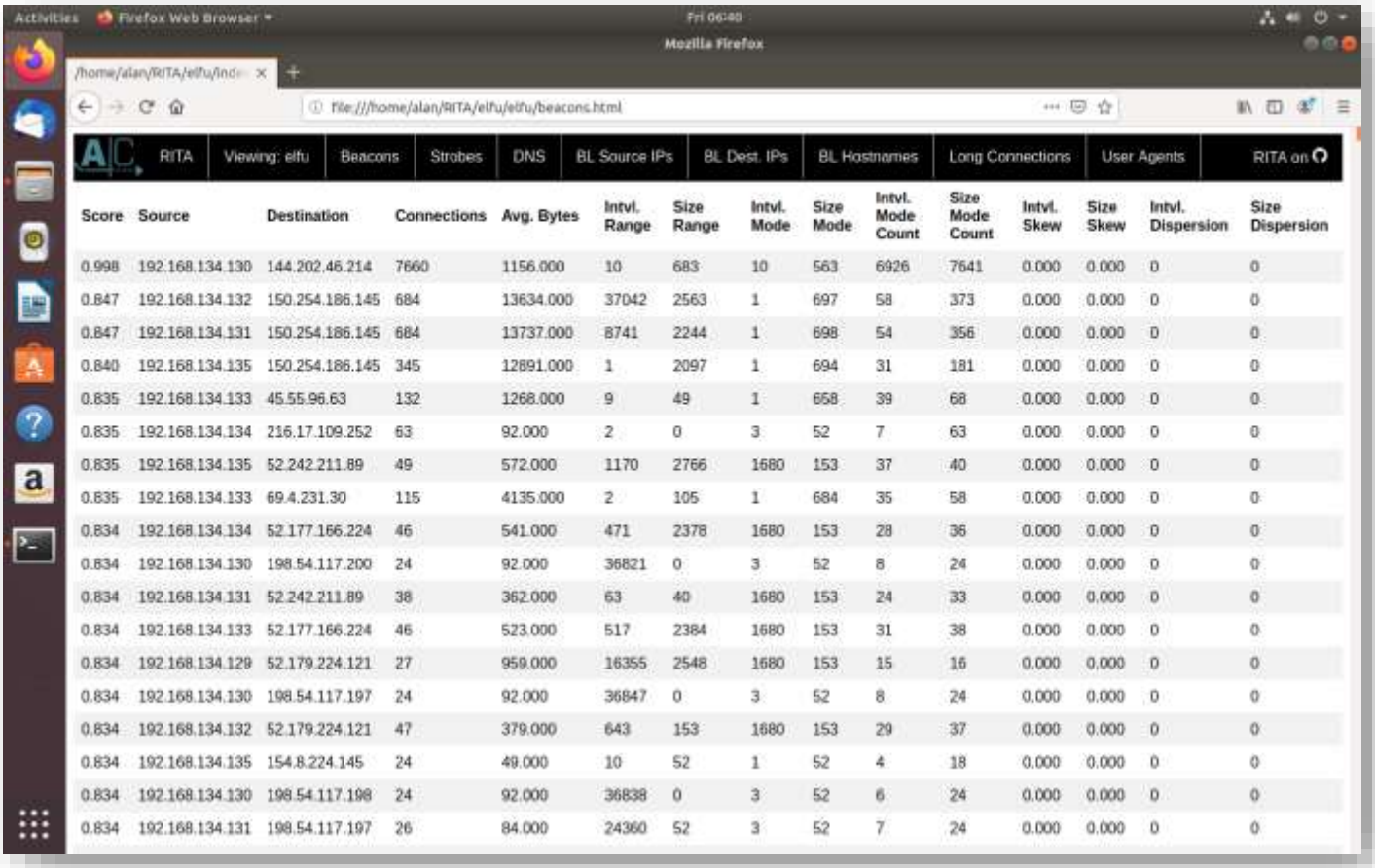


Figure 89 - RITA Screenshot

Whilst readable, we can actually output the results to HTML for easier consumption.

```
alan@ubuntu:~/RITA$ rita html-report elfu
[-] Writing: /home/alan/RITA/elfu/elfu
[-] Wrote outputs, check /home/alan/RITA/elfu for files
alan@ubuntu:~/RITA$
```

Figure 90 - RITA Screenshot



Score	Source	Destination	Connections	Avg. Bytes	Intvl. Range	Size Range	Intvl. Mode	Size Mode	Intvl. Mode Count	Size Mode Count	Intvl. Skew	Size Skew	Intvl. Dispersion	Size Dispersion
0.998	192.168.134.130	144.202.46.214	7660	1156.000	10	683	10	563	6926	7641	0.000	0.000	0	0
0.847	192.168.134.132	150.254.186.145	684	13634.000	37042	2563	1	697	58	373	0.000	0.000	0	0
0.847	192.168.134.131	150.254.186.145	684	13737.000	8741	2244	1	698	54	356	0.000	0.000	0	0
0.840	192.168.134.135	150.254.186.145	345	12891.000	1	2097	1	694	31	181	0.000	0.000	0	0
0.835	192.168.134.133	45.55.96.63	132	1268.000	9	49	1	658	39	68	0.000	0.000	0	0
0.835	192.168.134.134	216.17.109.252	63	92.000	2	0	3	52	7	63	0.000	0.000	0	0
0.835	192.168.134.135	52.242.211.89	49	572.000	1170	2766	1680	153	37	40	0.000	0.000	0	0
0.835	192.168.134.133	69.4.231.30	115	4135.000	2	105	1	684	35	58	0.000	0.000	0	0
0.834	192.168.134.134	52.177.166.224	46	541.000	471	2378	1680	153	28	36	0.000	0.000	0	0
0.834	192.168.134.130	198.54.117.200	24	92.000	36821	0	3	52	8	24	0.000	0.000	0	0
0.834	192.168.134.131	52.242.211.89	38	362.000	63	40	1680	153	24	33	0.000	0.000	0	0
0.834	192.168.134.133	52.177.166.224	46	523.000	517	2384	1680	153	31	38	0.000	0.000	0	0
0.834	192.168.134.129	52.179.224.121	27	959.000	16355	2548	1680	153	15	16	0.000	0.000	0	0
0.834	192.168.134.130	198.54.117.197	24	92.000	36847	0	3	52	8	24	0.000	0.000	0	0
0.834	192.168.134.132	52.179.224.121	47	379.000	643	153	1680	153	29	37	0.000	0.000	0	0
0.834	192.168.134.135	154.8.224.145	24	49.000	10	52	1	52	4	18	0.000	0.000	0	0
0.834	192.168.134.130	198.54.117.198	24	92.000	36838	0	3	52	6	24	0.000	0.000	0	0
0.834	192.168.134.131	198.54.117.197	26	84.000	24360	52	3	52	7	24	0.000	0.000	0	0

Figure 91 - RITA Screenshot

We can see that the source with the greatest number of connections with one of the lowest interval ranges, and by some amount, is listed at the top which we can safely take as our malware-infected system

The alternative solution to this was to look in the folder within the extracted downloaded ZIP file as this already contained the HTML file that was produced over the last few screenshots.

Answer

192.168.134.130

Splunk

Synopsis

Access https://splunk.elfu.org/ as elf with password elfsocks. What was the message for Kent that the adversary embedded in this attack? The SOC folks at that link will help you along! For hints on achieving this objective, please visit the Laboratory in Hermey Hall and talk with Prof. Banas.

- https://splunk.elfu.org/

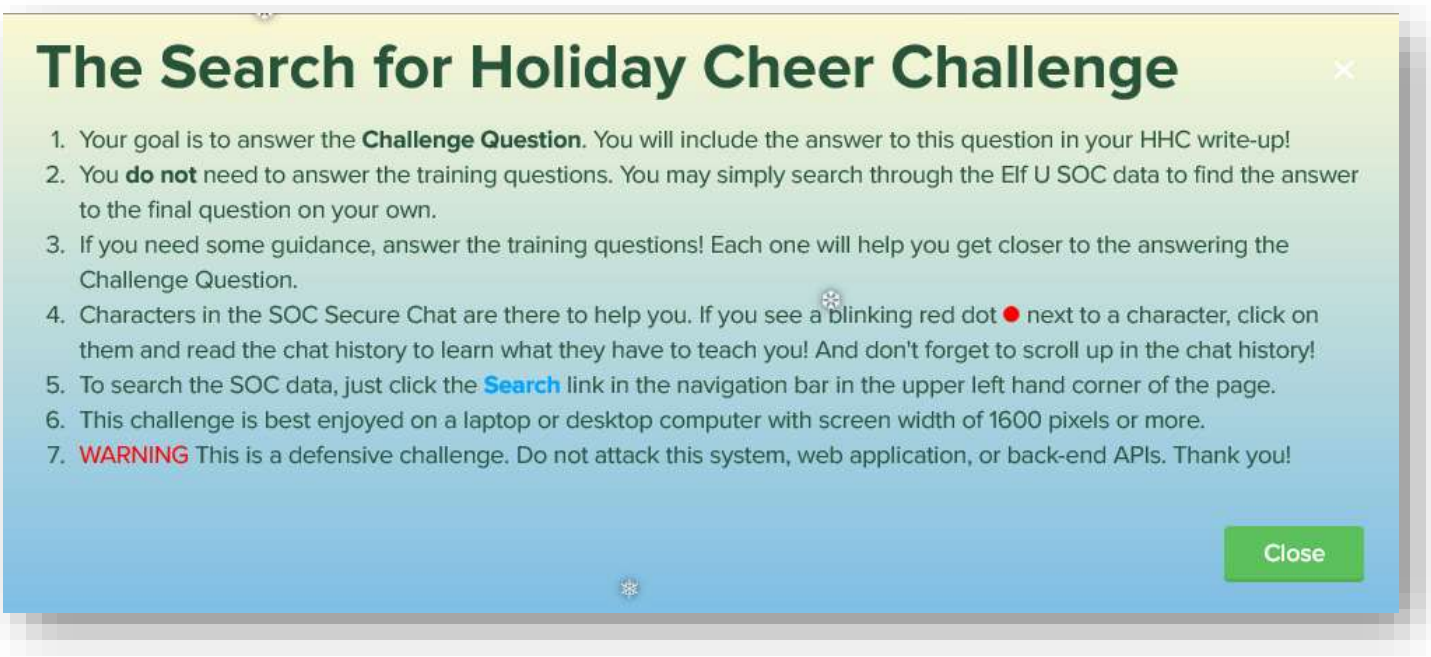


Figure 92 - Splunk Screenshot

Solution

When we visit the site and login, we get presented with the following.

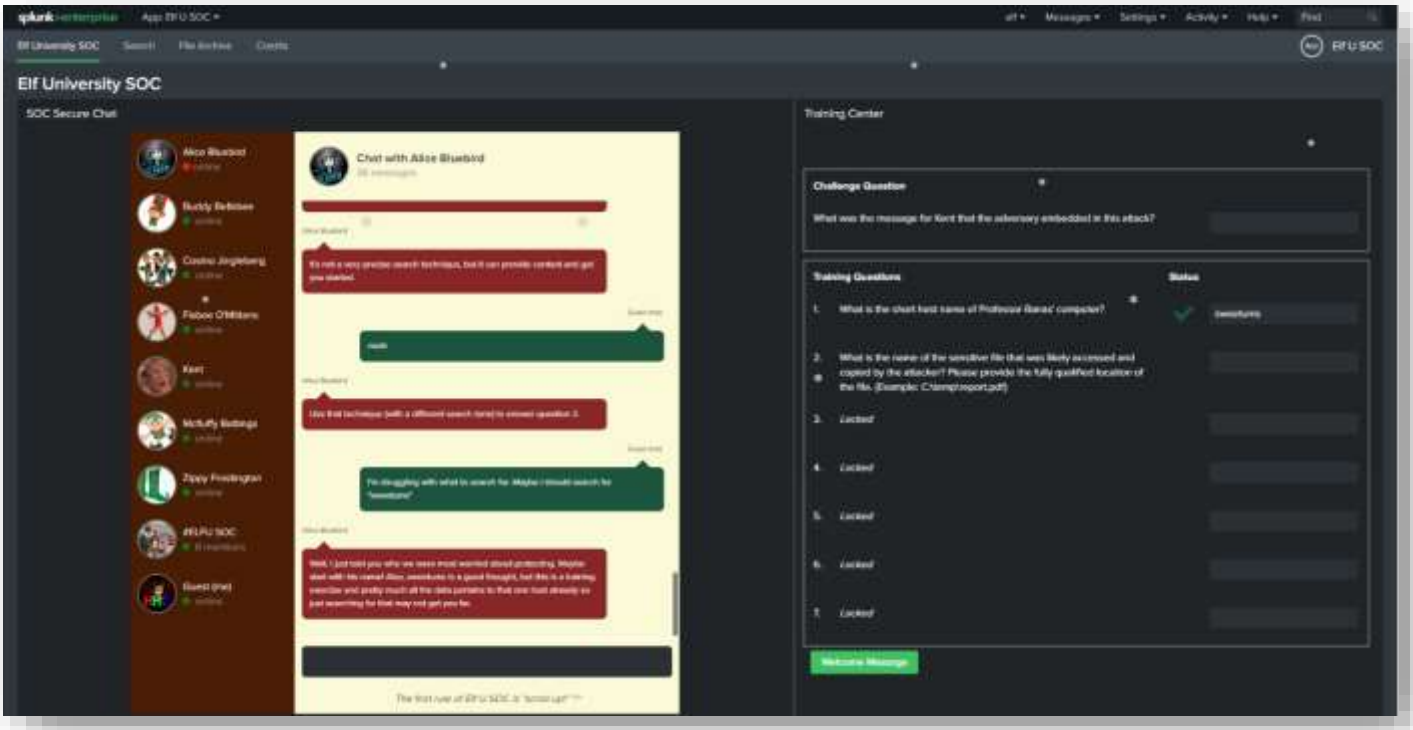


Figure 93 - Splunk Screenshot

There are 7 training questions to answer before we go for the challenge question itself. For brevity, only the relevant screenshots will be shown here. Everything else will be in the appendix.

Question 1

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

When we first go in, there is a chat window with what seems like an active chat where we have our first pointer.

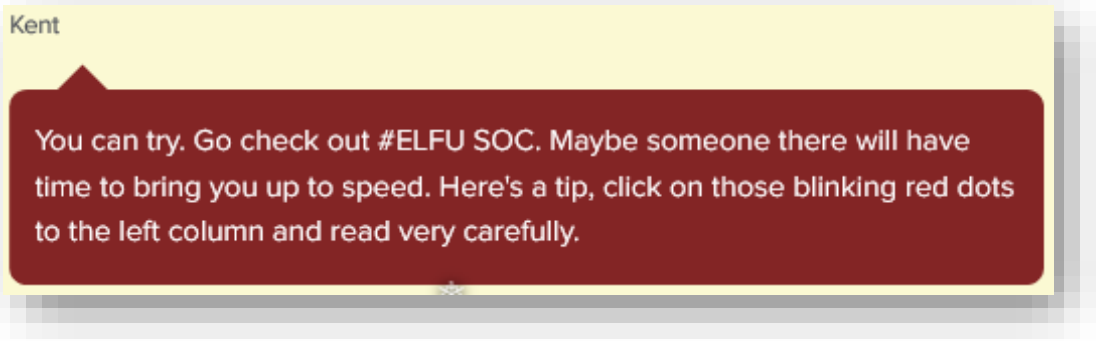


Figure 94 - Splunk Screenshot

Into the #ELFU SOC chat we go where we find the answer to our first question.

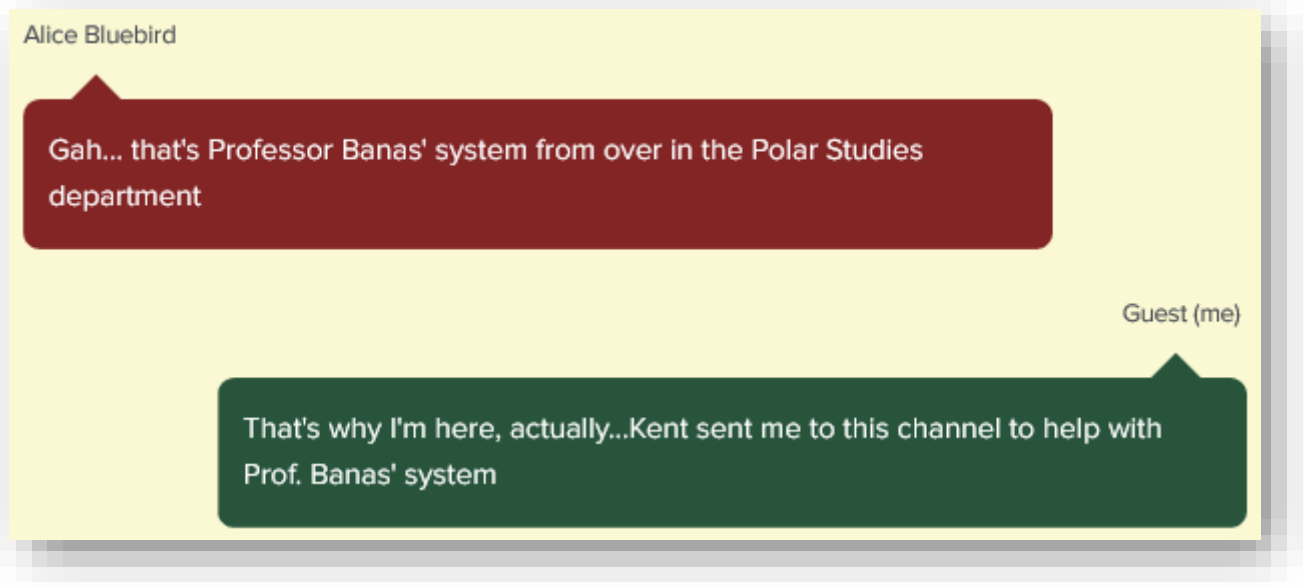


Figure 95 - Splunk Screenshot

Interestingly we see what looks like Empire[34][35] payloads going to an IP of http://144.202.46.214:8080 which again point to https://www.vultr.com/.

Figure 96 - Splunk Screenshot

[34] https://medium.com/@netscylla/powershell-that-looks-smells-like-empire-payloads-7f9bfdd39e5b
[35] http://www.powershellempire.com/

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)

Upon answering the first question correctly, the chat with Alice Bluebird continues and expands. The hint Alice gives is that a simple search will suffice and that Santa may be the keyword in question

A simple search for "Santa" reveals the document we are after

Figure 97 - Splunk Screenshot

Question 3

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

Several pieces of dialog help narrow down the search needed for this next one and the search string pretty much gives us exactly what we need.

The search string is `index=main sourcetype=XmlWinEventLog:Microsoft-Windows-Sysmon/Operational powershell EventCode=3`

This yields over 150 results but as the chat said, some interesting fields to the left which we go into that give us our next answer

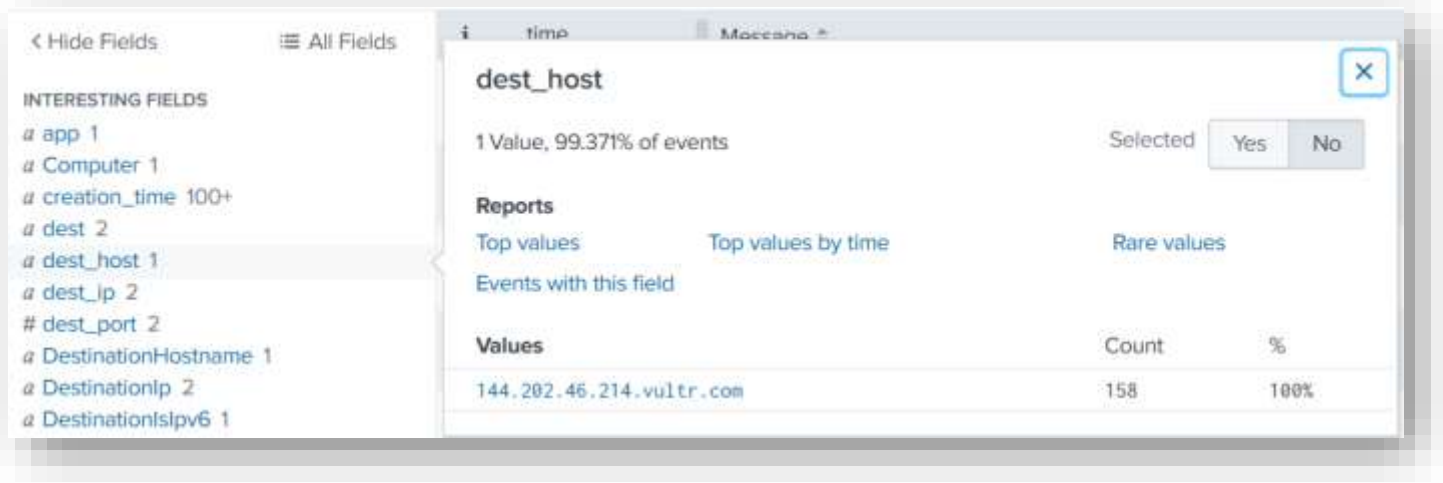


Figure 98 - Splunk Screenshot

Question 4

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

The search Alice gives us is `index=main sourcetype="WinEventLog:Microsoft-Windows-Powershell/Operational"` to look for all PowerShell logs on the system. We then need to determine the process ID or process GUID associated with these logs[36].

We pipe the search to `reverse` and apply a timeframe of 5 seconds either side to the first record to see what process IDs are present. These end up being 6268 and 5864.

Those IDs are turned into hex and added to a search string one at a time, looking for the event code `4688`[37] as follows: `index=main sourcetype=WinEventLog EventCode=4688 process_id=0x187c`

This only brings back one event and we are able to provide the details once again.

Figure 99 - Splunk Screenshot

Question 5

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

---

[36] The details of which will be documented in the appendix

[37] https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4688

We get given the following search which analyses email addresses via stoQ[38]: `index=main sourcetype=stoq | table _time results{}.workers.smtp.to results{}.workers.smtp.from results{}.workers.smtp.subject results{}.workers.smtp.body | sort - _time`

Unfortunately, this doesn't provide unique email address so we amend the script slightly to account for it and exclude Professor Banas sending to himself. This leaves us with the following search that gives us our answer: `index=main sourcetype=stoq results{}.workers.smtp.from!="carl banas <carl.banas@faculty.elfu.org>" | table _time results{}.workers.smtp.from | stats count by results{}.workers.smtp.from | rename results{}.workers.smtp.from as email_addy | eval email_addy_count=lower(email_addy) | stats count by email_addy_count`

Figure 100 - Splunk Screenshot

Question 6

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

A search of just **password** gave us the password we were after but also using the search from before (`index=main sourcetype=stoq | table _time results{}.workers.smtp.to results{}.workers.smtp.from results{}.workers.smtp.subject results{}.workers.smtp.body | sort - _time`) gave us it in a nicer view.

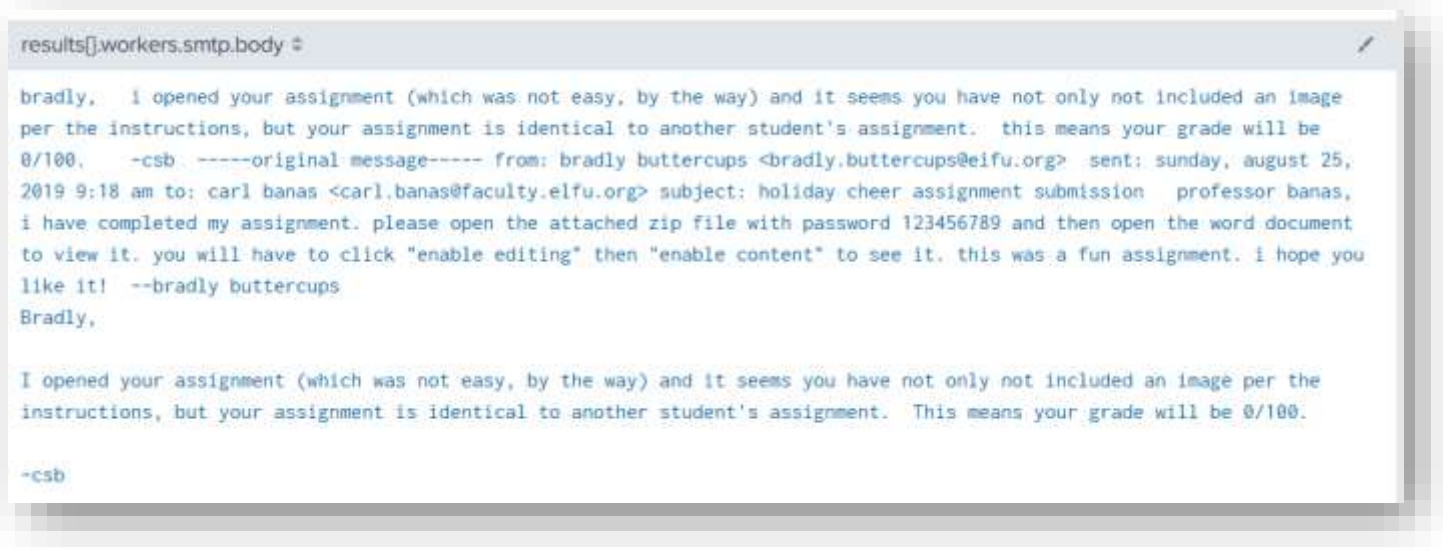


Figure 101 - Splunk Screenshot

Question 7

What email address did the suspicious file come from?

Again, the previous search was enough to answer this question as well.

---

[38] https://stoq.punchcyber.com/

Figure 102 - Splunk Screenshot

Challenge Question

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

Lastly, the two searches provided by Alice combine to make the following: `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`

Furthermore, Alice mentions a line word documents, specifically the word **core**. This points us at one specific record.

Figure 103 - Splunk Screenshot

Navigating to http://elfu-soc.s3-website-us-east-1.amazonaws.com/?prefix=stoQ%20Artifacts/home/ubuntu/archive/f/f/1/e/a/ presents us with a file which we download.

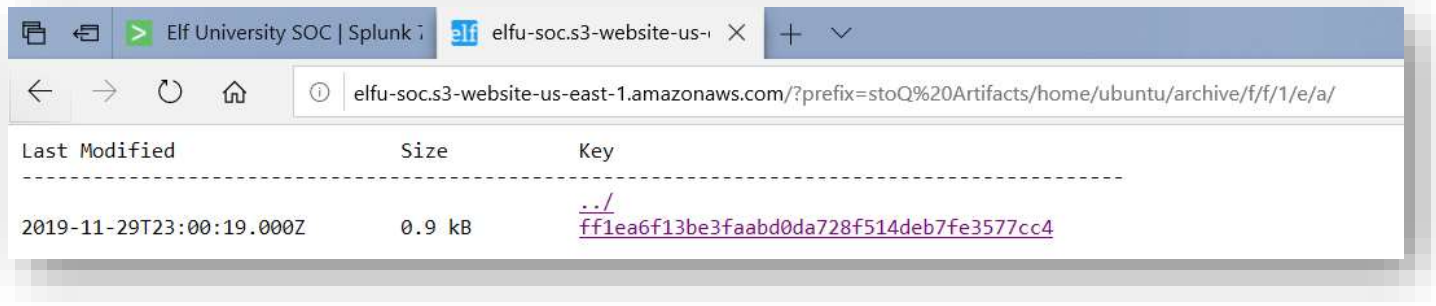


Figure 104 - Splunk Screenshot

The downloaded file is XML and has the details we are after.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cp:coreProperties
    xmlns:cp="http://schemas.openxmlformats.org/package/2006/metadata/core-properties"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:dcterms="http://purl.org/dc/terms/"
    xmlns:dcmitype="http://purl.org/dc/dcmitype/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <dc:title>Holiday Cheer Assignment</dc:title>
    <dc:subject>19th Century Cheer</dc:subject>
    <dc:creator>Bradly Buttercups</dc:creator>
    <cp:keywords></cp:keywords>
    <dc:description>Kent you are so unfair. And we were going to make you the king of the Winter Carnival.</dc:description>
    <cp:lastModifiedBy>Tim Edwards</cp:lastModifiedBy>
    <cp:revision>4</cp:revision>
    <dcterms:created xsi:type="dcterms:W3CDTF">2019-11-19T14:54:00Z</dcterms:created>
    <dcterms:modified xsi:type="dcterms:W3CDTF">2019-11-19T17:50:00Z</dcterms:modified>
    <cp:category></cp:category>
</cp:coreProperties>
```

Figure 105 - Splunk Screenshot

Finally, we have all the answers in one place

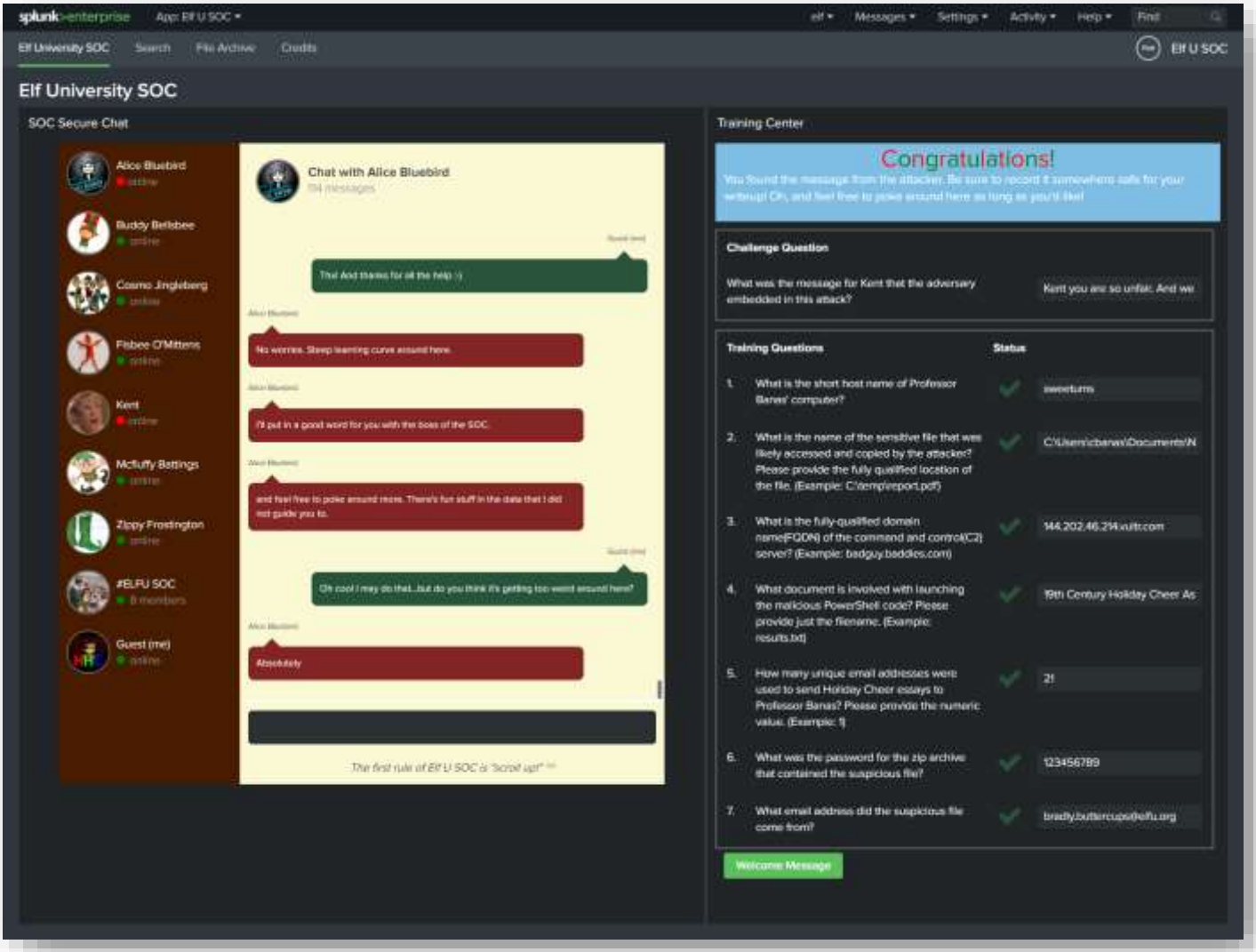


Figure 106 - Splunk Screenshot

Get Access To The Steam Tunnels

Synopsis

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.

Hint

Deviant Ollam, Optical Decoding of Keys - https://youtu.be/KU6FJnbkeLA

Solution

This solution starts off in Minty's dorm room.

Figure 107 - Steam Tunnels Screenshot

When we first go into the room, we see someone go into the back room (who we later find out to be Krampus). Following them to the back room reveals they have disappeared.

Figure 108 - Steam Tunnels Screenshot

Both rooms open up a terminal of sorts:

- Front room: https://key.elfu.org/?challenge=bitting-cutter
- Back room: https://thisisit.elfu.org/?challenge=bitting-keyhole

From these two terminals, it was clear we had to cut a key in the first one to unlock the keyhole in the second one.

The main question was how would we know what key to cut?

Figure 109 - Steam Tunnels Screenshot

The bitting machine would have given us 1,000,000 combinations so brute forcing would have been an option but with each file produced from the machine (having pressed on **Cut**) being 143Kb, downloading 143Gb worth of data didn't seem reasonable.

However, something appeared out of the ordinary. Not only did the avatar that moved between the rooms had a key attached to them where no other avatar did, when inspecting their size, they were about 8 times as large as all the other avatars as we can see in the following image.

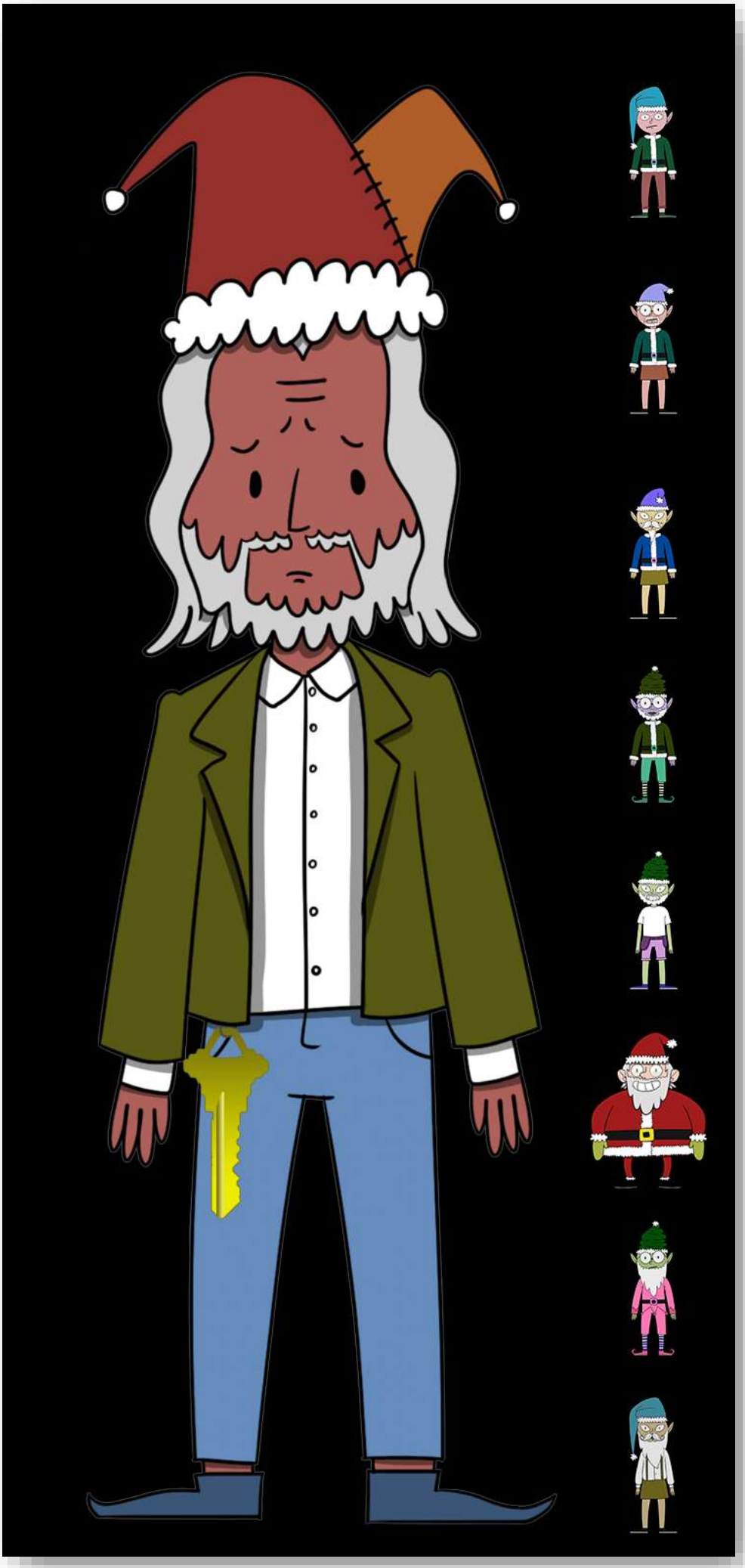


Figure 110 - Steam Tunnels Screenshot

So far, we had the following clues:

- The avatar went into the room and seemingly vanished
- The avatar is around 8 times the size of a normal avatar
- They key seems to be drawn differently to the rest of the avatar

Zooming in on the key gave us a reasonable outline.

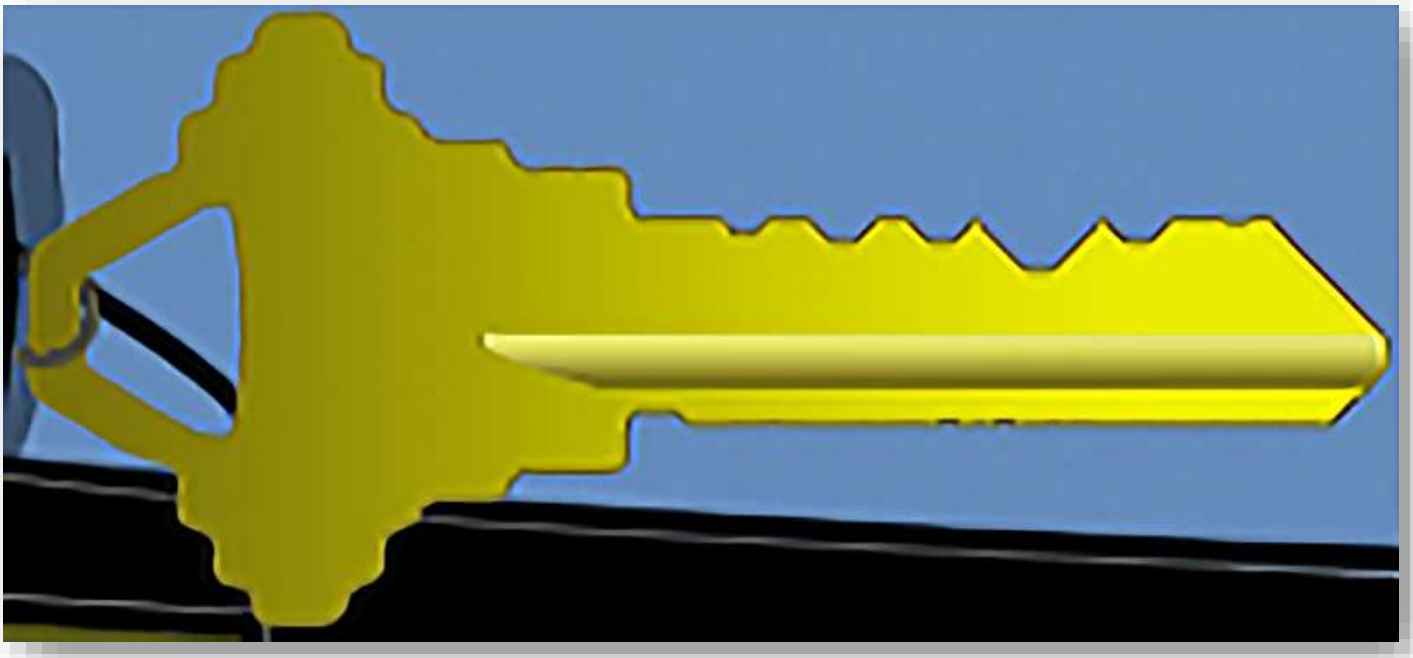


Figure 111 - Steam Tunnels Screenshot

We needed to produce a key using the bitting machine what would match the pattern of the key that the avatar was wearing. Using the machine, we saw we could just get the images via a URL e.g. https://key.elfu.org/backend/keys/SC4_preview/000000.png

After producing a few examples, there is a pattern emerging where the grooves are relevant to whatever number gets passed in the URL.

SC4 was interesting as it pointed to it being 6 pin[39], hence the 6 numbers we could use.

[39] https://kc.allegion.com/kb/article/what-is-an-sc-keyway/

Figure 112 - Steam Tunnels Screenshot

This is where the talk within KringleCon came in hugely useful: https://www.youtube.com/watch?v=KU6FJnbkeLA as well as another related one: https://www.youtube.com/watch?v=AayXf5aRFTI. It pointed us nicely to the type of key being used which gave us a guide on how to decode the zoomed in version we have.

Figure 113 - Steam Tunnel Screenshot

Once identified, we could start using the guide below to decode our key

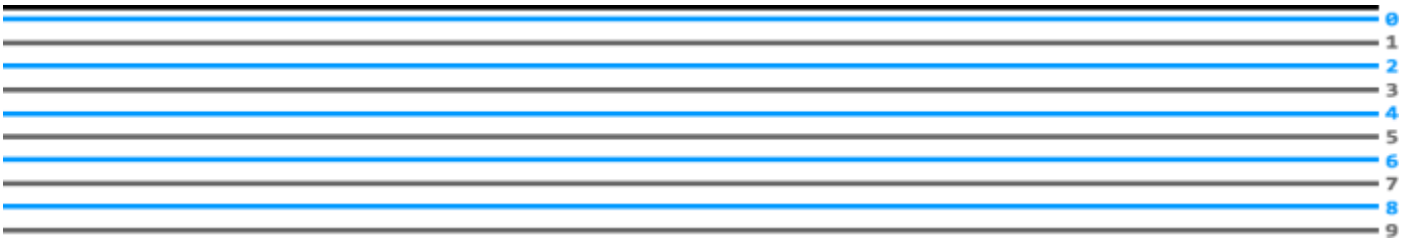


Figure 114 - Steam Tunnels Screenshot

After imposing our key onto the guide and adjusting as needed, we had our match.

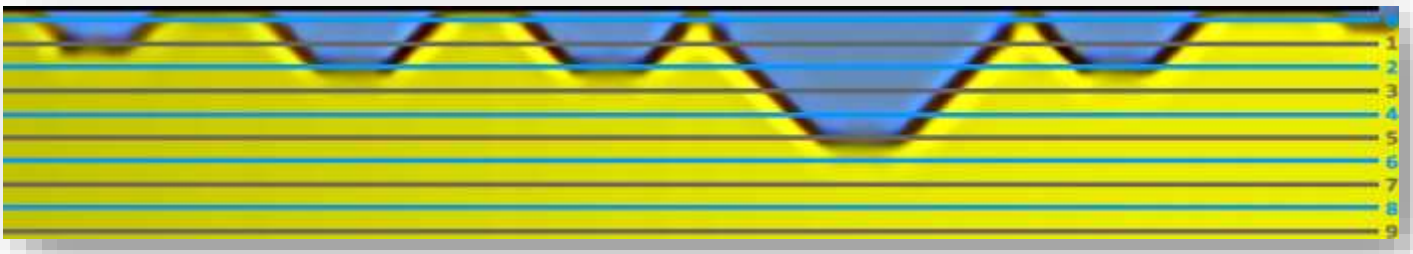


Figure 115 - Steam Tunnels Screenshot

The reference gave us a number of **122520**. Putting this into https://key.elfu.org/backend/keys/SC4_preview/122520.png gave us the following key:

Figure 116 - Steam Tunnels Screenshot

Now, we had the key to access https://thisisit.elfu.org/?challenge=bitting-keyhole.

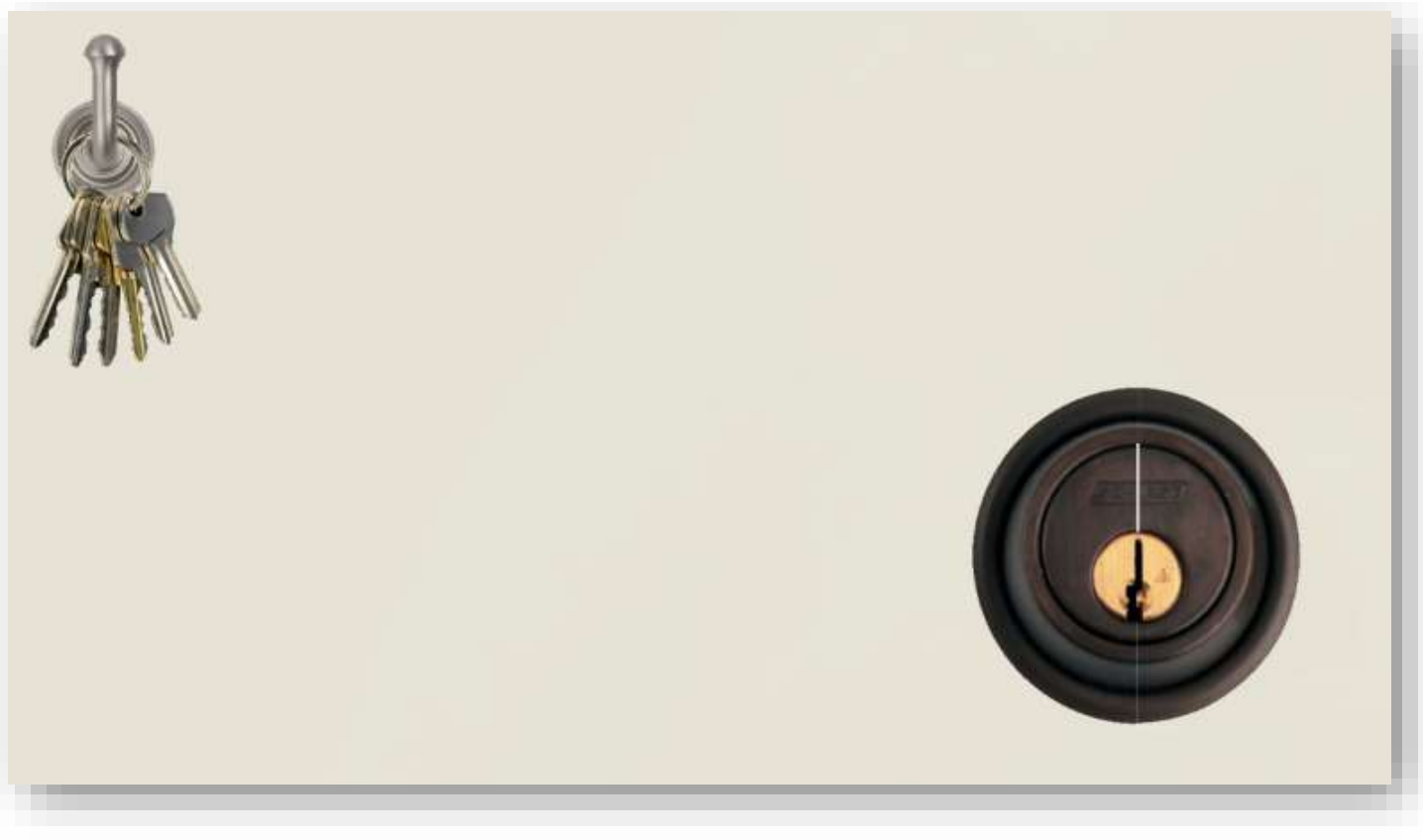


Figure 117 - Steam Tunnels Screenshot

Clicking on the keys allowed us to select the key we generated and dragging and dropping it onto the lock, unlocked the door and keyhole, helping us complete the objective.

It led us to Krampus Hollyfeld who told us it was him who took the two turtle doves.

Hello there! I'm Krampus Hollyfeld.

I maintain the steam tunnels underneath Elf U,

Keeping all the elves warm and jolly.

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.

Answer

Krampus Hollyfeld

Bypassing the Frido Sleigh CAPTEHA

Synopsis

Help Krampus beat the Frido Sleigh contest. For hints on achieving this objective, please talk with Alabaster Snowball in the Speaker Unpreparedness Room.

- https://fridosleigh.com/

Hint

Machine Learning Use Cases for Cyber Security - https://youtu.be/jmVPLwjm_zs

Solution

INSERT PIECE ABOUT TRYING TO SOLVE IT A DIFFERENT WAY

Visiting the site presents us with the screenshot shown further on. Filling it out eventually brings us down to the CAPTEHA.

Figure 118 - Capteha Screenshot

More details about it can be found by clicking on the information link[40]. The capteha is created from a call to https://fridosleigh.com/api/capteha/request which pulls in 100 random images.

[40] https://fridosleigh.com/about_CAPTEHA.html

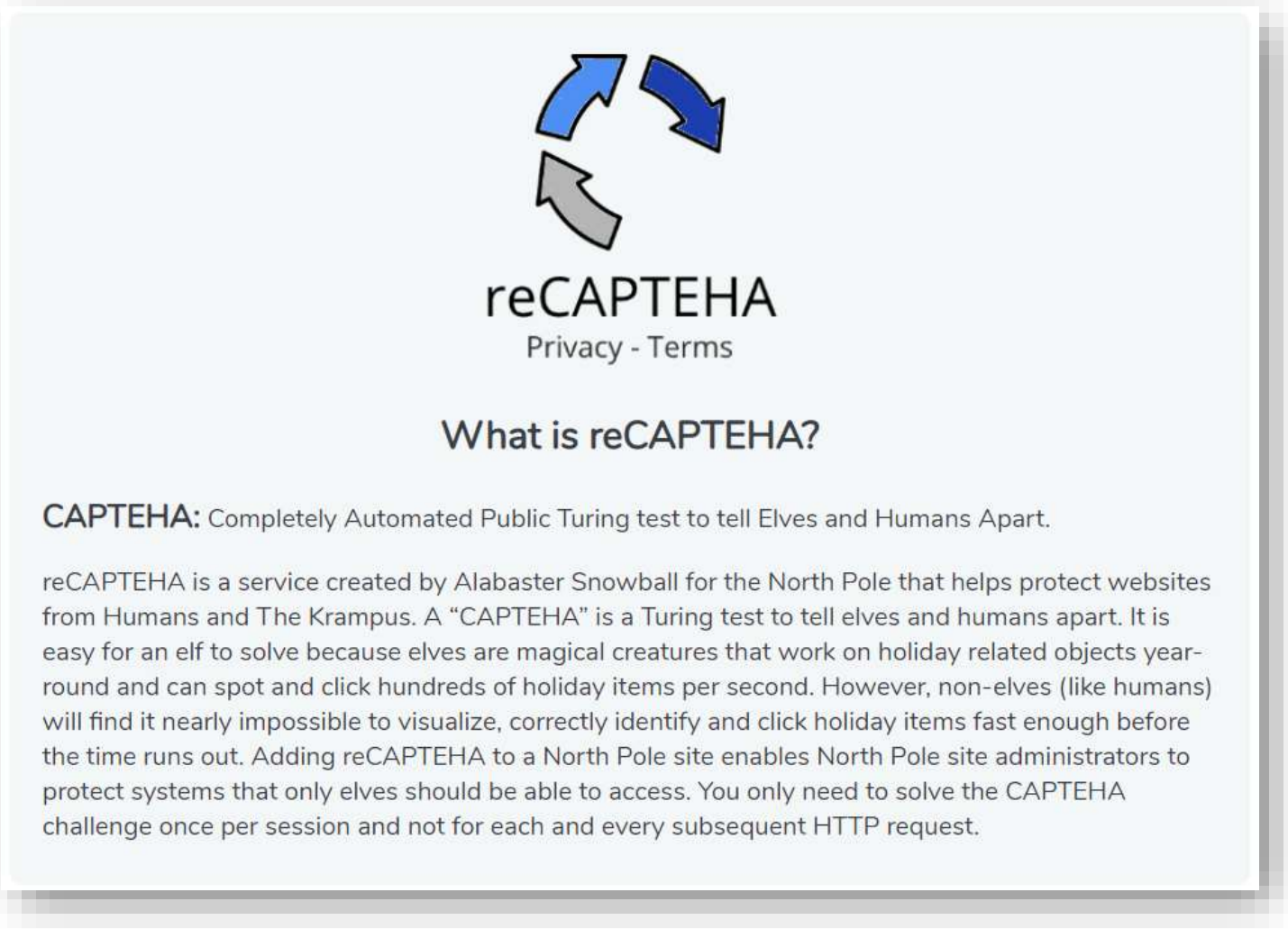


Figure 119 - Capteha Screenshot

The important line is at the end.

> *You only need to solve the CAPTEHA challenge once per session and not for each and every subsequent HTTP request.*

A few links are given to help us along our way:

- https://downloads.elfu.org/capteha_images.tar.gz
- https://downloads.elfu.org/capteha_api.py
- https://github.com/chrisjd20/img_rec_tf_ml_demo

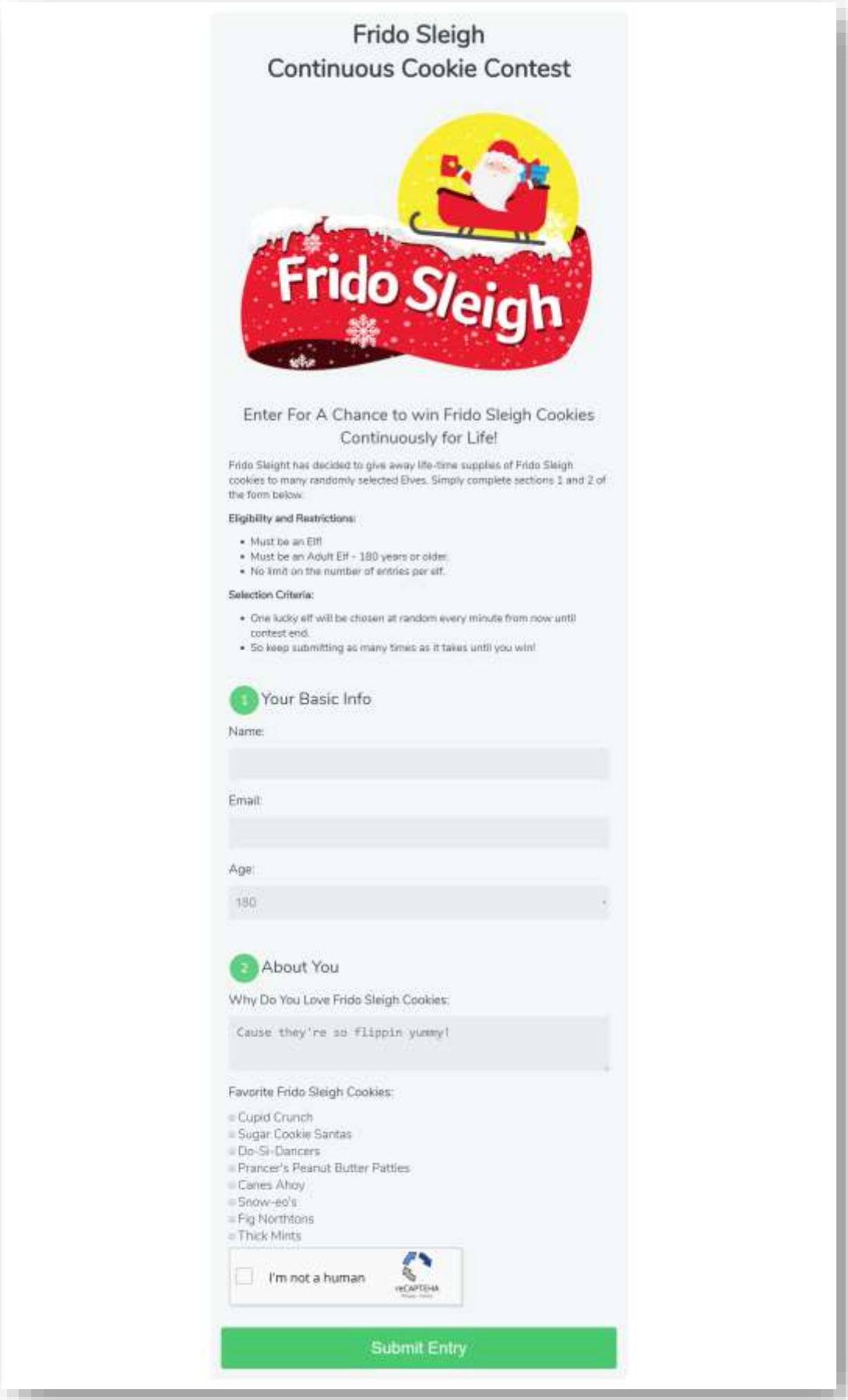


Figure 120 – Capteha Screenshot

At this point, it would be crass not to acknowledge the great work of Chris Davis, specifically his work around the code[41] that helped solve this objective.

The code used to solve this objective leaned *very heavily* on what Chris produced and it would be a disservice to Chris if we didn't acknowledge both his repository and video on machine learning [42].

It's not up to this report to teach about machine learning. It's a very complicated area which Chris does a better job of covering than we ever will and one in which there is a plethora of resoirces available.

To complete this objective, given the example Chris laid out for us in comparing apples to bananas, it was almost a plug-and-play setup where we merely had to slightly at least one of the python scripts. Amending both scripts helped optimise the process if the machine power wasn't up to scratch.

Training

By default, `python3 retrain.py --image_dir ./training_images/` would have ran trained the TensorFlow machine learning model based on images within the designated folder, under default settings.

````
Run floating-point version of Mobilenet:

```bash
python retrain.py --image_dir ~/flower_photos \
    --tfhub_module https://tfhub.dev/google/imagenet/mobilenet_v1_100_224/feature_vector/3
```

Run Mobilenet, instrumented for quantization:

```bash
python retrain.py --image_dir ~/flower_photos/ \
    --tfhub_module https://tfhub.dev/google/imagenet/mobilenet_v1_100_224/quantops/feature_vector/3
```

These instrumented models can be converted to fully quantized mobile models via
TensorFlow Lite.

There are different Mobilenet models to choose from, with a variety of file
size and latency options.
  - The first number can be '100', '075', '050', or '025' to control the number
    of neurons (activations of hidden layers); the number of weights (and hence
    to some extent the file size and speed) shrinks with the square of that
    fraction.
  - The second number is the input image size. You can choose '224', '192',
    '160', or '128', with smaller sizes giving faster speeds.
````

Figure 121 - Capteha Screenshot

For this solution, running the floating-point version versus the one instrumented for quantization made negligible difference.

---

[41] https://github.com/chrisjd20/img_rec_tf_ml_demo

[42] https://youtu.be/jmVPLwjm_zs

However, amending the file size and latency options would make a difference. Using 025 as the first value and 128 as the second allowed the scripts to be run more reliable on less powerful machines.

Using the default settings did occasionally work and when loading on a powerful AWS instance, it always worked, but for this objective, choosing speed over accuracy was fine.

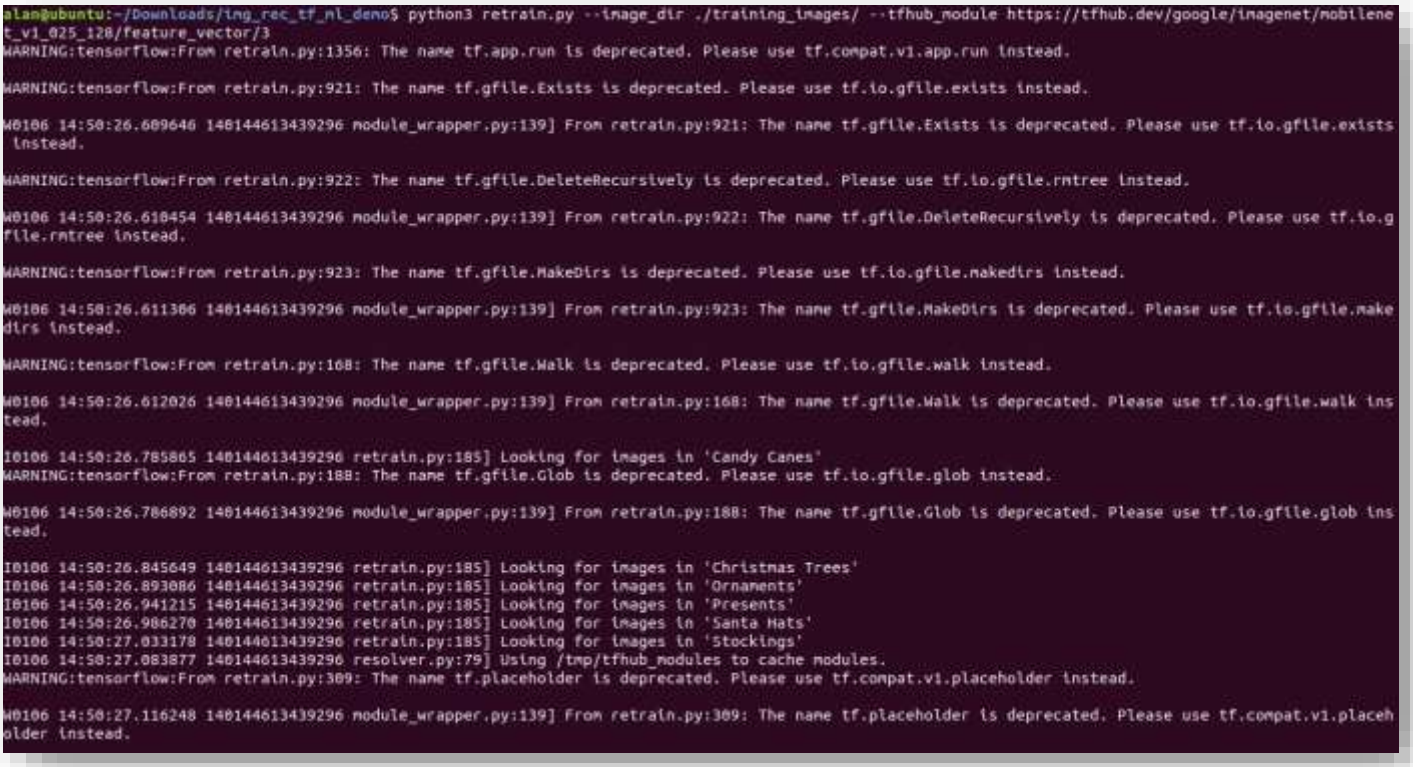


Figure 122 - Capteha Screenshot

Soliving

As mentioned before, truth be told, not a lot was changed from the original file.

The full script used is available in the appendix with comments and the output can be seen below along with the email.

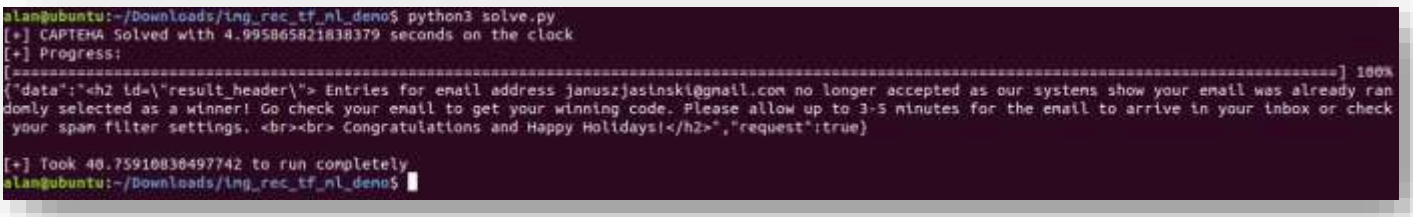


Figure 123 - Capteha Screenshot

Not long after, the email was received confirming victory.

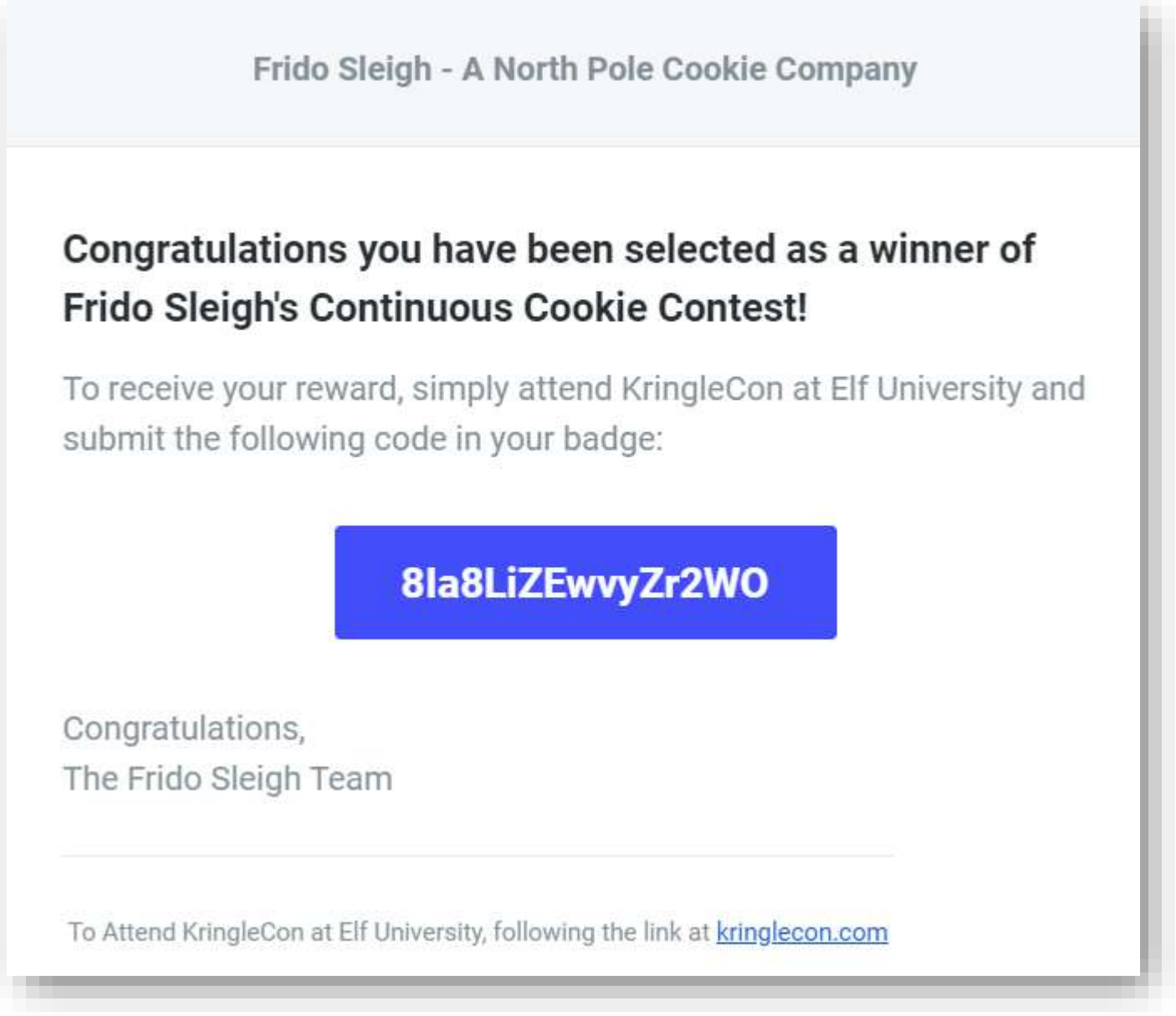


Figure 124 - Capteha Screenshot

Answer

8Ia8LiZEwvyZr2WO

Retrieve Scraps of Paper from Server

Synopsis

Gain access to the data on the Student Portal server and retrieve the paper scraps hosted there. What is the name of Santa's cutting-edge sleigh guidance system? For hints on achieving this objective, please visit the dorm and talk with Pepper Minstix.

- https://studentportal.elfu.org/

Hint

Sqlmap Tamper Scripts - https://pen-testing.sans.org/blog/2017/10/13/sqlmap-tamper-scripts-for-the-win

Solution

SQLi it is then! There seems to be two points of injection:

- https://studentportal.elfu.org/apply.php
- https://studentportal.elfu.org/check.php

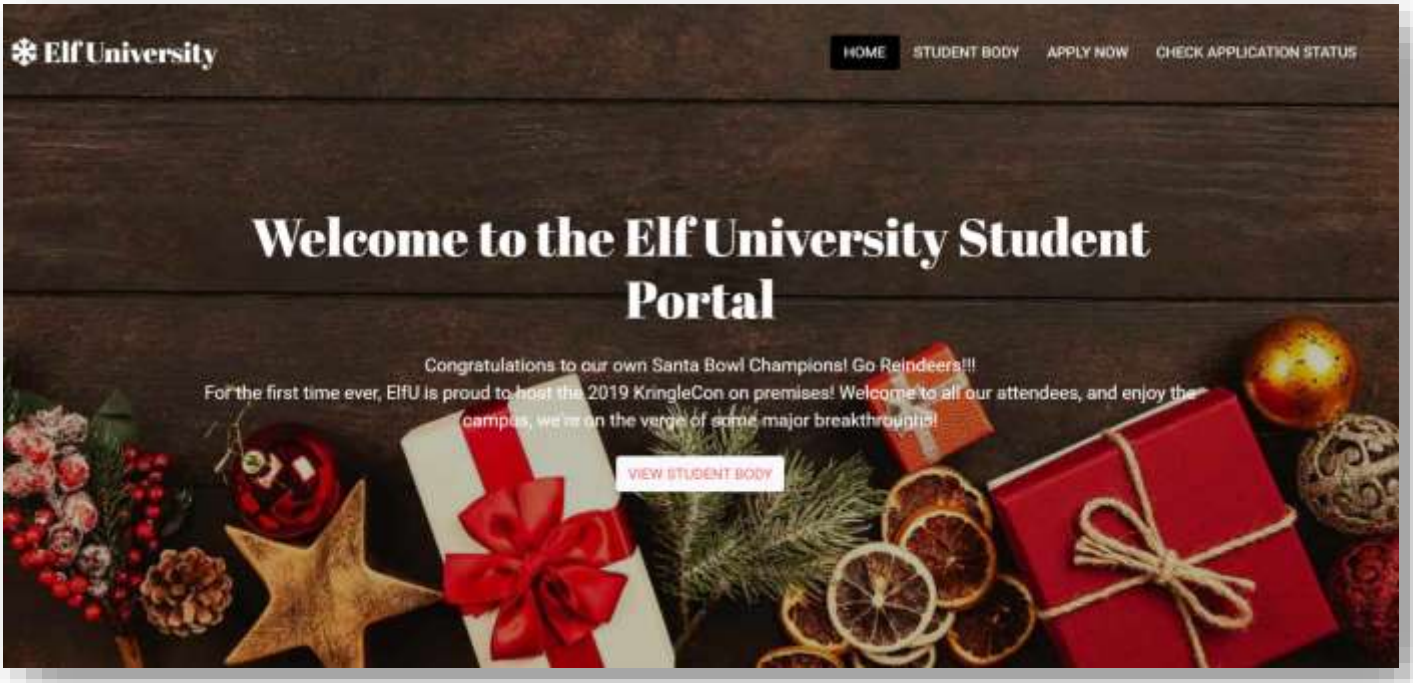


Figure 125 - SQLi Screenshot

The apply link will be inserting data and the check link will be reading data so personal preference would be to read data because otherwise it will leave the table for applications massive if we do lots of testing against it and therefore reading data would take a lot longer.

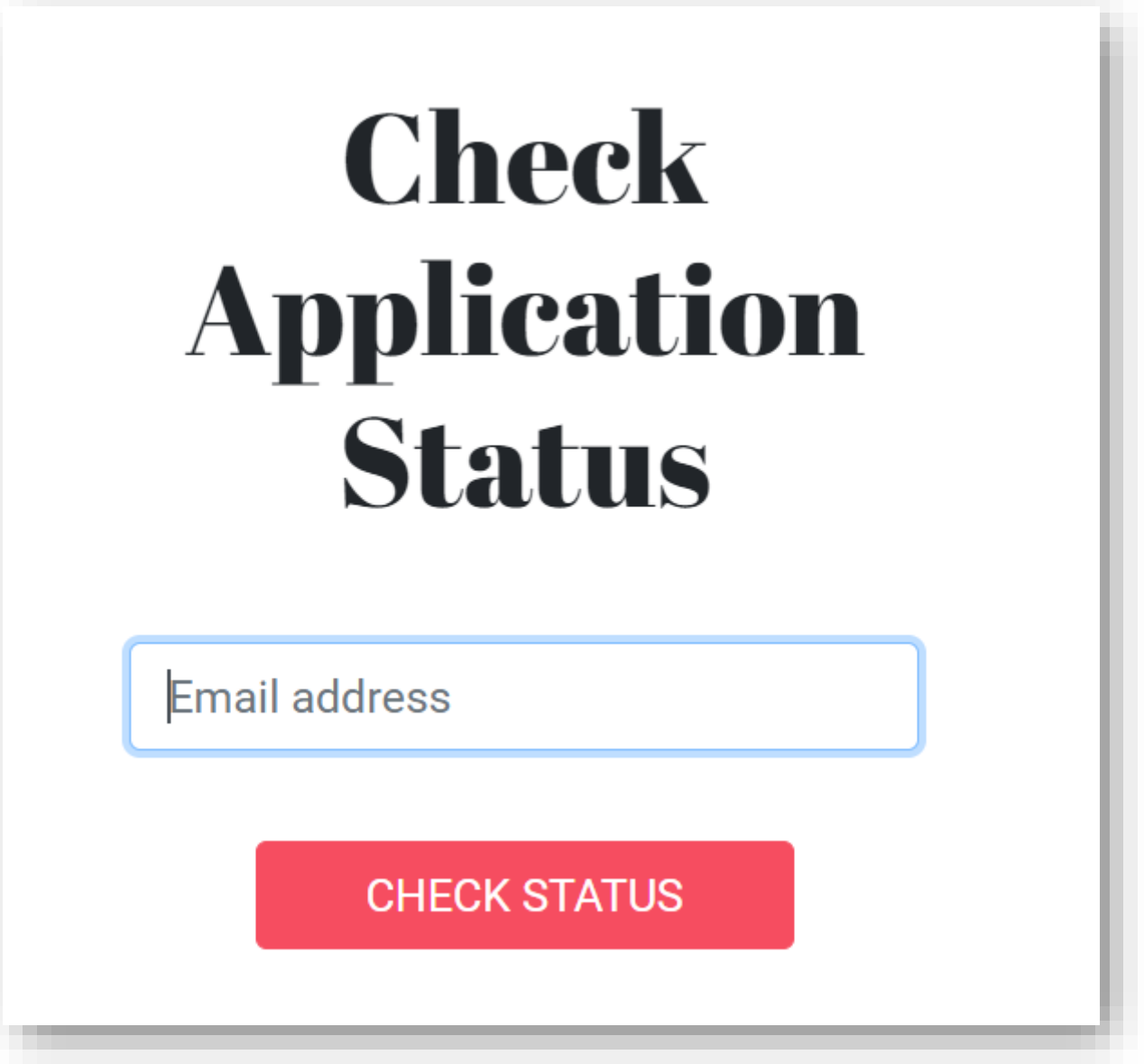


Figure 126 - SQLi Screenshot

However, adding the classic apostrophe isn't allowed.

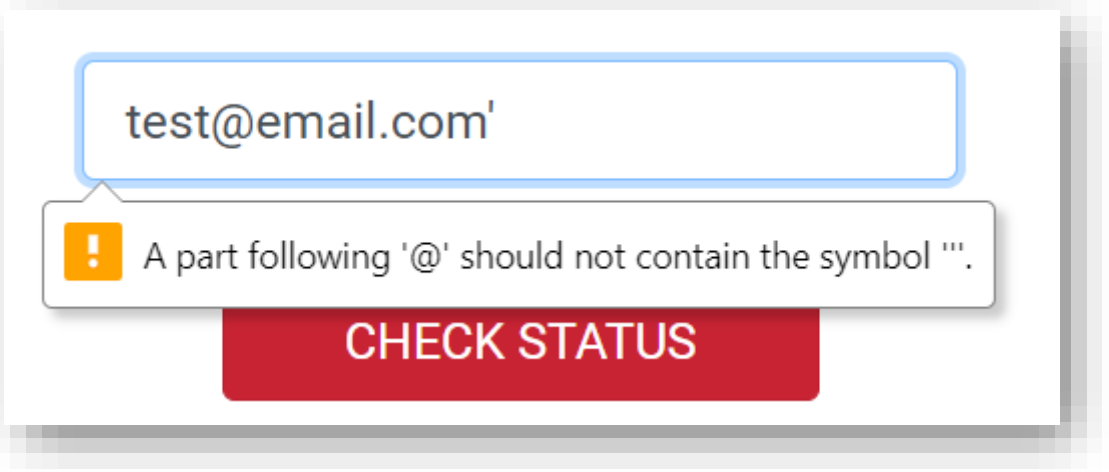


Figure 127 - SQLi Screenshot

A few ways around this – we can just amend the type of input to be standard text which will allow the form to be processed.

```
▼<div class="form-group">
    <label for="inputEmail" class="sr-only">Elf Mail Address</label>
    <input name="elfmail" type="text" id="inputEmail" class="form-control form-control-lg"
    placeholder="Email address" required autofocus> == $0
  </div>
  <input type="hidden" id="token" name="token" value>
```

Figure 128 - SQLi Screenshot

There's also a token field that's hidden. It's empty on page load but there is some JavaScript that on submission of the form, gets a token from `validator.php` and populates the field before the form finally gets submitted.

```
▼<script>

        function submitApplication() {
          console.log("Submitting");
          elfSign();
          document.getElementById("check").submit();
        }
        function elfSign() {
          var s = document.getElementById("token");

          const Http = new XMLHttpRequest();
          const url='/validator.php';
          Http.open("GET", url, false);
          Http.send(null);

          if (Http.status === 200) {
            console.log(Http.responseText);
            s.value = Http.responseText;
          }

        }


  </script>
```

Figure 129 - SQLi Screenshot

This seems to be CSRF[43]. An example of a string generated by the file is:
MTAxMDE5MDcxMjk2MTU3ODQyMjk4OTEwMTAxOTA3MS4yOTY=_MTI5MzA0NDExMjU4ODgzMjMyNjEwMjgxLjQ3Mg==

It's two parts of base64 encoded strings. Decoded both halves reveal a string of numbers that seem to be loosely based on time. Running it a dozen or more times in quick succession reveals something like the below, with the 3rd column being the difference between the two

1009693460481577646032100969346.048'	1292407629414432310190 73.536'	-1292407629414430000000000
1009693461121577646033100969346.112'	1292407630233632310190 75.584'	-1292407630233630000000000
1009693461121577646033100969346.112'	1292407630233632310190 75.584'	-1292407630233630000000000
1009693461761577646034100969346.176'	1292407631052832310190 77.632'	-1292407631052830000000000
1009693461761577646034100969346.176'	1292407631052832310190 77.632'	-1292407631052830000000000
1009693462401577646035100969346.24'	1292407631872032310190 79.68'	-1292407631872030000000000
1009693462401577646035100969346.24'	1292407631872032310190 79.68'	-1292407631872030000000000
1009693463041577646036100969346.304'	1292407632691232310190 81.728'	-1292407632691230000000000
1009693463041577646036100969346.304'	1292407632691232310190 81.728'	-1292407632691230000000000
1009693463681577646037100969346.368'	1292407633510432310190 83.776'	-1292407633510430000000000
1009693463681577646037100969346.368'	1292407633510432310190 83.776'	-1292407633510430000000000
1009693464321577646038100969346.432'	1292407634329632310190 85.824'	-1292407634329630000000000
1009693464321577646038100969346.432'	1292407634329632310190 85.824'	-1292407634329630000000000
1009693464961577646039100969346.496'	1292407635148832310190 87.872'	-1292407635148830000000000
1009693464961577646039100969346.496'	1292407635148832310190 87.872'	-1292407635148830000000000
1009693465601577646040100969346.56'	1292407635968032310190 89.92'	-1292407635968030000000000
1009693466241577646041100969346.624'	1292407636787232310190 91.968'	-1292407636787230000000000
1009693466241577646041100969346.624'	1292407636787232310190 91.968'	-1292407636787230000000000
1009693466881577646042100969346.688'	1292407637606432310190 94.016'	-1292407637606430000000000
1009693466881577646042100969346.688'	1292407637606432310190 94.016'	-1292407637606430000000000
1009693467521577646043100969346.752'	1292407638425632310190 96.064'	-1292407638425630000000000
1009693467521577646043100969346.752'	1292407638425632310190 96.064'	-1292407638425630000000000
1009693468161577646044100969346.816'	1292407639244832310190 98.112'	-1292407639244830000000000
1009693468161577646044100969346.816'	1292407639244832310190 98.112'	-1292407639244830000000000
1009693468801577646045100969346.88'	1292407640064032310191 00.16'	-1292407640064030000000000
1009693468801577646045100969346.88'	1292407640064032310191 00.16'	-1292407640064030000000000
1009693469441577646046100969346.944'	1292407640883232310191 02.208'	-1292407640883230000000000
1009693469441577646046100969346.944'	1292407640883232310191 02.208'	-1292407640883230000000000
1009693470081577646047100969347.008'	1292407641702432310191 04.256'	-1292407641702430000000000

Figure 130 - SQLi Screenshot

The first column has a mix of epoch/unix time and increases with every request.

Moving on with the SQLi attempt, we submit the form and get an error from the following page which has the following URL: https://studentportal.elfu.org/application-check.php?elfmail=test%40email.com%27&token=MTAxMDE5MDg2MjcyMTU3ODQyMzIyMzEwMTAxOTA4Ni4yNzI%3D_MTI5MzA0NDMwNDI4MTYzMjMyNjEwNzYwLjcwNA%3D%3D.

[43] https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

Figure 131 - SQLi Screenshot

Refreshing the page, removing the token or putting a different token in results in another error.

Figure 132 - SQLi Screenshot

Let's throw SQLMAP at the URL and see what happens.

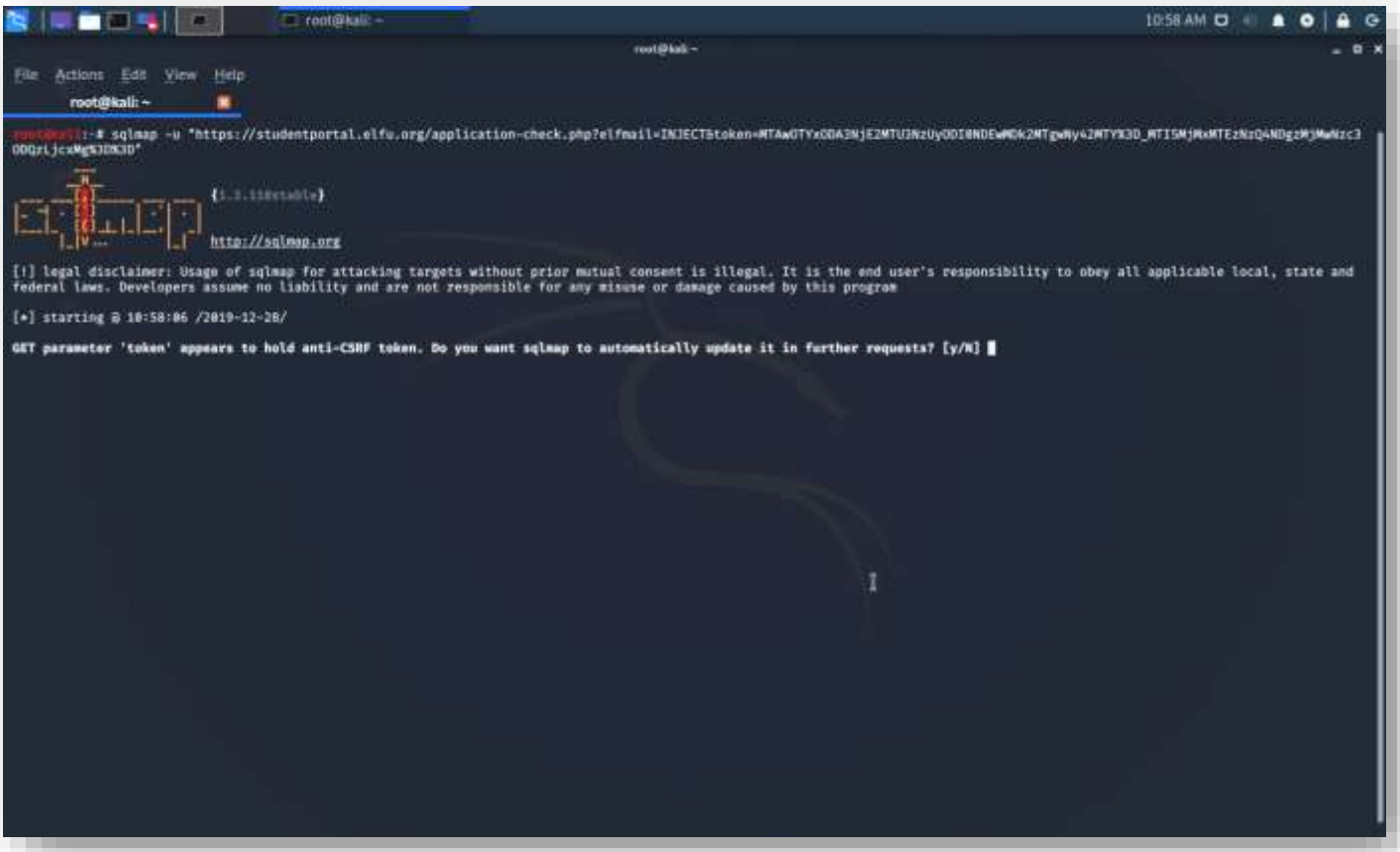


Figure 133 - SQLi Screenshot

SQLMAP recognises the token as an anti-CSRF token. In order to use it, we can use some of the parameters that SQLMAP allows[44].

Figure 134 - SQLi Screenshot

Unfortunately, this errors out. Reading certain resources[45], it seems we have to provide the token in a certain way. We therefore setup a file on our server where it outputs the token in as many ways as is reasonable.

[44] https://github.com/sqlmapproject/sqlmap/wiki/Usage
[45] https://github.com/sqlmapproject/sqlmap/issues/2

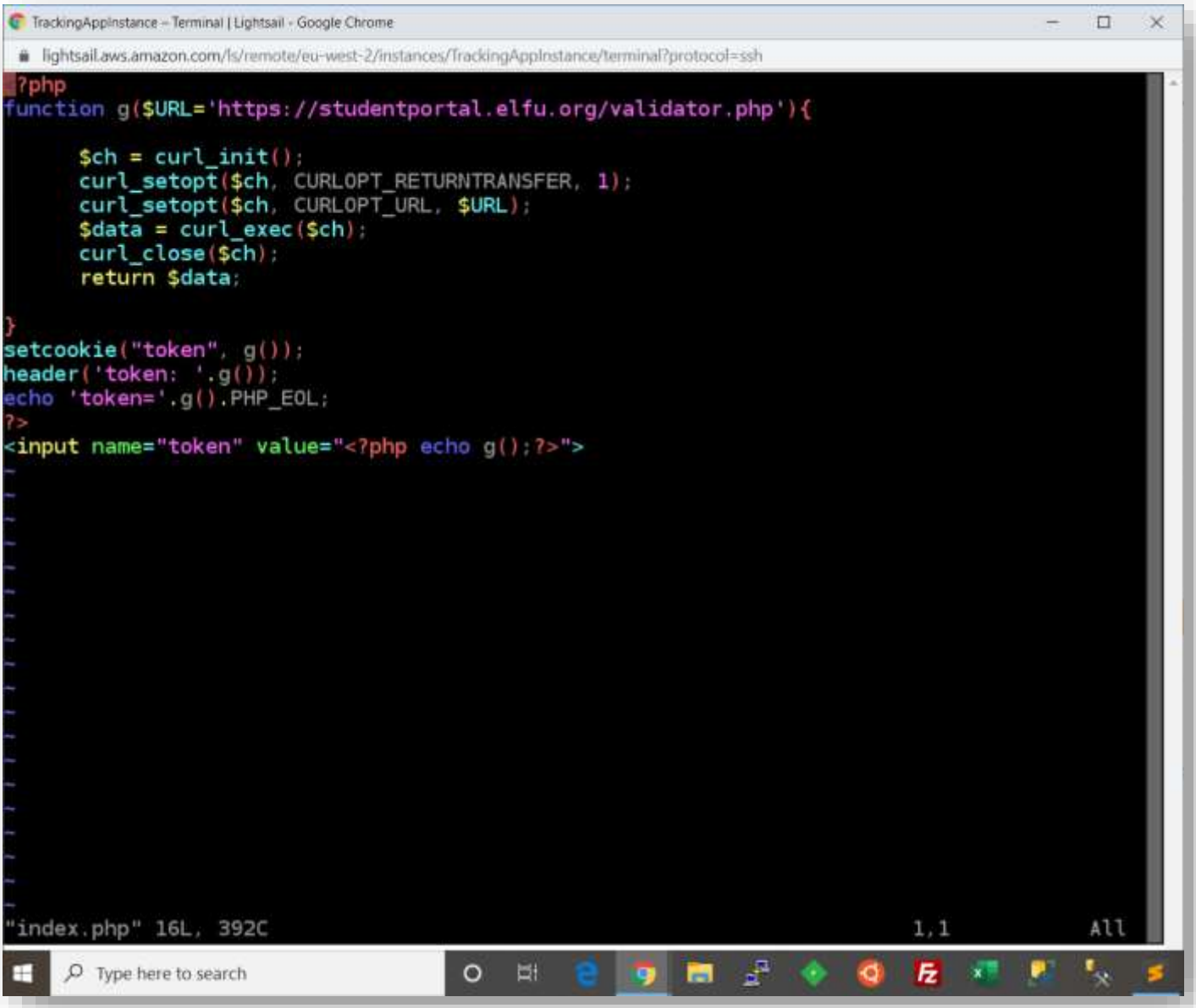


Figure 135 - SQLi Screenshot

The script is re-run but instead of pointing at the validator URL, we point it at our own server.

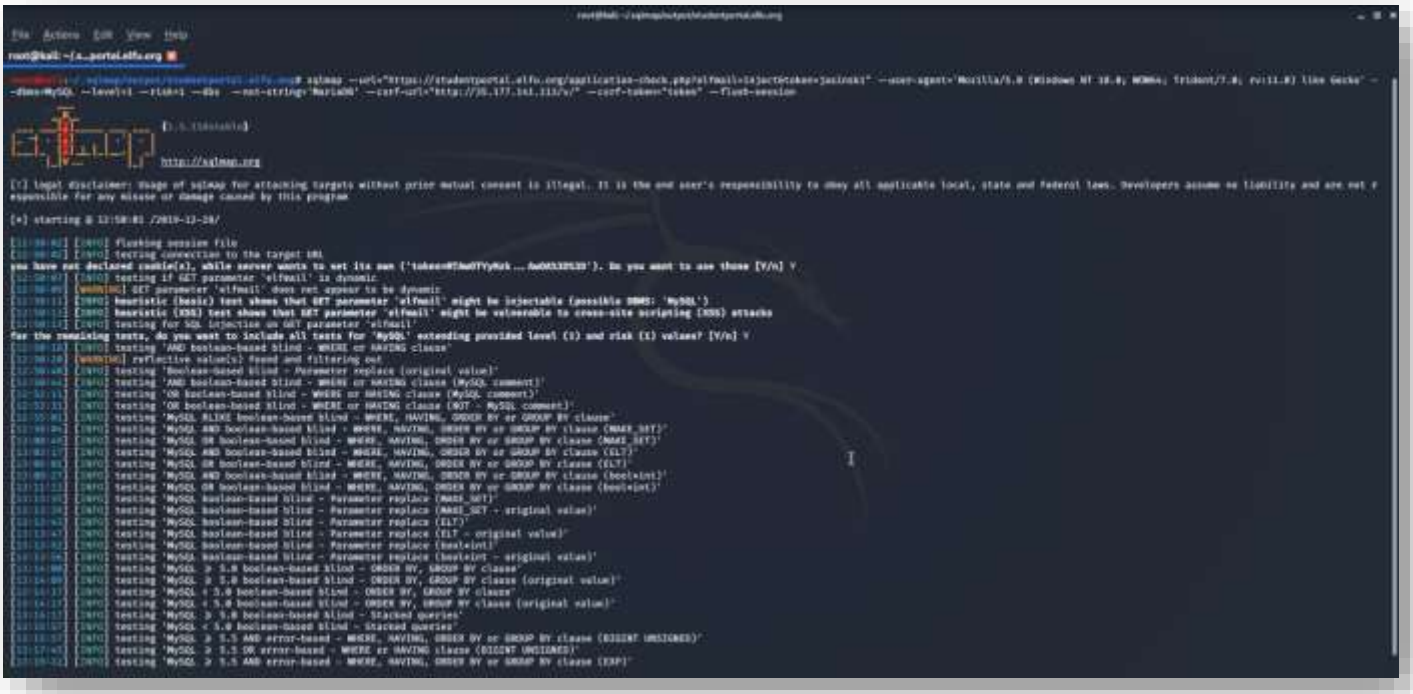


Figure 136 - SQLi Screenshot

Looks more promising this time around so we let the processing continue. Whilst it runs, we see if we can take advantage of another SQLMAP parameter – `eval`.

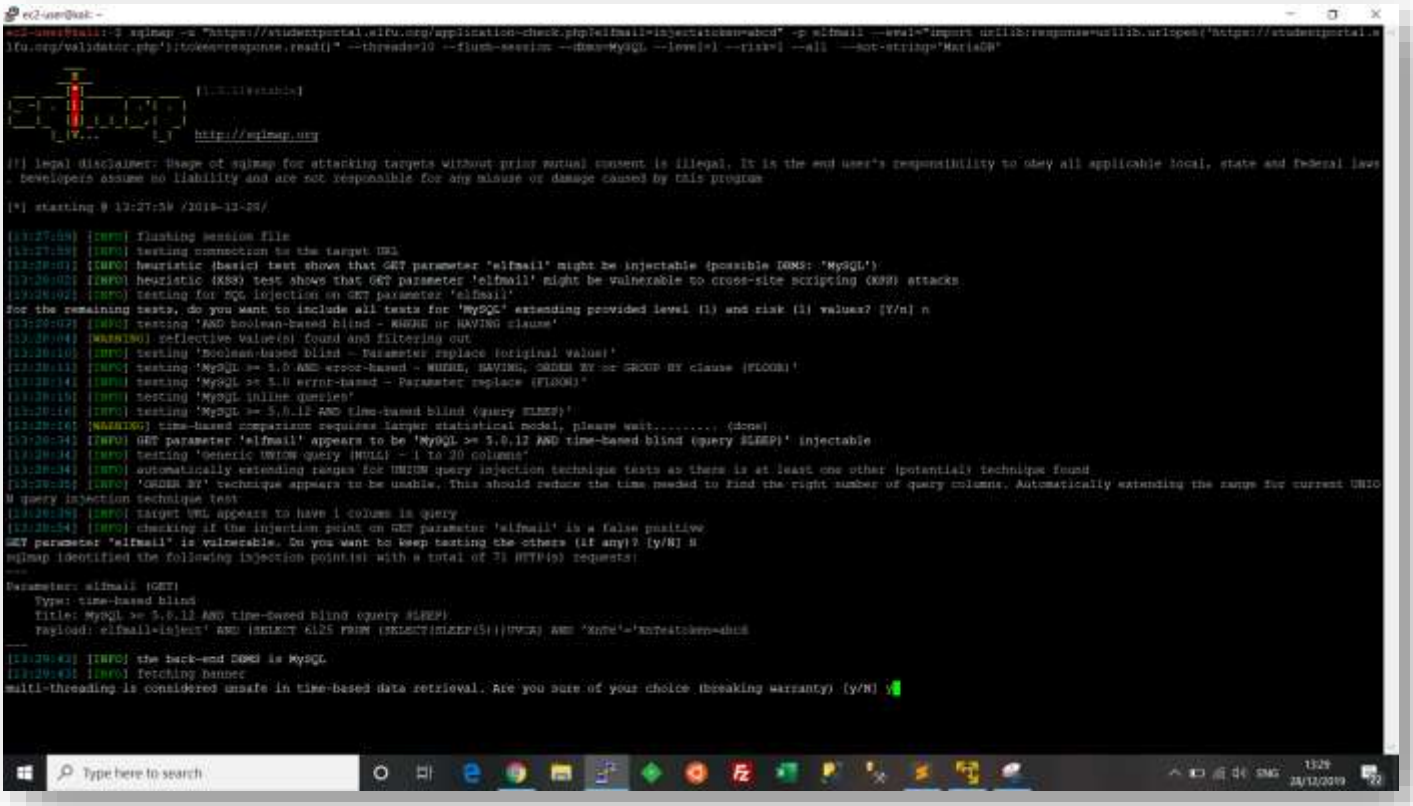


Figure 137 - SQLi Screenshot

Seems this way works as well but since the first route is further down the line, we'll continue with that.

We start getting some data through.

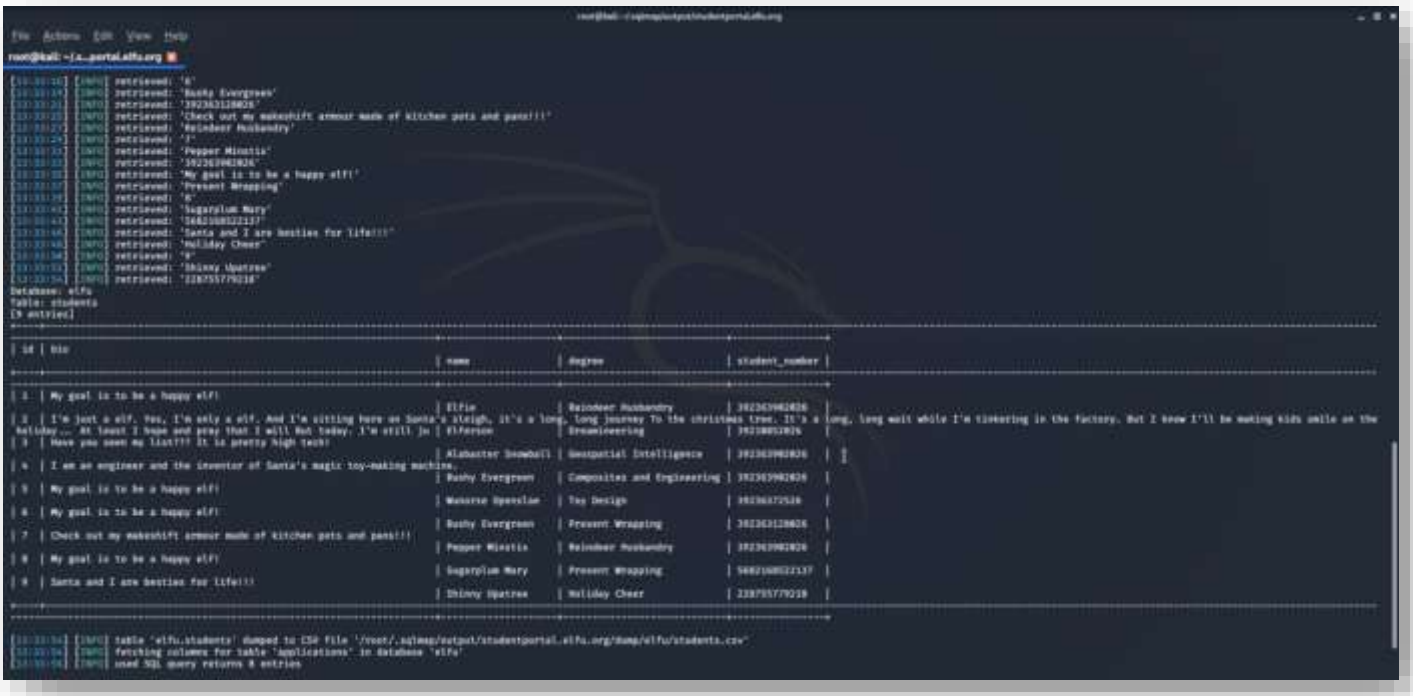


Figure 138 - SQLi Screenshot

At the time of writeup, the applications table had in excess of 25k records which looked to be full of SQLMAP attempts so we cancel out of that early and just target the table we want.

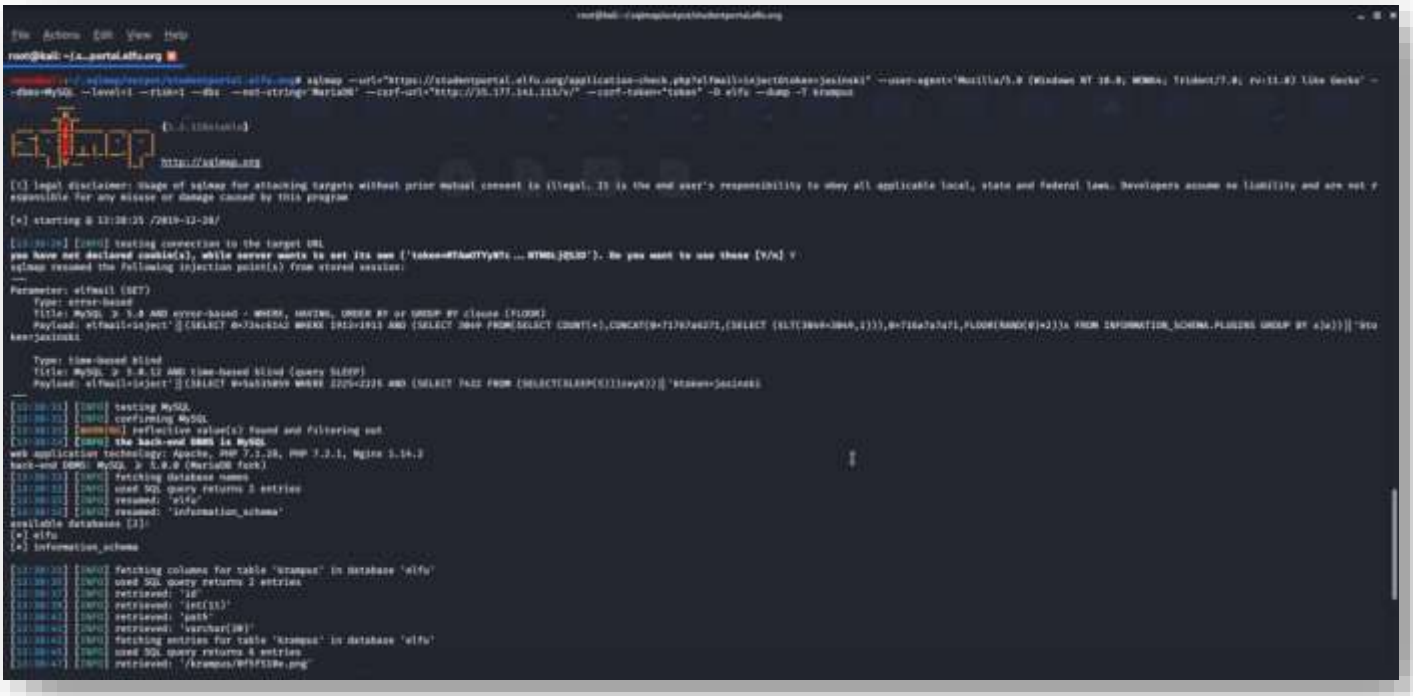


Figure 139 - SQLi Screenshot

In the end, we downloaded two tables – `students` and `krampus`, both of which sat in the `elfu` database.

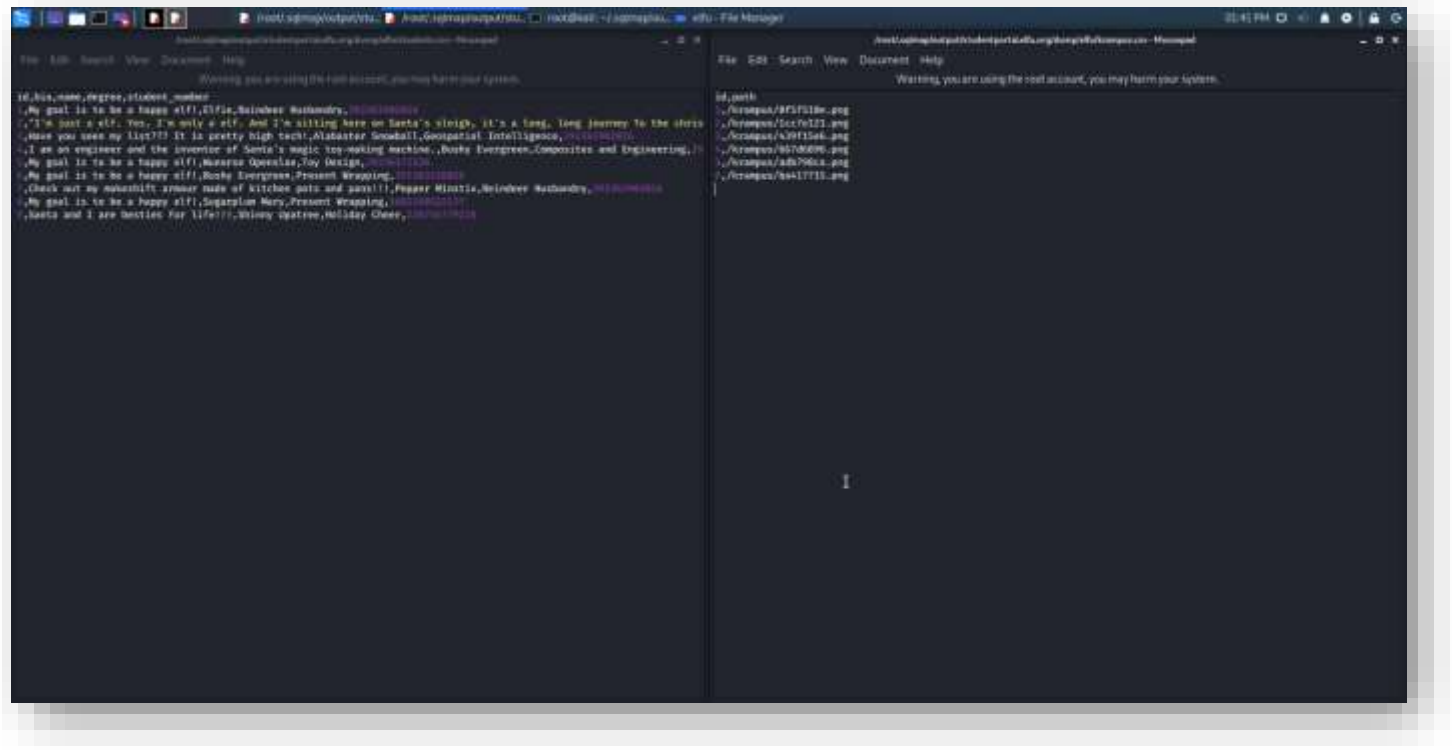


Figure 140 - SQLi Screenshot

The records in the `krampus` table look like URLs so we append them to the https://studentportal.elfu.org/ URL which gives us:

- https://studentportal.elfu.org/krampus/0f5f510e.png
- https://studentportal.elfu.org/krampus/1cc7e121.png
- https://studentportal.elfu.org/krampus/439f15e6.png
- https://studentportal.elfu.org/krampus/667d6896.png
- https://studentportal.elfu.org/krampus/adb798ca.png
- https://studentportal.elfu.org/krampus/ba417715.png

This gives us several pieces of torn paper which when put together gives us an almost complete letter.

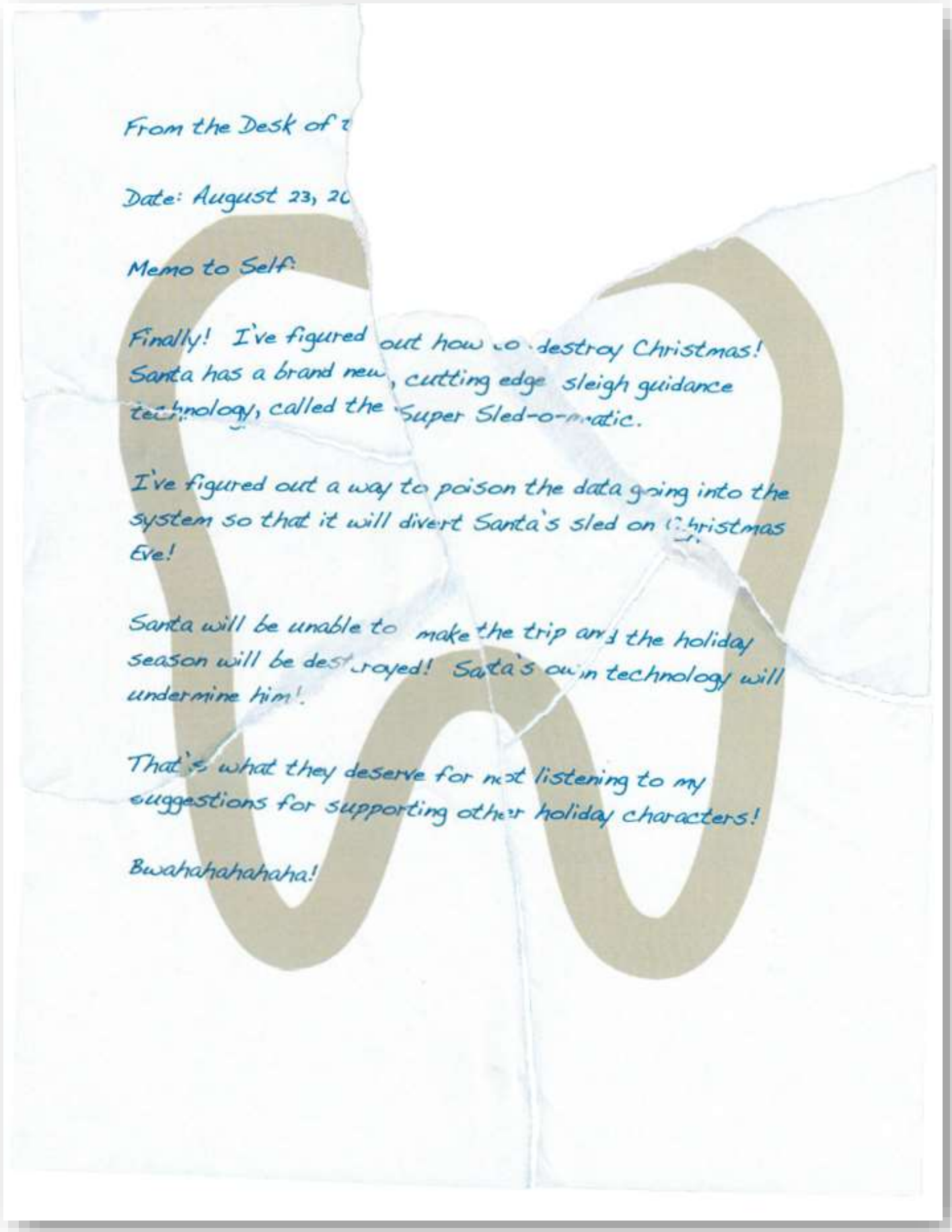


Figure 141 - SQLi Screenshot

Recover Cleartext Document

Synopsis

The Elfscrow Crypto tool is a vital asset used at Elf University for encrypting SUPER SECRET documents. We can't send you the source, but we do have debug symbols that you can use.

Recover the plaintext content for this encrypted document. We know that it was encrypted on December 6, 2019, between 7pm and 9pm UTC.

What is the middle line on the cover page? (Hint: it's five words)

For hints on achieving this objective, please visit the NetWars room and talk with Holly Evergreen.

- https://downloads.elfu.org/elfscrow.exe
- https://downloads.elfu.org/elfscrow.pdb
- https://downloads.elfu.org/ElfUResearchLabsSuperSledOMaticQuickStartGuideV1.2.pdf.enc

Hint

Reversing Crypto the Easy Way - https://youtu.be/obJdpKDpFBA

Solution

Having downloaded the files, we load the executable and get presented with the following.

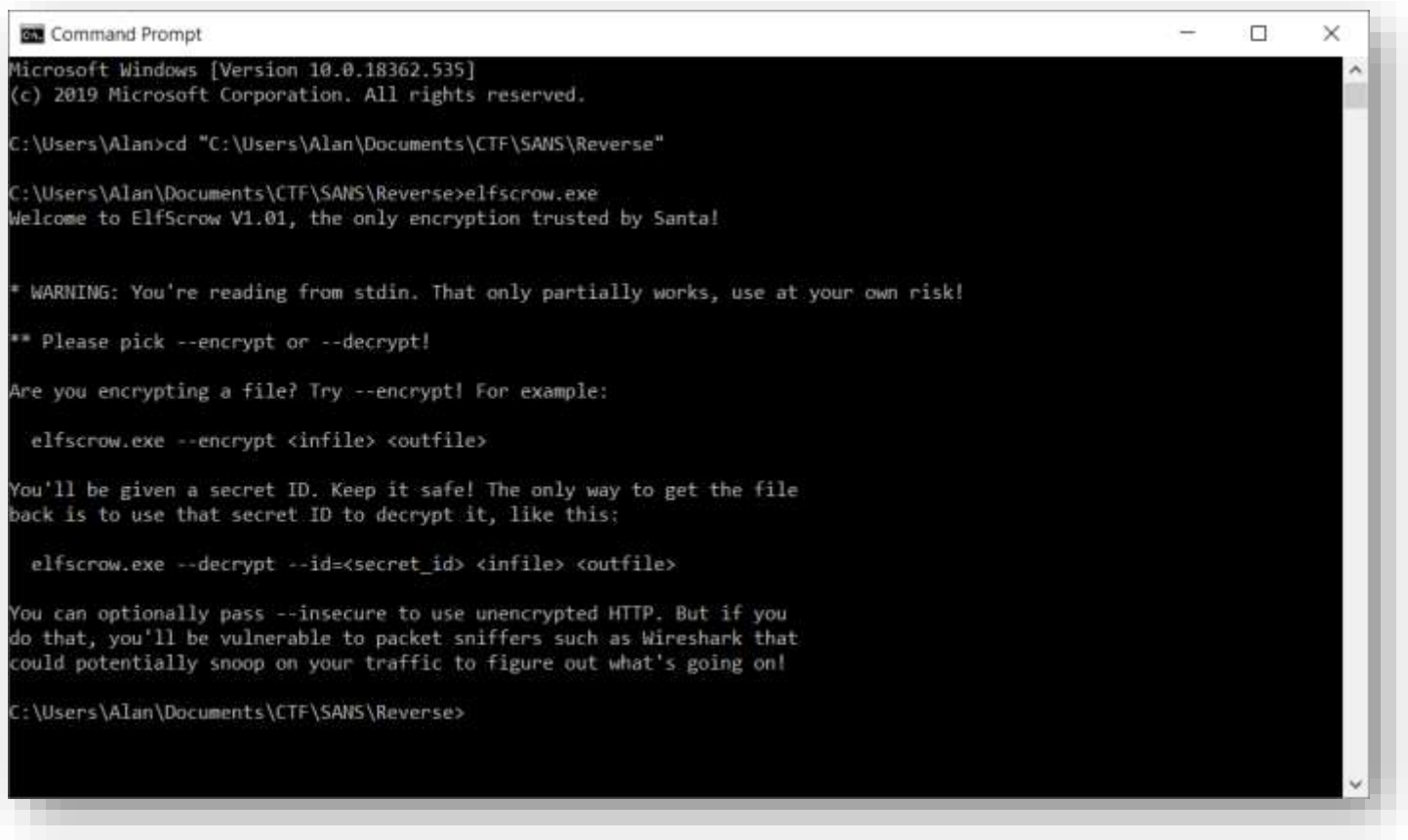


Figure 142 - Recover Cleartext Screenshot

With this, we encode a file and see what happens.

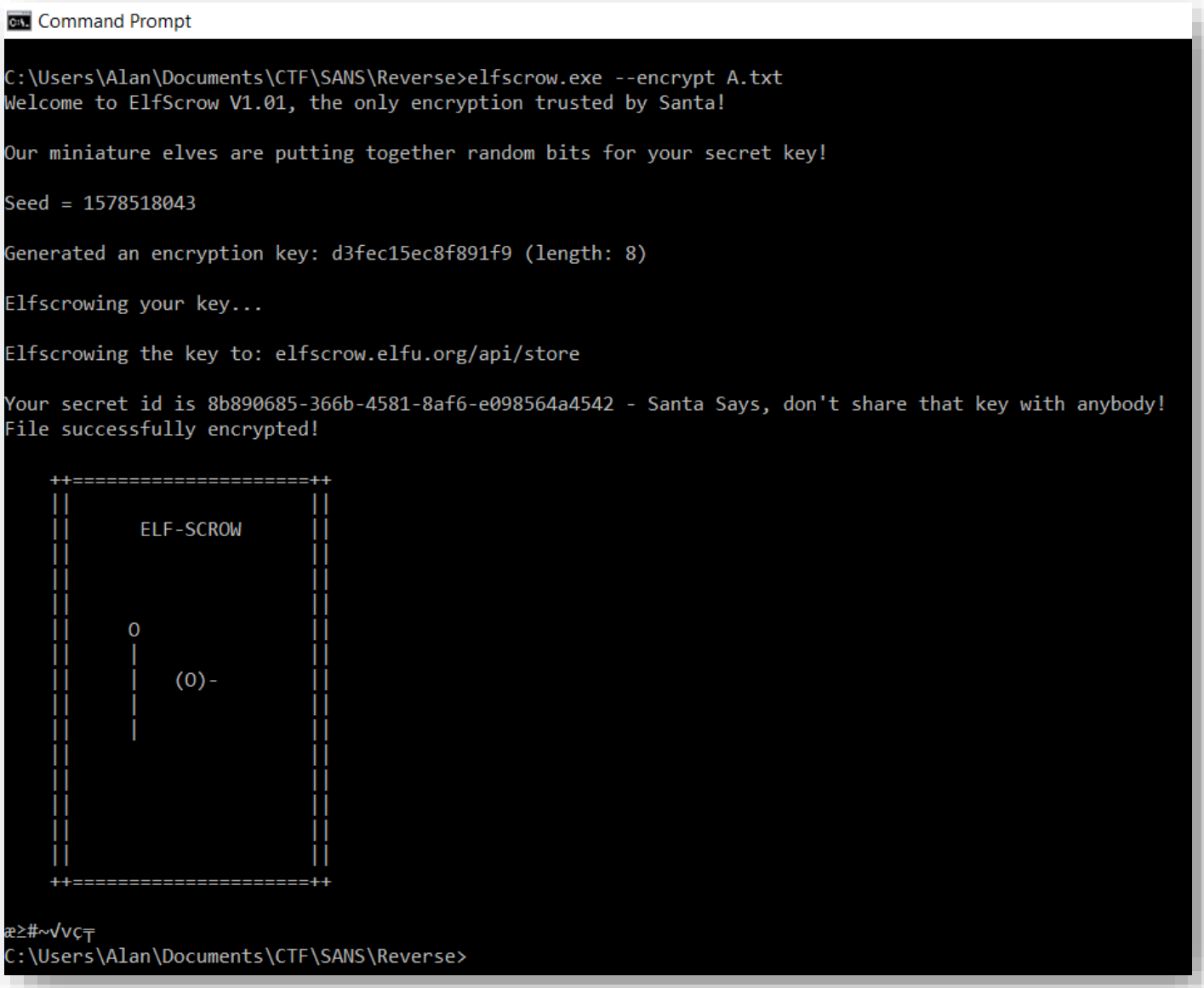


Figure 143 - Recover Cleartext Screenshot

The seed looks familiar – and indeed, it's an epoch timestamp[46] that references the time the script was ran.

Figure 144 - Recover Cleartext Screenshot

[46] https://www.epochconverter.com/

In the initial screenshot, we also see a line about passing –insecure and in passing it, we will be able to use Wireshark[47] to monitor the traffic.

Firstly, we encrypt the file and see what happens in Wireshark.

```
C:\Users\Alan\Documents\CTF\SANS\Reverse>elfscrow.exe --encrypt A.txt A.enc --insecure
Welcome to ElfScrow V1.01, the only encryption trusted by Santa!

*** WARNING: This traffic is using insecure HTTP and can be logged with tools such as Wireshark

Our miniature elves are putting together random bits for your secret key!

Seed = 1578519407

Generated an encryption key: 3ad94bb61f131816 (length: 8)

Elfscrowing your key...

Elfscrowing the key to: elfscrow.elfu.org/api/store

Your secret id is 673608e8-9edf-41b9-8c94-07c9cf1c261c - Santa Says, don't share that key with anybody!
File successfully encrypted!

    ++======================++
    ||                      ||
    ||      ELF-SCROW       ||
    ||                      ||
    ||                      ||
    ||                      ||
    ||     O                ||
    ||     |                ||
    ||     |   (O)-         ||
    ||     |                ||
    ||     |                ||
    ||                      ||
    ||                      ||
    ||                      ||
    ||                      ||
    ||                      ||
    ++======================++
```

Figure 145 - Recover Cleartext Screenshot

Now let's see what Wireshark has to say.

[47] https://www.wireshark.org/

Figure 146 - Recover Cleartext Screenshot

We can see that the secret id in the command prompt was delivered in the response of a POST request to http://elfscrow.elfu.org/api/store when we post the encryption key.

Interestingly, sending the same encryption key to the same endpoint multiple times results in different secret IDs, meaning it's not derived directly from the key so even if we did find the original key, it wouldn't give us the encryption key. This was done in Burp[48].

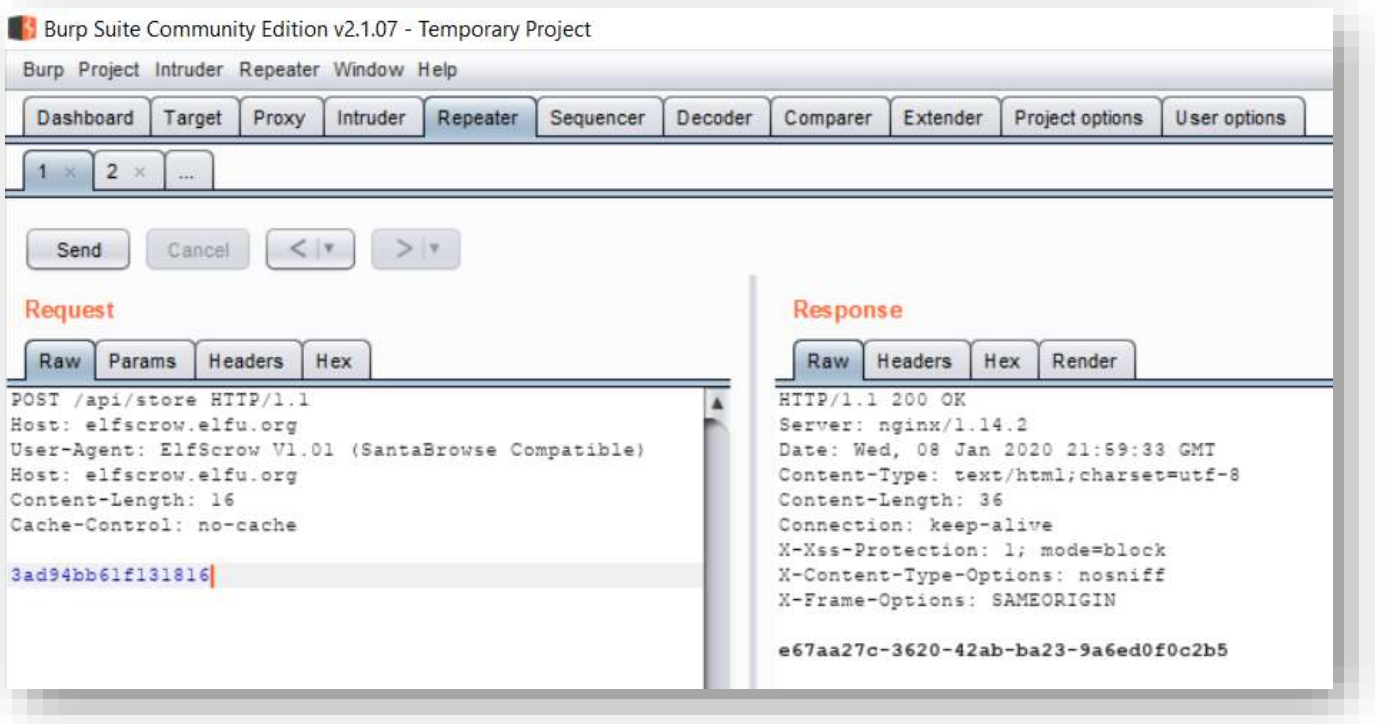


Figure 147 - Recover Cleartext Screenshot

[48] https://portswigger.net/burp

Similar results are got when decrypting.

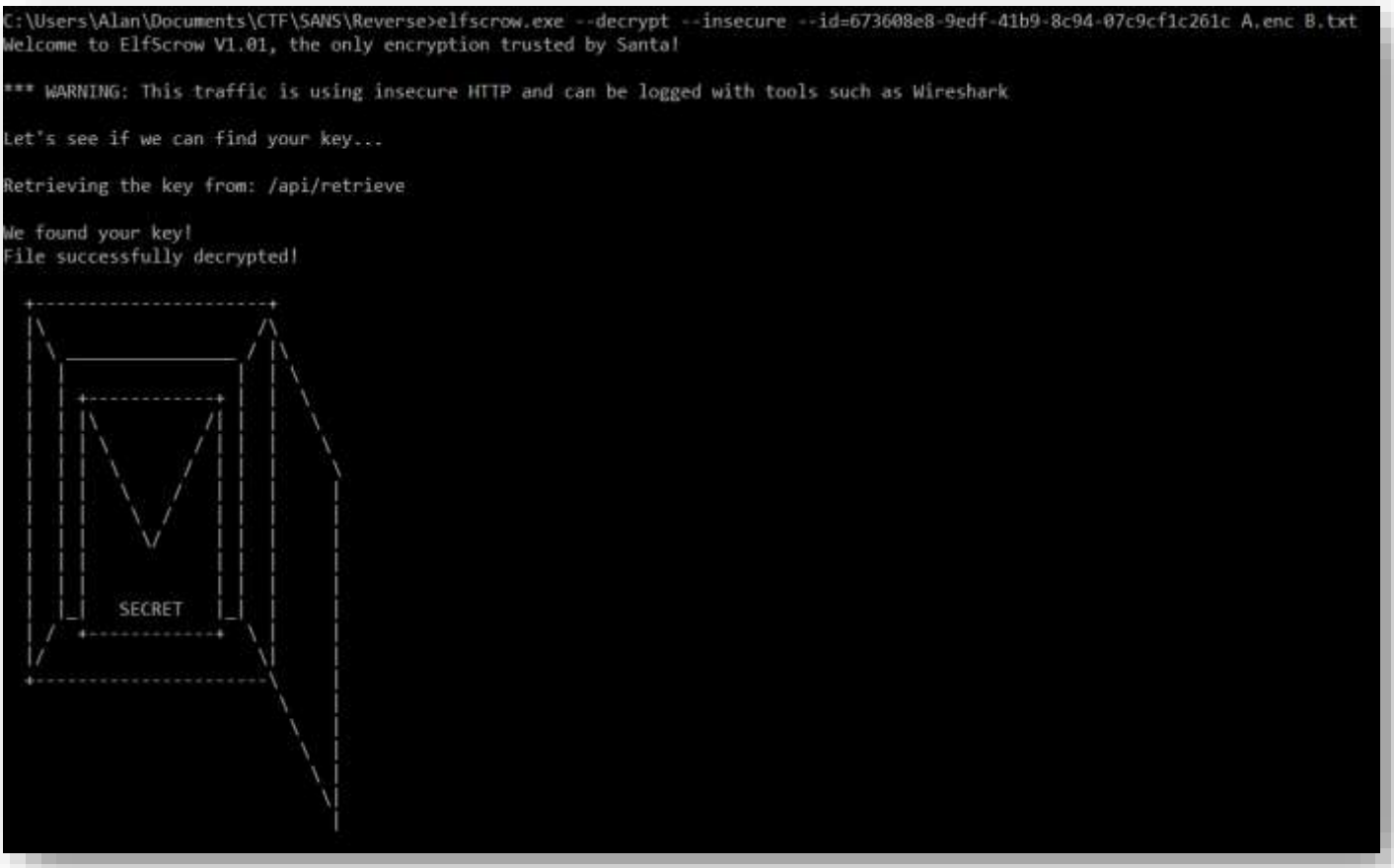


Figure 148 - Recover Cleartext Screenshot

Figure 149 - Recover Cleartext Screenshot

This time around, when posting the secret ID, we get the encryption key back.

With this in mind, how do we decrypt the file? Since we don't know the inner workings of the server and given that it gives us a different secret ID even with the same encryption key that avenue is closed down.

At this point, as with the CAPTEHA challenge, we have to make reference to the incredible work that Ron Bowes did in his video "Reversing Crypto the Easy Way"[49]. The video gives us everything we need to complete this challenge and we're not going to pretend that we didn't lean heavily on this.

Out first clue is shown below:

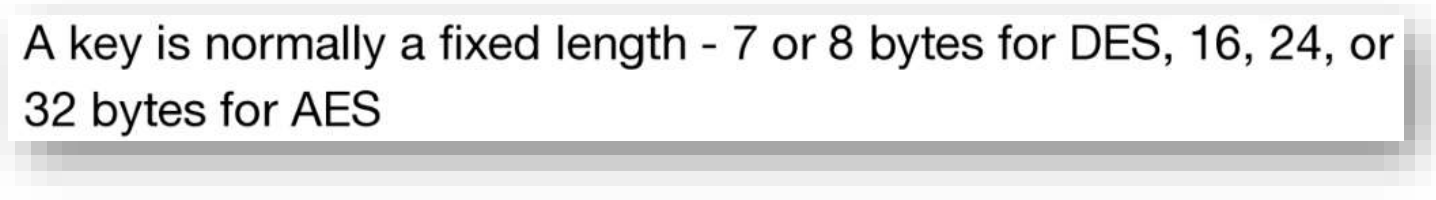


Figure 150 - Recover Cleartext Screenshot

In a previous screenshot where we encrypted the file, it told us what the key length was – shown again below.

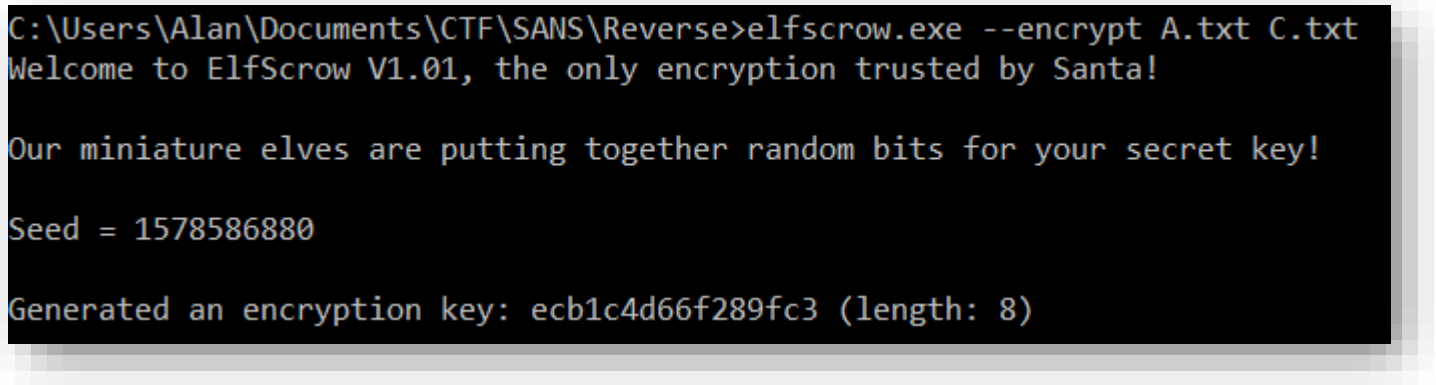


Figure 151 - Recover Cleartext Screenshot

As we can see, the key has a length of 8 and therefore it should be safe to presume that we are looking for DES[50][51].

Our next clue in the next screenshot and as pointed out in the video[52]. Running the tool on separate files at the same time, gives us the same key. This is a flag that points us to the fact that the key is generated based on current time rather than random entropy.

---

[49] https://www.youtube.com/watch?v=obJdpKDpFBA&feature=youtu.be
[50] https://en.wikipedia.org/wiki/Data_Encryption_Standard
[51] https://youtu.be/obJdpKDpFBA?t=333 – specific time
[52] https://youtu.be/obJdpKDpFBA?t=812 – specific time

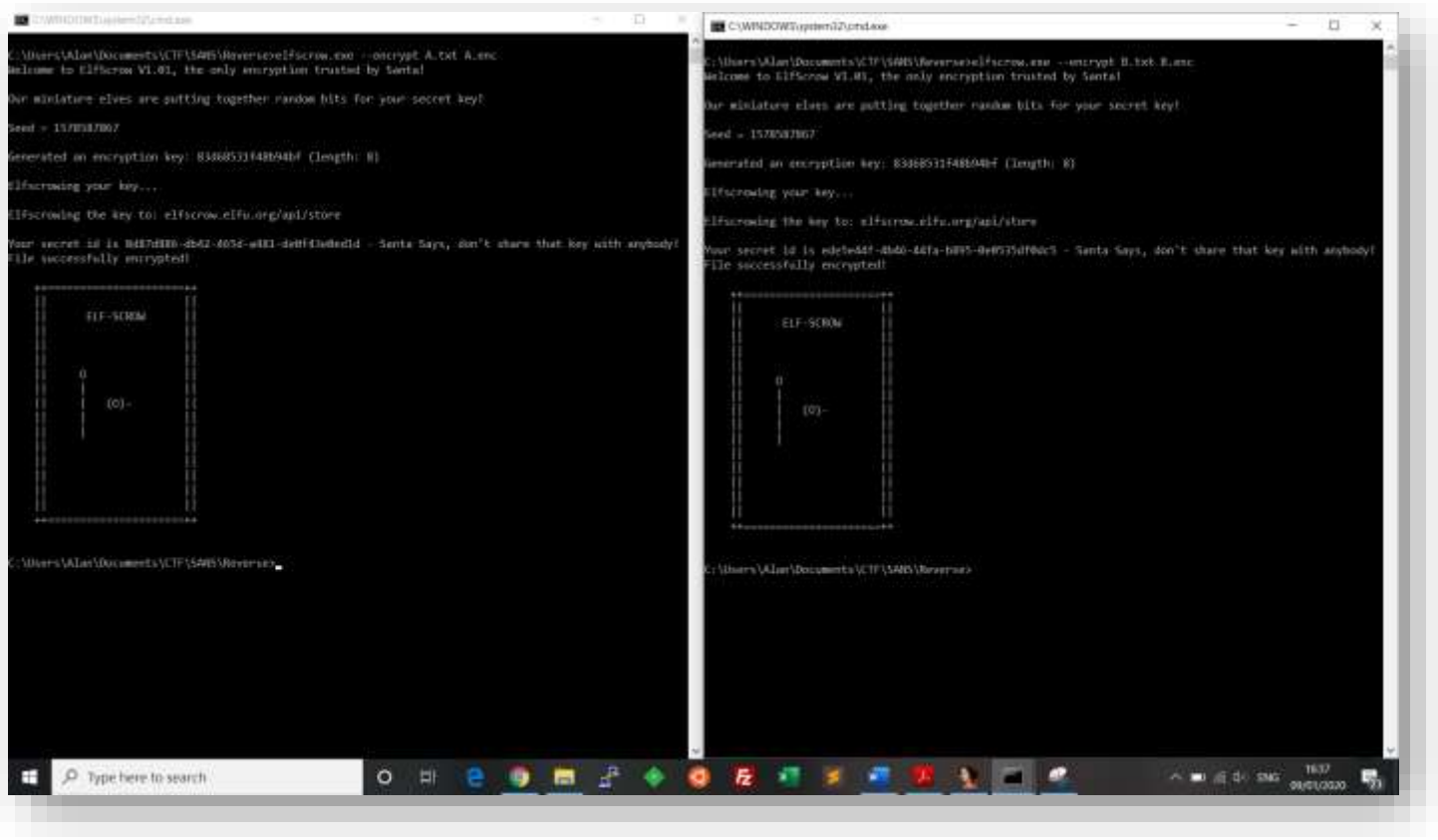


Figure 152 - Recover Cleartext Screenshot

Now we have an idea on the algorithm and next we'll get an idea about the particular mode. The video tells[53] us that if we encode 7 characters, we should get a length of 8 as a result (for CBC mode[54]). If we encode between 8 and 15, we get a length of 16 characters, between 16 and 23 we get 24, illustrated below.

INPUT LENGTH	ENCODED LENGTH
1-7	8
8-15	16
16-23	24
24-31	32
32-29	40
40-47	48
48-55	56
56-63	64
64-71	72
72-79	80

Figure 153 - Recover Cleartext

We run the tool against files with varying length strings and can confirm that the table above is being adhered to. This can be seen in the next screenshot where the first column shows the file length and the first the filename that corresponds to the original file length.

[53] https://www.youtube.com/watch?v=obJdpKDpFBA&feature=youtu.be&t=1600 – specific time

[54] https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

```
C:\Users\Alan\Documents\CTF\SANS\Reverse>for %I in (*.enc) do @echo %~znI
8 1
24 16
32 24
72 64
8 7
72 71
16 8
```

Figure 154 - Recover Cleartext Screenshot

Let's take stock of what we know so far:

- DES cipher[55]
- CBC mode
- Key is generated based on time and not random entropy and given we have a timeframe in which we know it was encrypted[56], we might be able to decrypt the document.

The video references[57] a quick test we can do so we encode two files. One with AAAAA and one with BAAAA. If everything changes in the encoded files, it's likely to be CBC.

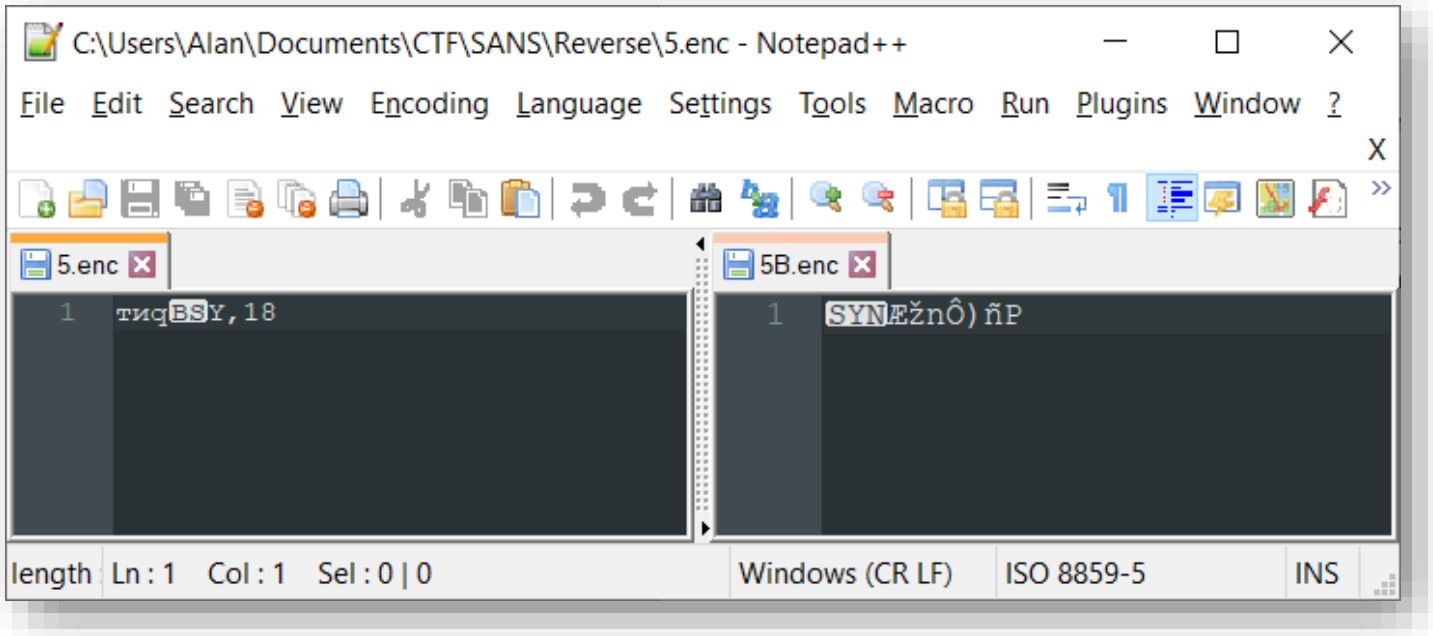


Figure 155 - Recover Cleartext Screenshot

As we can see, both are entirely different, thus adding to the thoughts that we are looking at CBC. A last sanity check, the exe is uploaded to https://onlinedisassembler.com/ where we can confirm it is DES-CBC.

---

[55] https://youtu.be/obJdpKDpFBA?t=1691 – exact time, inn addition to previous clue

[56] We know that it was encrypted on December 6, 2019, between 7pm and 9pm UTC.

[57] https://youtu.be/obJdpKDpFBA?t=1878 – exact time

0x404a5c	Microsoft Enhanced Cryptographic Provider v1.0
0x404a8c	CryptAcquireContext failed
0x404aa8	CryptImportKey failed for DES-CBC key
0x404ad0	CryptDecrypt failed
0x404ae4	File successfully decrypted!

Figure 156 - Recover Cleartext Screenshot

Knowing what we have so far and again, referencing the video[58], we should be able to do a fairly simple lift and shift of the code that Ron does gives. On the face of it, the tool is fairly close to what we have downloaded and ran based on previous screenshots.

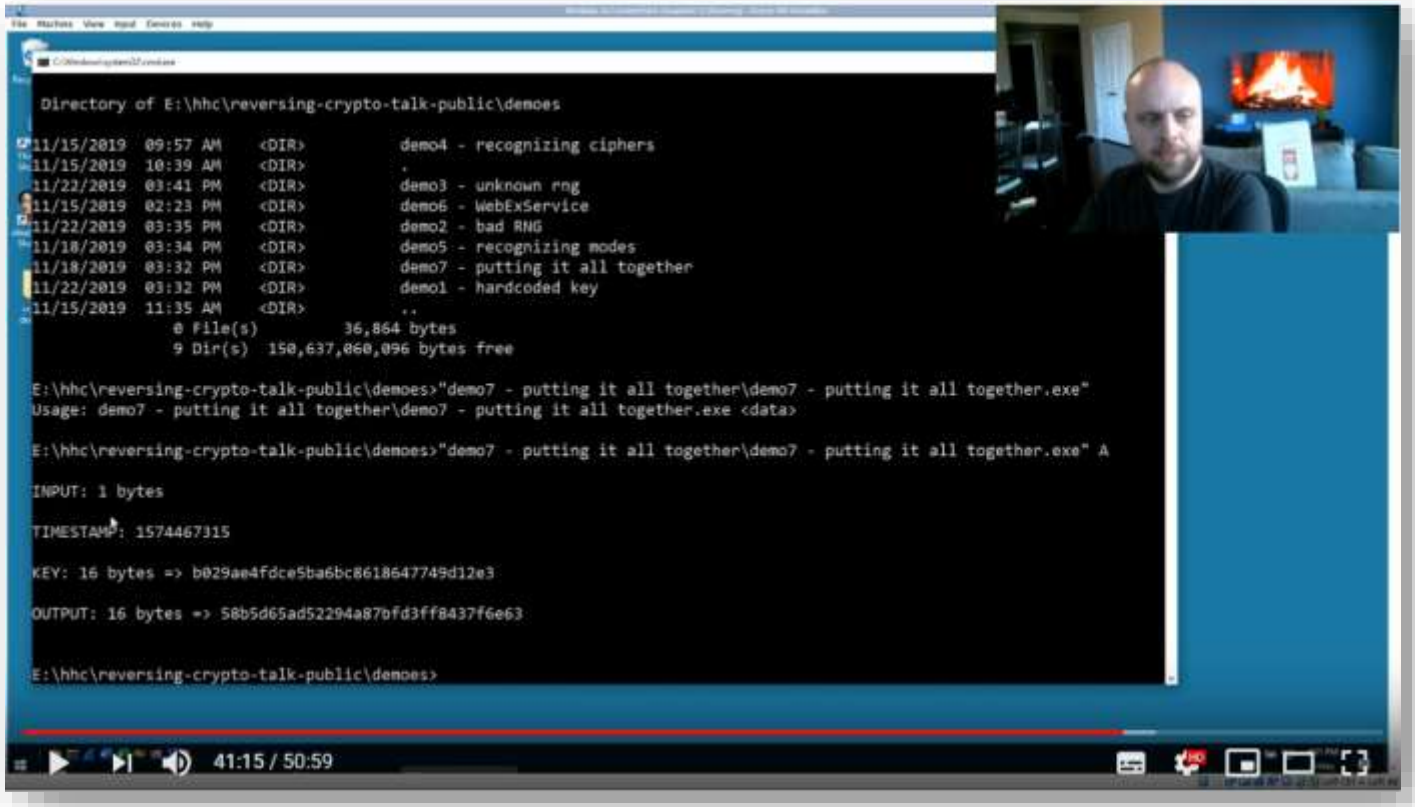


Figure 157 - Recover Cleartext Screenshot

Following the video, we find similar patterns and we start off with `do_encrypt`.

[58] https://youtu.be/obJdpKDpFBA?t=2430 – exact time

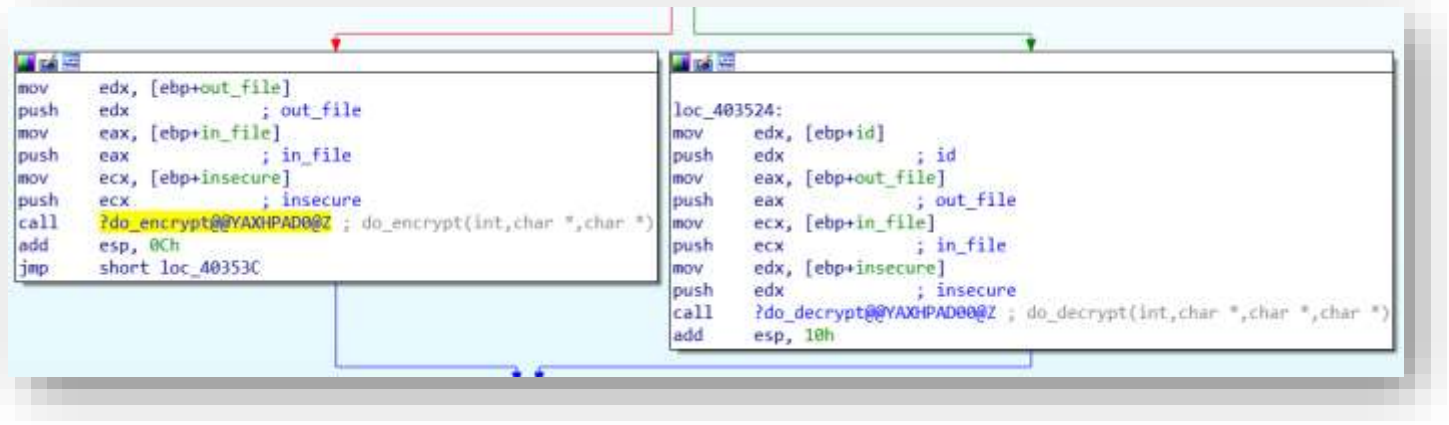


Figure 158 - Recover Cleartext Screenshot

Going into this, we have some "weird" cookie stuff going on and `CryptAcquireContext`[59] being called which is as Ron says, the first function used when encrypting data on Windows.

```
push    ebp
mov     ebp, esp
sub     esp, 30h
mov     eax, ___security_cookie
xor     eax, ebp
mov     [ebp+var_10], eax
lea     eax, [ebp+data_len]
push    eax             ; len
mov     ecx, [ebp+in_file]
push    ecx             ; filename
call    ?read_file@@YAPAEPADPAK@Z ; read_file(char *,ulong *)
add     esp, 8
mov     [ebp+data], eax
mov     edx, [ebp+data_len]
add     edx, 10h
push    edx             ; NewSize
mov     eax, [ebp+data]
push    eax             ; Memory
call    ds:__imp__realloc
add     esp, 8
mov     [ebp+data], eax
push    0F0000000h      ; dwFlags
push    1               ; dwProvType
push    offset szProvider ; "Microsoft Enhanced Cryptographic Provid"...
push    0               ; szContainer
lea     ecx, [ebp+hProv]
push    ecx             ; phProv
call    ds:__imp__CryptAcquireContextA@20 ; CryptAcquireContextA(x,x,x,x,x)
test    eax, eax
jnz     short loc_402733
```

Figure 159 - Recover Cleartext Screenshot

[59] https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptacquirecontexta

Clicking on `CryptAcquireContext` takes us to the following screenshot where we can hopefully now, for the last time, confirm we're looking at DES-CBC.

```
szProvider[]
der       db 'Microsoft Enhanced Cryptographic Provider v1.0',0
                                        ; DATA XREF: do_encrypt(int,char *,char *)+41↑o
          align 10h
aCryptacquireco[]
cquireco db 'CryptAcquireContext failed',0
                                        ; DATA XREF: do_encrypt(int,char *,char *)+56↑o
          align 4
title[]
          db 'Generated an encryption key',0
                                        ; DATA XREF: do_encrypt(int,char *,char *)+75↑o
aCryptimportkey[]
mportkey db 'CryptImportKey failed for DES-CBC key',0
                                        ; DATA XREF: do_encrypt(int,char *,char *)+C6↑o
```

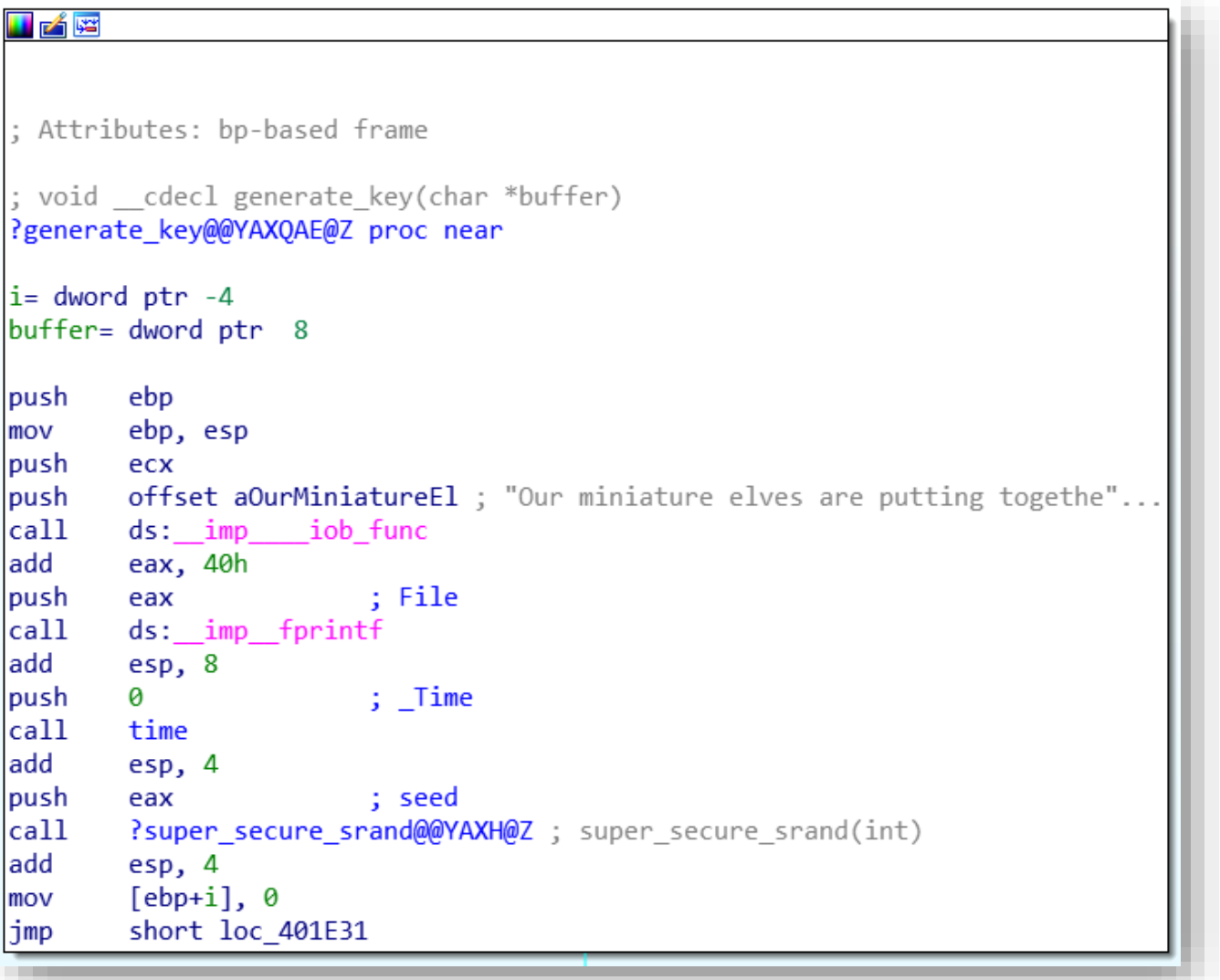
Figure 160 - Recover Cleartext Screenshot

Going further into `do_encrypt` we see `generate_key` which we can safely presume will generate a key.

```
loc_402733:
lea     edx, [ebp+key]
push    edx             ; buffer
call    ?generate_key@@YAXQAE@Z ; generate_key(uchar * const)
```

Figure 161 - Recover Cleartext Screenshot

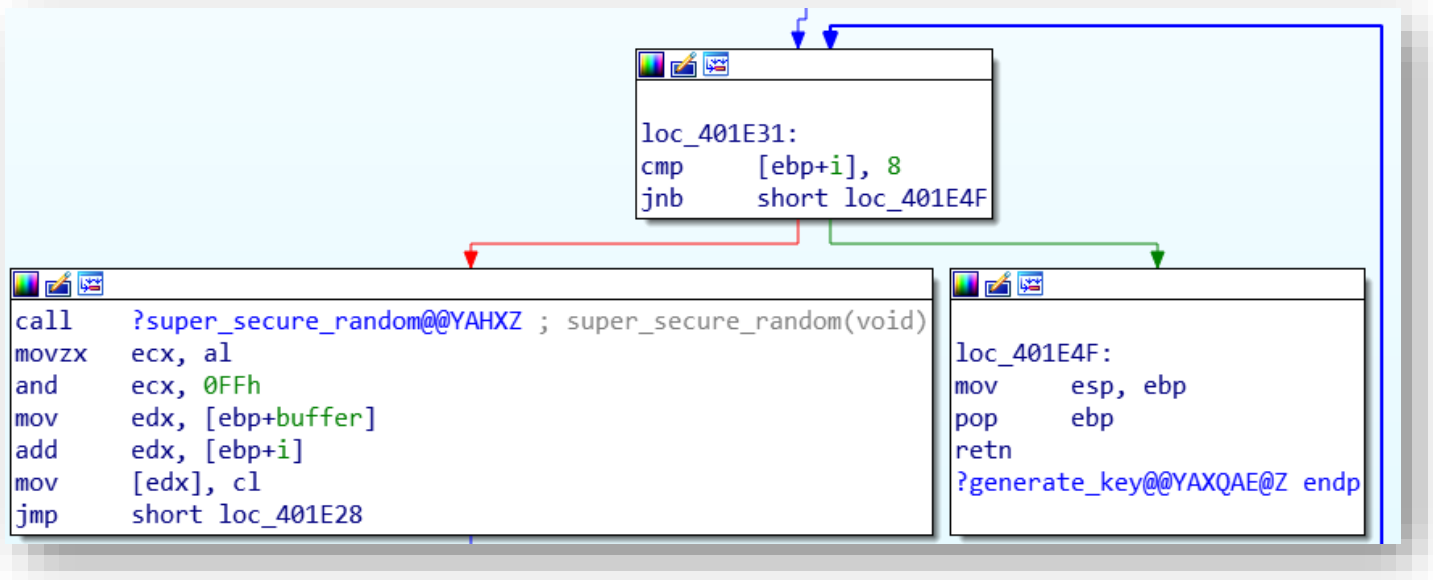
Let's now go into `generate_key` and see what it has in store for us. It calls `time` and `super_secure_srand`.



```
; Attributes: bp-based frame

; void __cdecl generate_key(char *buffer)
?generate_key@@YAXQAE@Z proc near

i= dword ptr -4
buffer= dword ptr  8

push    ebp
mov     ebp, esp
push    ecx
push    offset aOurMiniatureEl ; "Our miniature elves are putting togethe"...
call    ds:__imp____iob_func
add     eax, 40h
push    eax             ; File
call    ds:__imp__fprintf
add     esp, 8
push    0               ; _Time
call    time
add     esp, 4
push    eax             ; seed
call    ?super_secure_srand@@YAXH@Z ; super_secure_srand(int)
add     esp, 4
mov     [ebp+i], 0
jmp     short loc_401E31
```

Figure 162 - Recover Cleartext Screenshot

It then goes on to call `super_secure_random` 8 times.



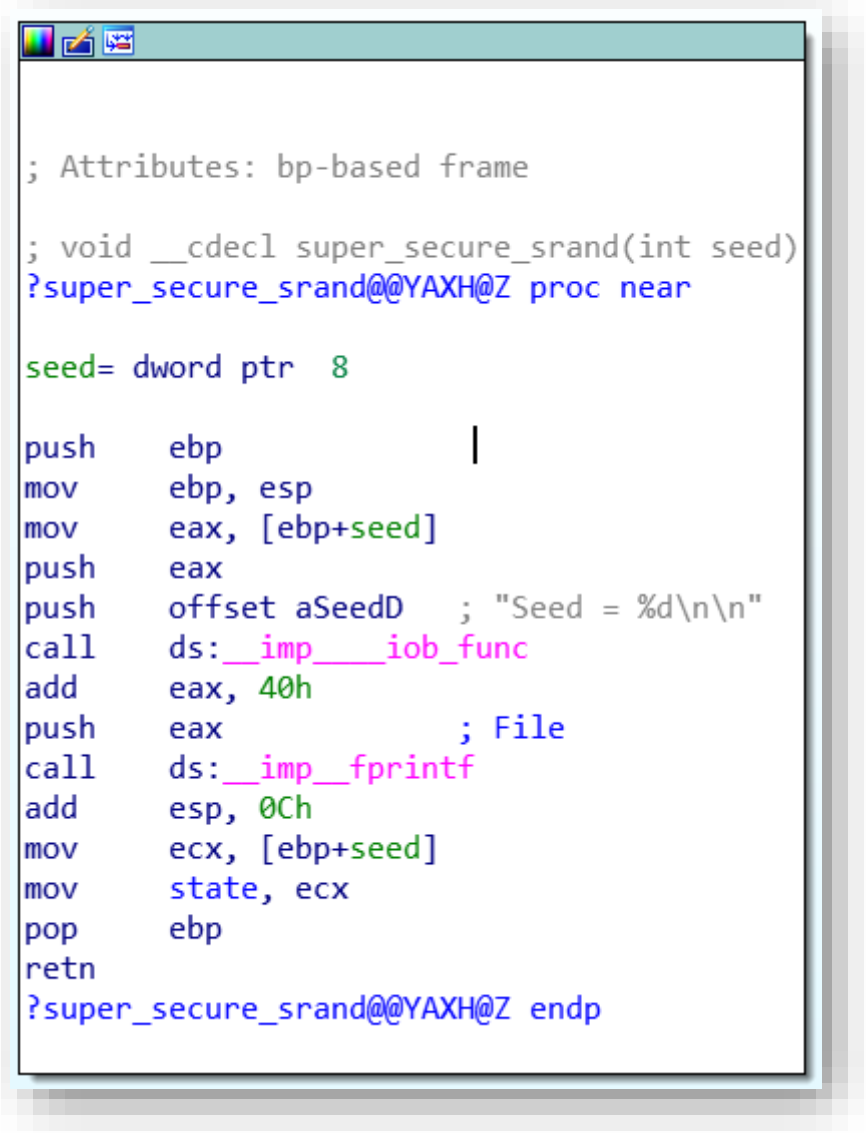
```
loc_401E31:
cmp     [ebp+i], 8
jnb     short loc_401E4F
```

```
call    ?super_secure_random@@YAHXZ ; super_secure_random(void)
movzx   ecx, al
and     ecx, 0FFh
mov     edx, [ebp+buffer]
add     edx, [ebp+i]
mov     [edx], cl
jmp     short loc_401E28
```

```
loc_401E4F:
mov     esp, ebp
pop     ebp
retn
?generate_key@@YAXQAE@Z endp
```

Figure 163 - Recover Cleartext Screenshot

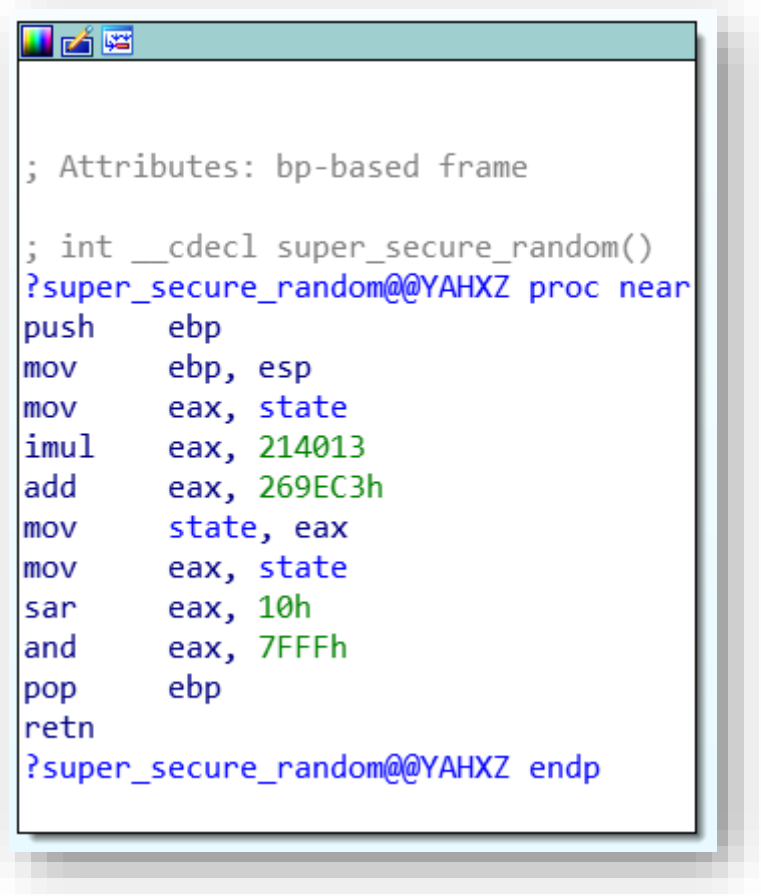
Going into `super_secure_srand` we get the following where it sores the seed in a variable called `state`:



```
; Attributes: bp-based frame

; void __cdecl super_secure_srand(int seed)
?super_secure_srand@@YAXH@Z proc near

seed= dword ptr  8

push    ebp
mov     ebp, esp
mov     eax, [ebp+seed]
push    eax
push    offset aSeedD   ; "Seed = %d\n\n"
call    ds:__imp____iob_func
add     eax, 40h
push    eax             ; File
call    ds:__imp__fprintf
add     esp, 0Ch
mov     ecx, [ebp+seed]
mov     state, ecx
pop     ebp
retn
?super_secure_srand@@YAXH@Z endp
```

Figure 164 - Recover Cleartext Screenshot

Next, we go into `super_secure_random`.



```
; Attributes: bp-based frame

; int __cdecl super_secure_random()
?super_secure_random@@YAHXZ proc near
push    ebp
mov     ebp, esp
mov     eax, state
imul    eax, 214013
add     eax, 269EC3h
mov     state, eax
mov     eax, state
sar     eax, 10h
and     eax, 7FFFh
pop     ebp
retn
?super_secure_random@@YAHXZ endp
```

Figure 165 - Recover Cleartext Screenshot

It does a *multiplication*, *addition* and an *and*, utilising the previous variable, `state`. We'll Google then `214013` value and it's an LCG[60]. Specifically, one pertaining to Microsoft[61].

Using Ghidra[62], we can go a little better and view more readable code as shown in the next few screenshots.

[60] https://en.wikipedia.org/wiki/Linear_congruential_generator

[61] https://rosettacode.org/wiki/Linear_congruential_generator#Ruby

[62] https://www.nsa.gov/resources/everyone/ghidra/

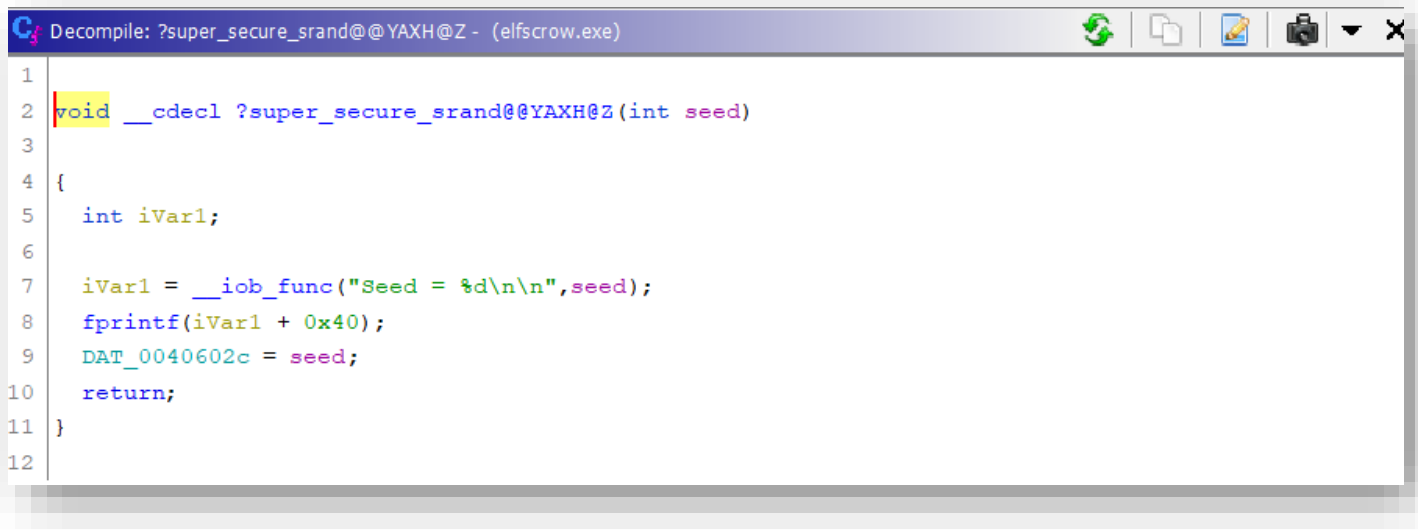


Figure 166 - Recover Cleartext Screenshot

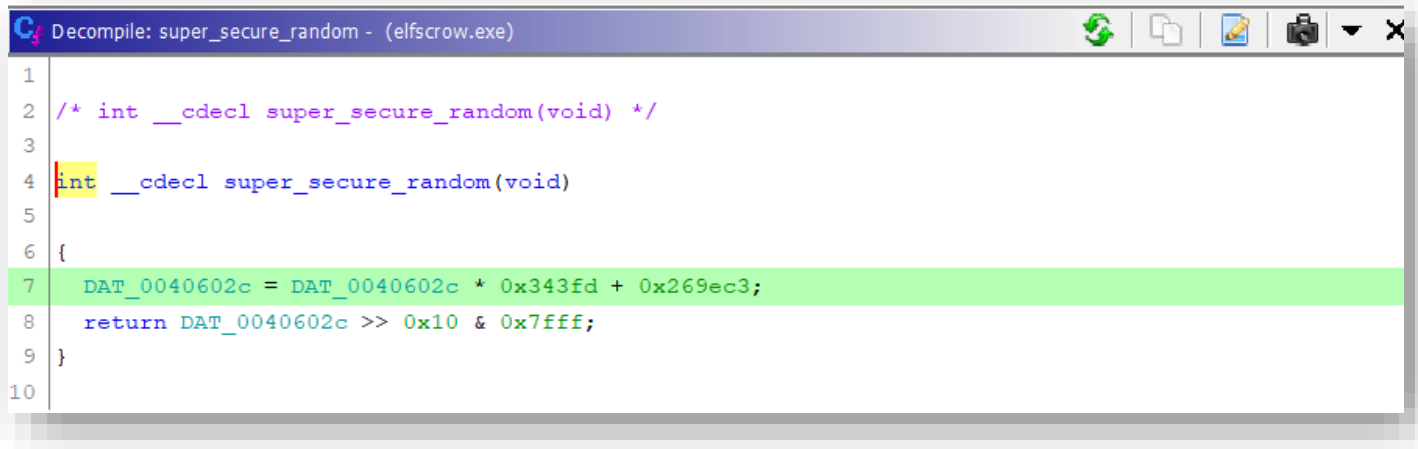


Figure 167 - Recover Cleartext Screenshot

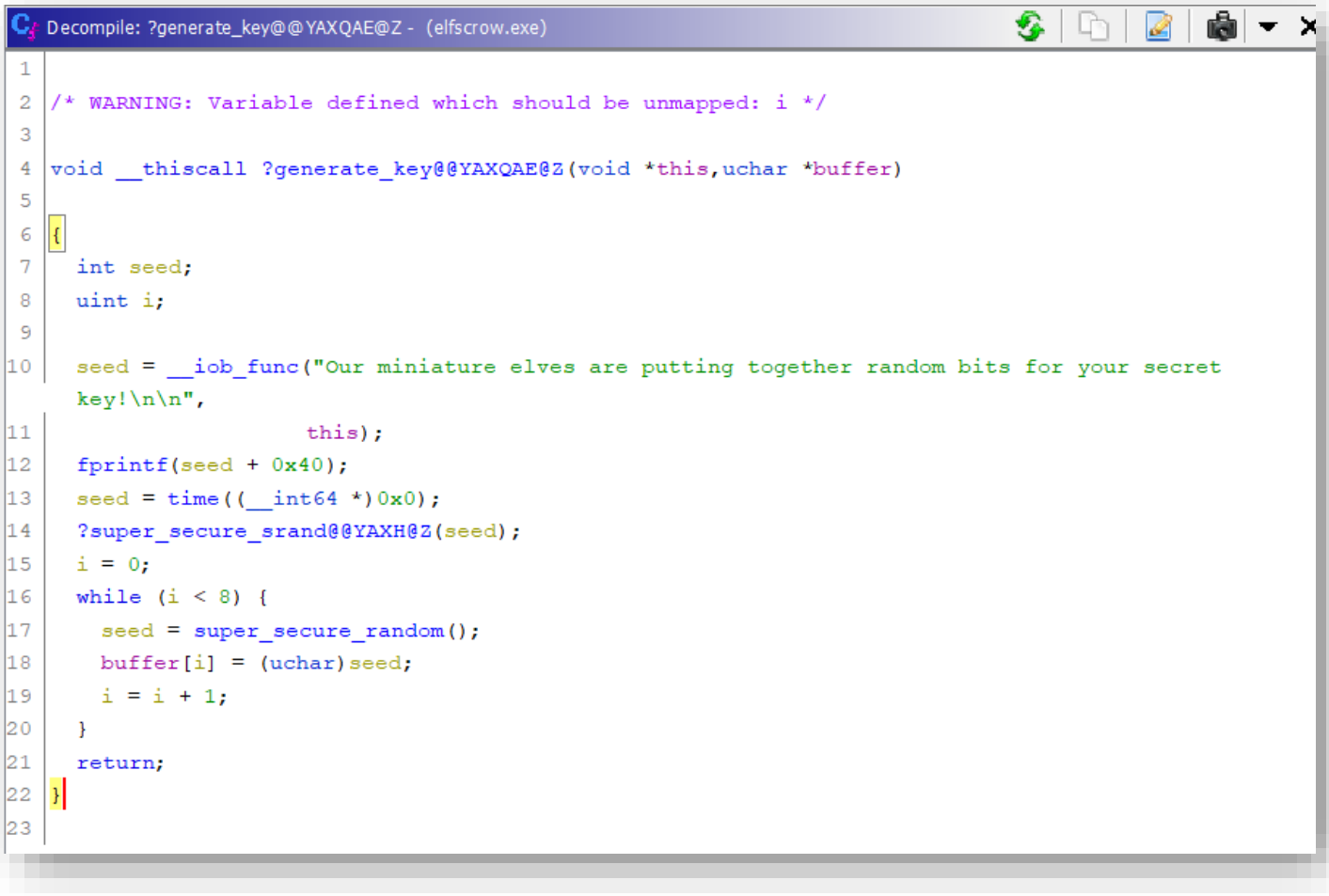


Figure 168 - Recover Cleartext Screenshot

As mentioned at the outset, Ron does a great job in the video and since the example he's using is very close to what we need, we take the skeleton code and amend it for our needs.

```ruby
require 'openssl'
require 'date'

KEYLENGTH=8

def generate_key(seed)
  key=""
  1.upto(KEYLENGTH) do
    seed = (214013 * seed + 2531011)
    key += (((seed >> 16) & 0x7fff_ffff) & 0x0FF).chr
  end
  return key
end

def decrypt(data, key)
  c=OpenSSL::Cipher::DES.new('cbc')
  c.decrypt
  c.key=key
  return(c.update(data) + c.final())
end

file = File.open("file.enc")
contents = file.read
file.close
seed=1575658800 # Friday, 6 December 2019 19:00:00

puts "[+] Processing..."
for i in 1..7200 do
  key=generate_key(seed)
  begin
    mydata=decrypt(contents,key)

    name= seed.to_s  + ".pdf"
    File.write(name, mydata)
    result = IO.binread(name, 4).unpack("H*").first
    valid_pdf = result == '25504446'
    if valid_pdf
        puts "[+] Found the fella: " + seed.to_s  + ".pdf which was created at " + DateTime.strptime(seed.to_s,'%s')
        break
    else
        File.delete(name)
    end
  rescue
    #puts "[+] Ach, bugger!"
  end
  seed=seed+1
end
```

Figure 169 - Recover Cleartext Document

Executing the above will eventually give us the screen below and the file is produced. Based on the timestamp, we can see the encryption took place at Friday, 6 December 2019 20:20:50.

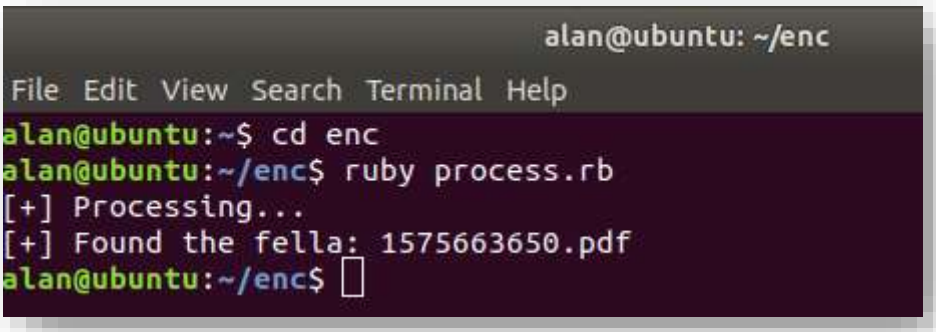


Figure 170 - Recover Cleartext Screenshot

Opening the file reveals an 8 page PDF document which can be bets described using the following screenshot which also gives the answer to the objective.

ELF UNIVERSITY

Ille te videt dum dormit

Research Labs

Super Sled-O-Matic

Machine Learning Sleigh Route Finder

QUICK-START GUIDE

SUPER SANTA SECRET:

DO NOT REDISTRIBUTE

Figure 171 - Recover Cleartext Screenshot

Open the Sleigh Shop Door

Synopsis

Visit Shinny Upatree in the Student Union and help solve their problem. What is written on the paper you retrieve for Shinny?

For hints on achieving this objective, please visit the Student Union and talk with Kent Tinseltooth.

We are presented with 10 locks that need a code in order to be unlocked that should need nothing more than the developer tools without our browser of choice.

Hint

- Chrome Dev Tools - https://developers.google.com/web/tools/chrome-devtools
- Safari Dev Tools - https://developer.apple.com/safari/tools/
- Edge Dev Tools - https://docs.microsoft.com/en-us/microsoft-edge/devtools-guide/console
- Firefox Dev Tools - https://developer.mozilla.org/en-US/docs/Tools

We also have some additional hints from completing the trail game

1. When I'm down, my F12 key consoles me
2. Reminds me of the transition to the paperless naughty/nice list...
3. Like a present stuck in the chimney! It got sent...
4. We keep that next to the cookie jar
5. My title is toy maker the combination is 12345
6. Are we making hologram elf trading cards this year?
7. If we are, we should have a few fonts to choose from
8. The parents of spoiled kids go on the naughty list...
9. Some toys have to be forced active
10. Sometimes when I'm working, I slide my hat to the left and move odd things onto my scalp!

Solution

The crate isn't visible at first but looking at the source code reveals it's location.

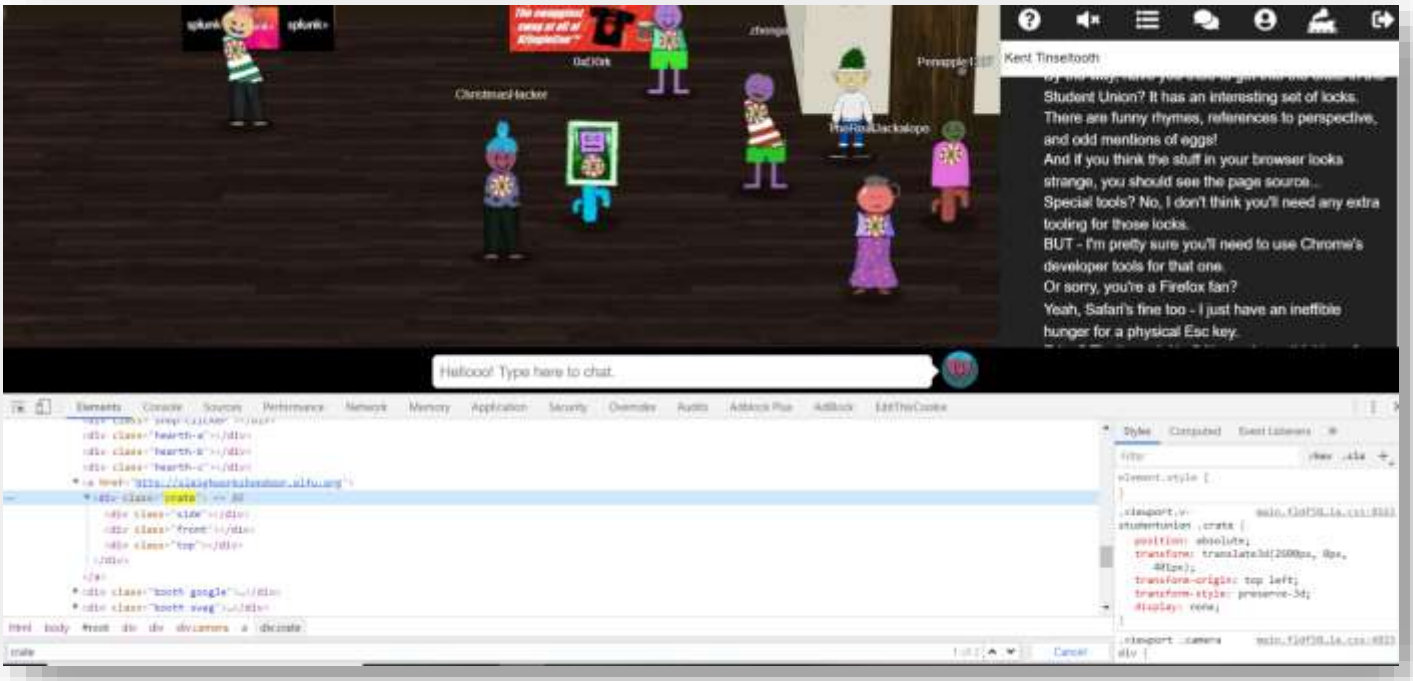


Figure 172 - Crate Screenshot

Question 1

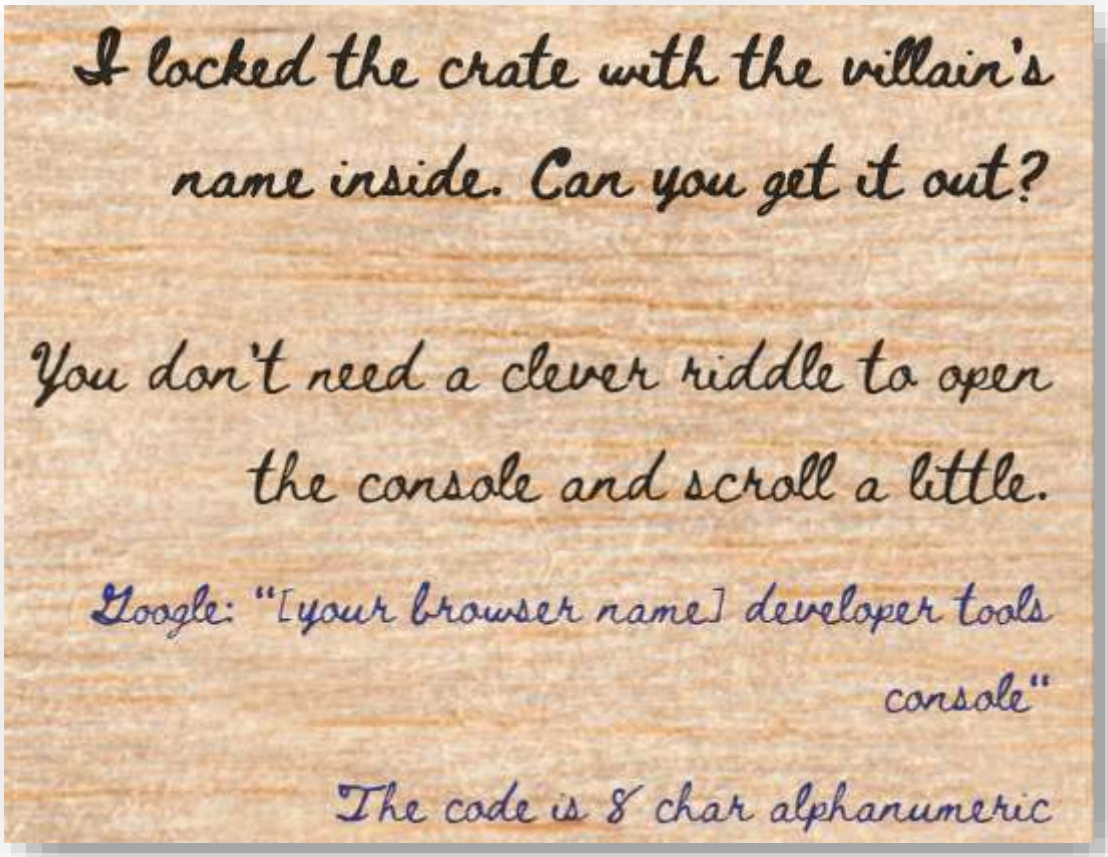


Figure 173 - Crate Screenshot

An easy one to start off – going to the console and scrolling up reveals the first code. At this point it's worth noting that on each refresh, the values change so whatever codes are shown here, won't be replicated for others.

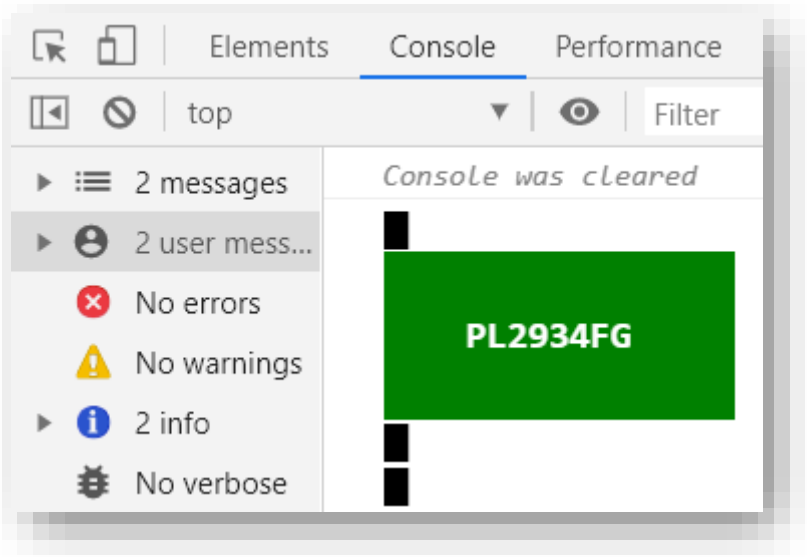


Figure 174 - Crate Screenshot

Question 2

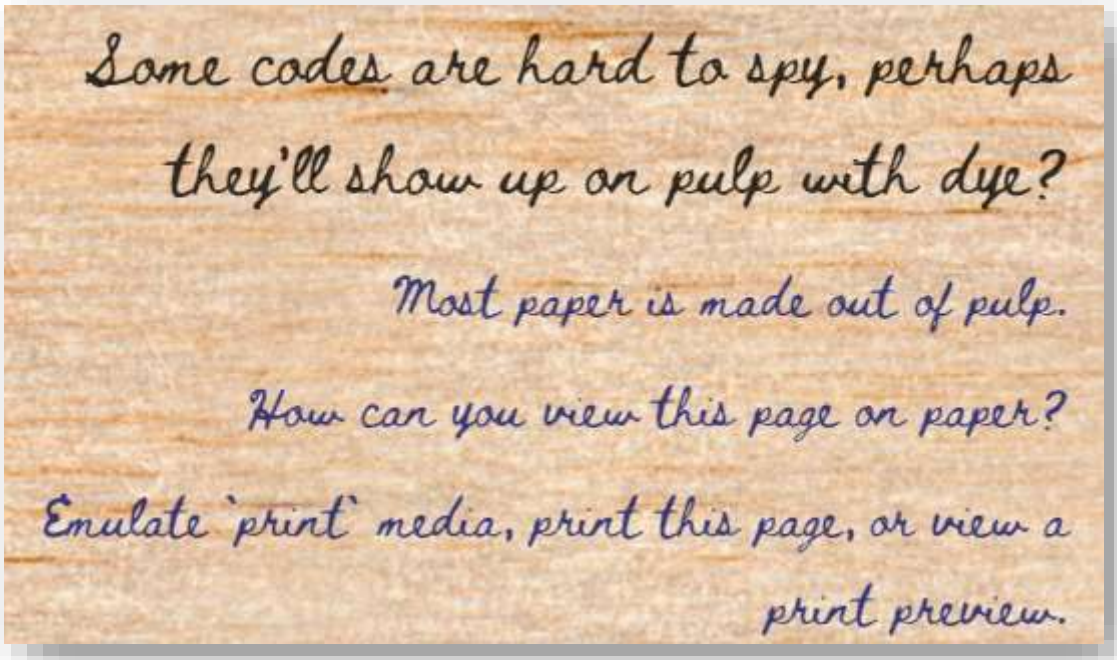


Figure 175 - Crate Screenshot

Another fairly easy one – doing a print preview shows us the next code.

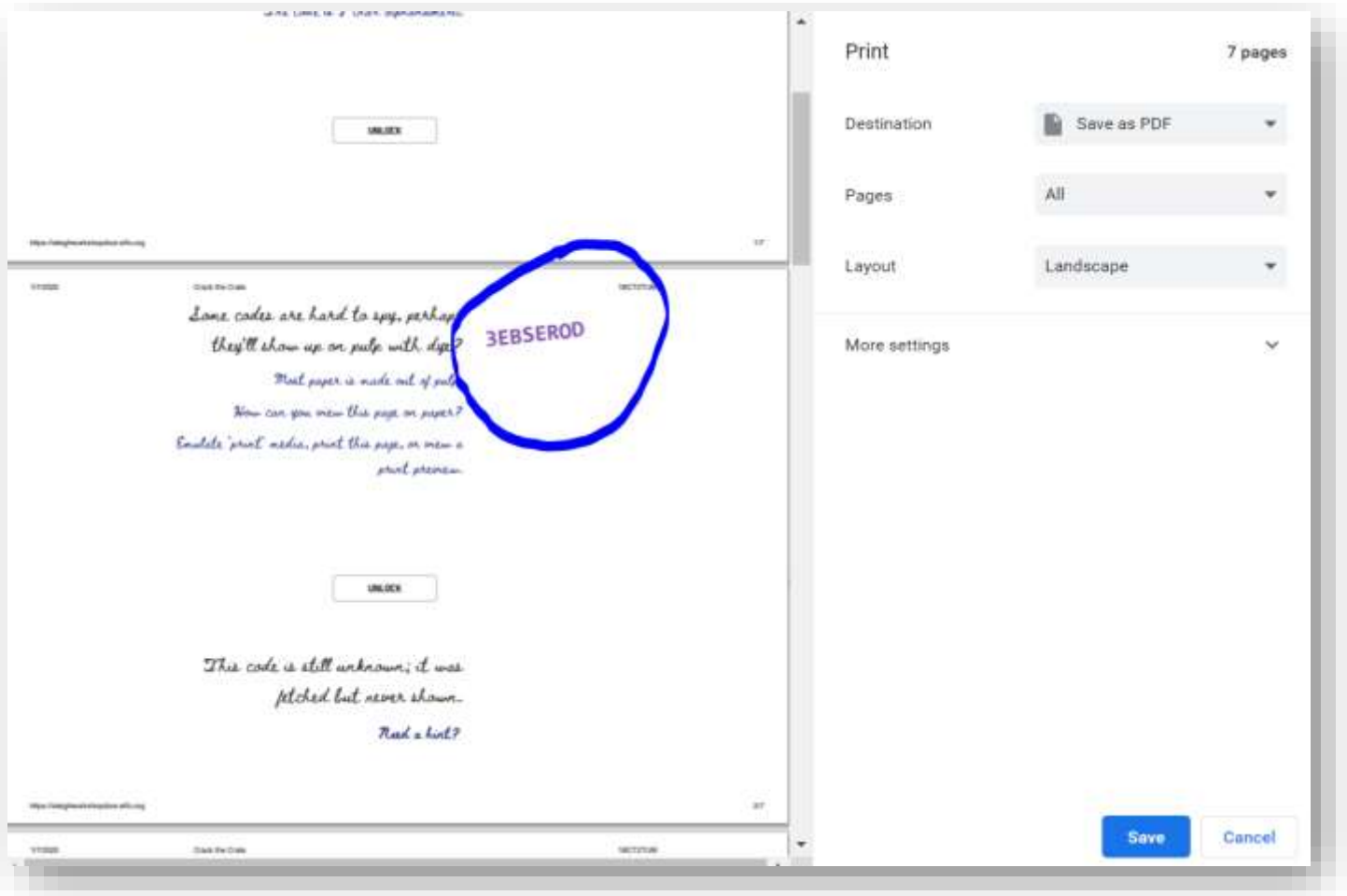


Figure 176 - Crate Screenshot

Question 3

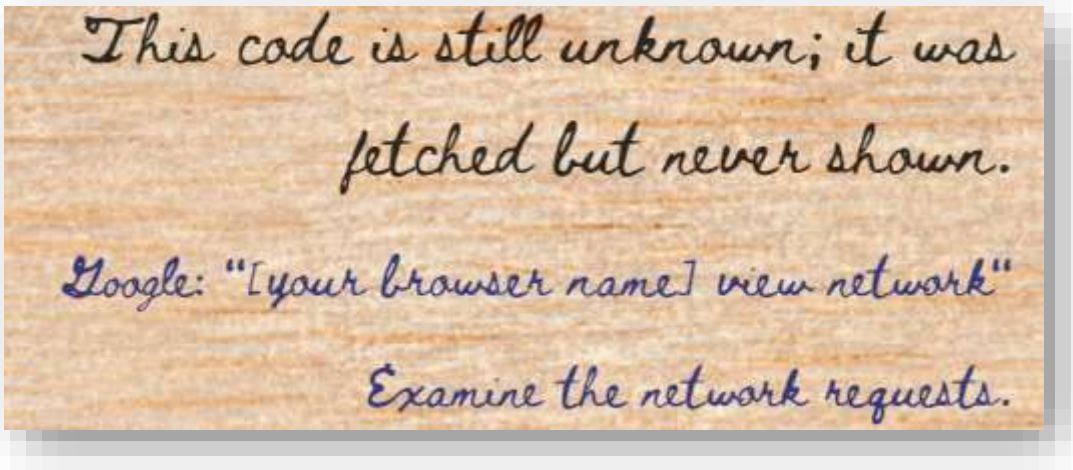


Figure 177 - Crate Screenshot

Doing what it says, examining the network requests, specifically XHR gives us the next answer.

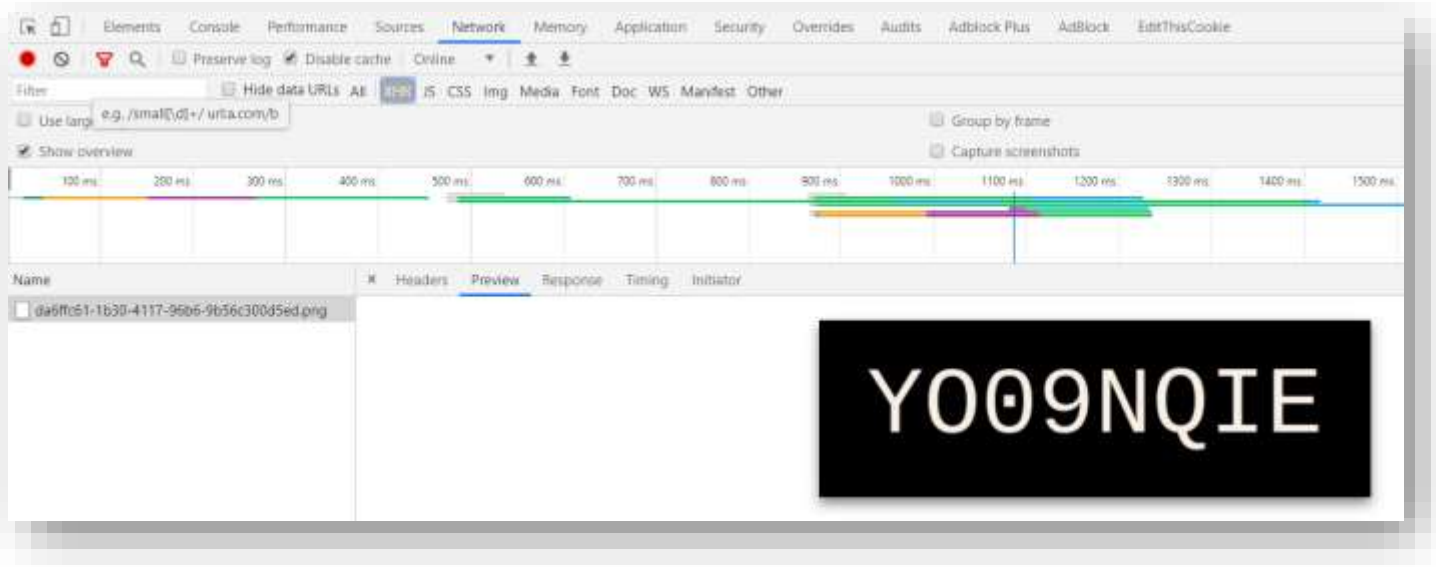


Figure 178 - Crate Screenshot

Question 4

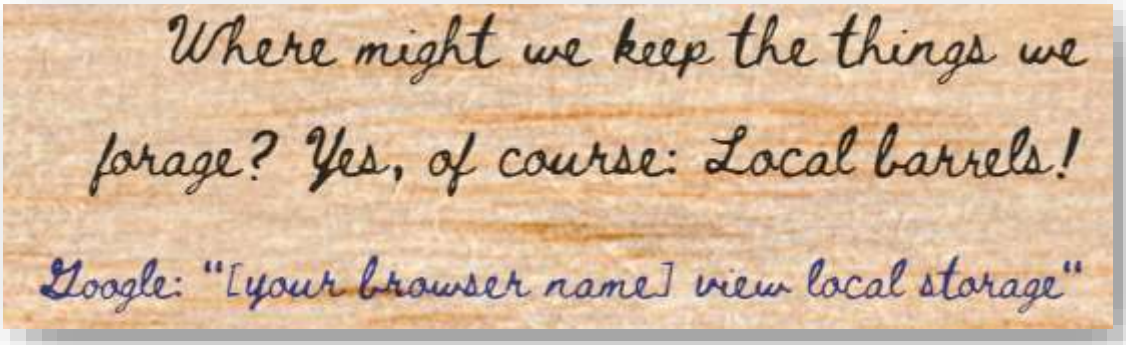


Figure 179 - Crate Screenshot

Question 5

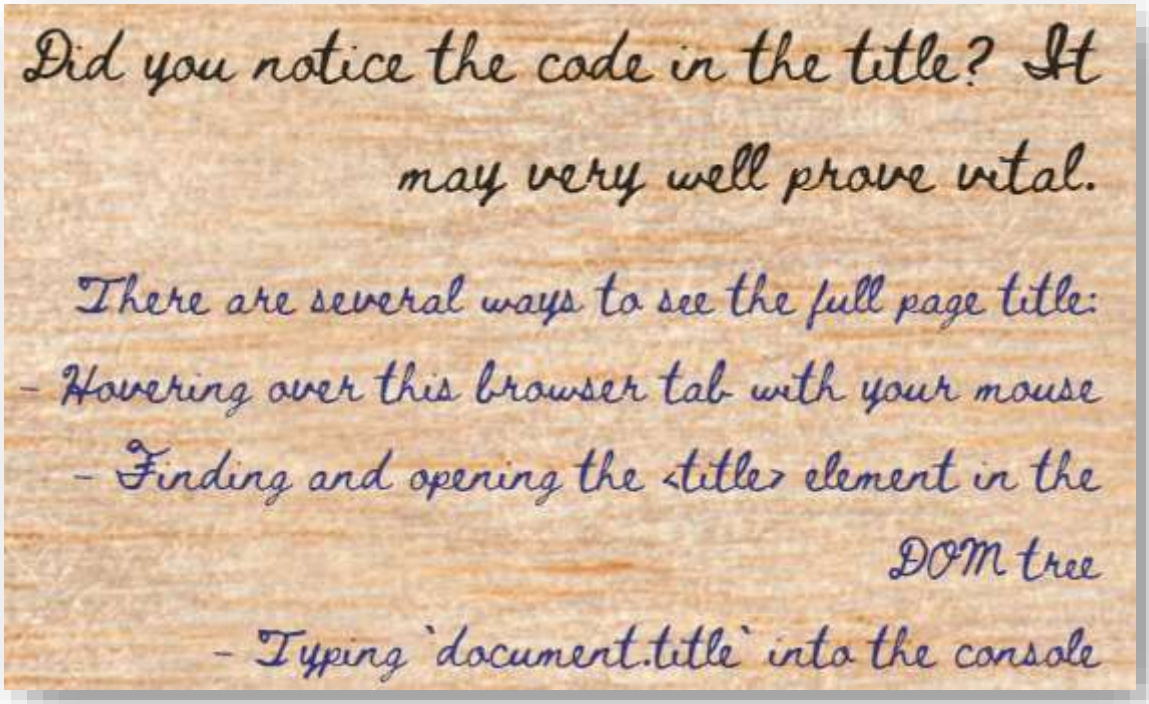


Figure 180 - Crate Screenshot

Another easy one

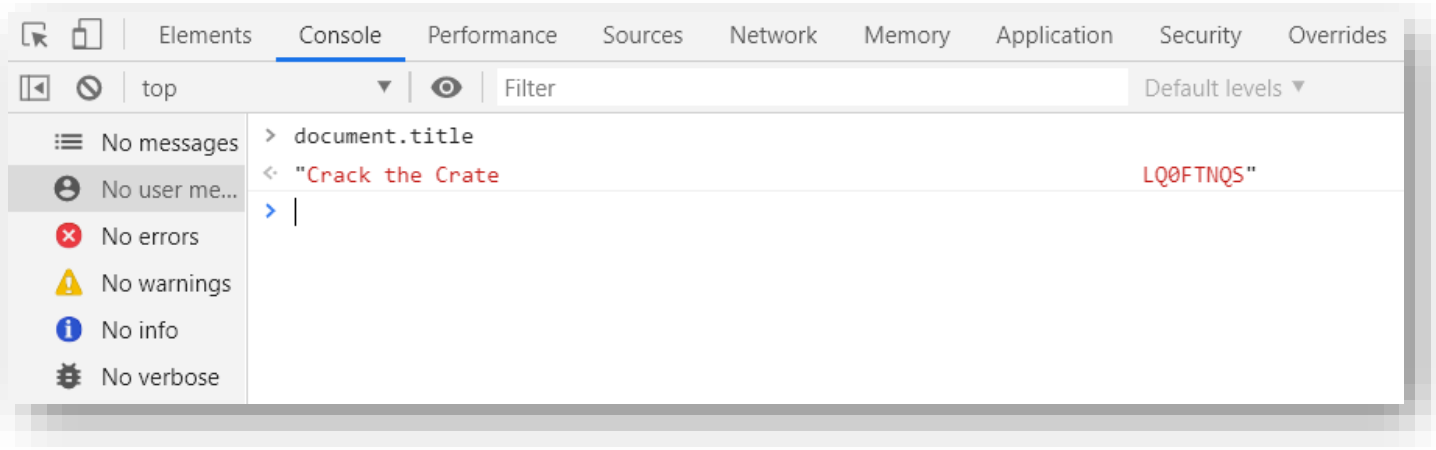


Figure 181 - Crate Screenshot

Question 6

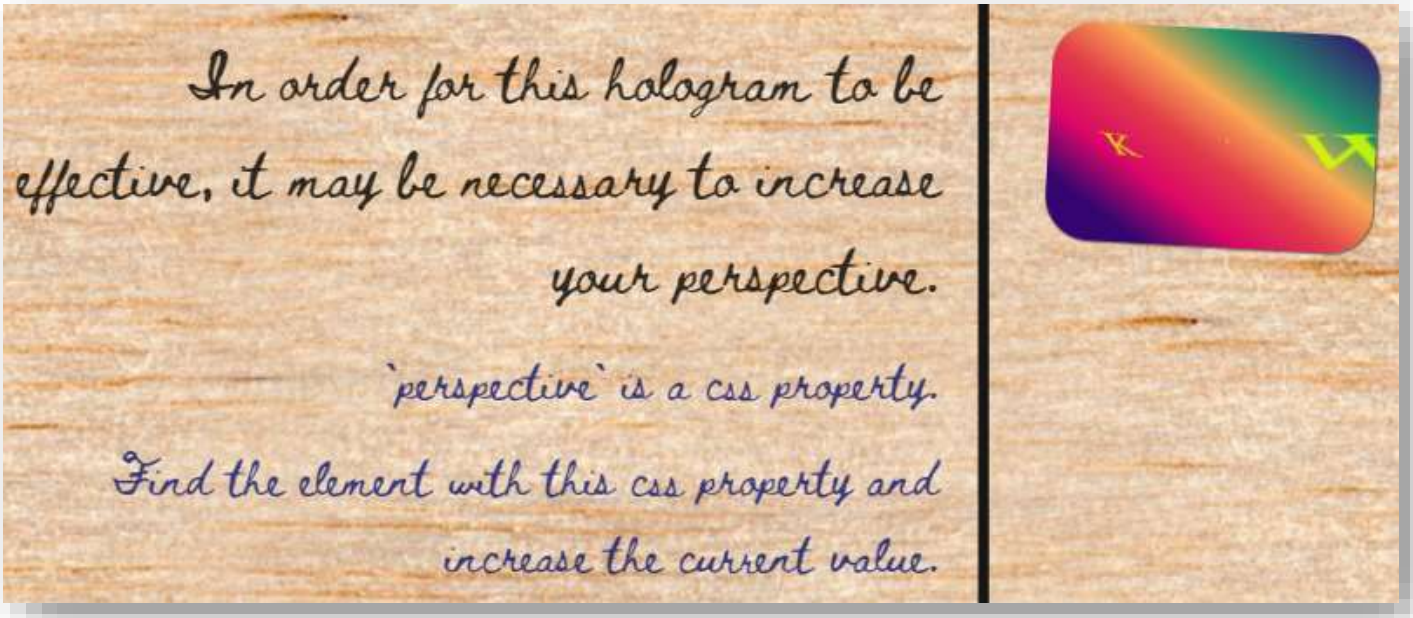


Figure 182 - Crate Screenshot

A bit more involved but not by much. Inspecting the hologram reveals that it has a `perspective`[63] CSS property attached to it.

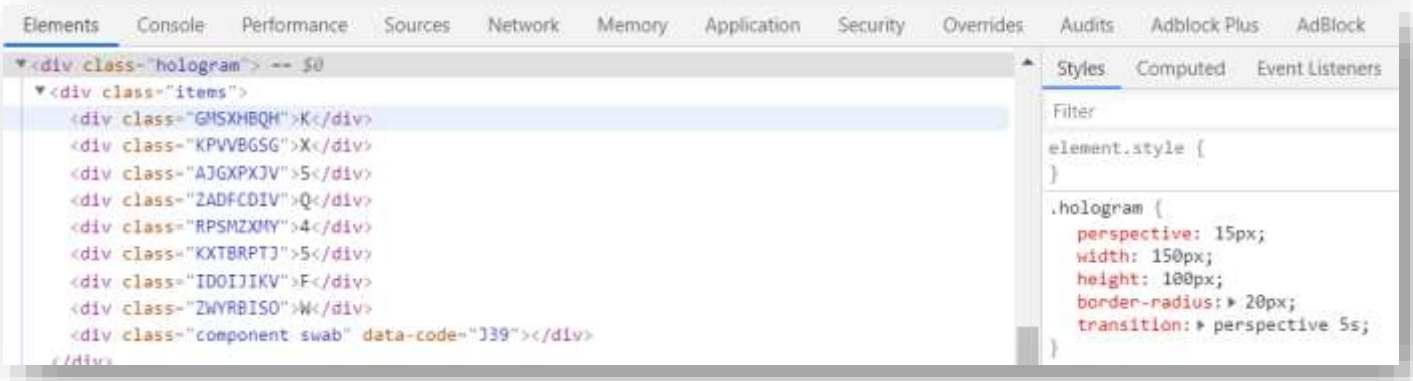


Figure 183 - Crate Screenshot

The clue says to increase the current value but removing the property works just as well. This can be done by just unchecking the box in the right-hand pane.

---

63 https://developer.mozilla.org/en-US/docs/Web/CSS/perspective

```
.hologram {
  perspective: 15px;
  width: 150px;
  height: 100px;
  border-radius: 20px;
  transition: perspective 5s;
}
```

Figure 184 - Crate Screenshot

This then reveals the code.

Figure 185 - Crate Screenshot

Question 7

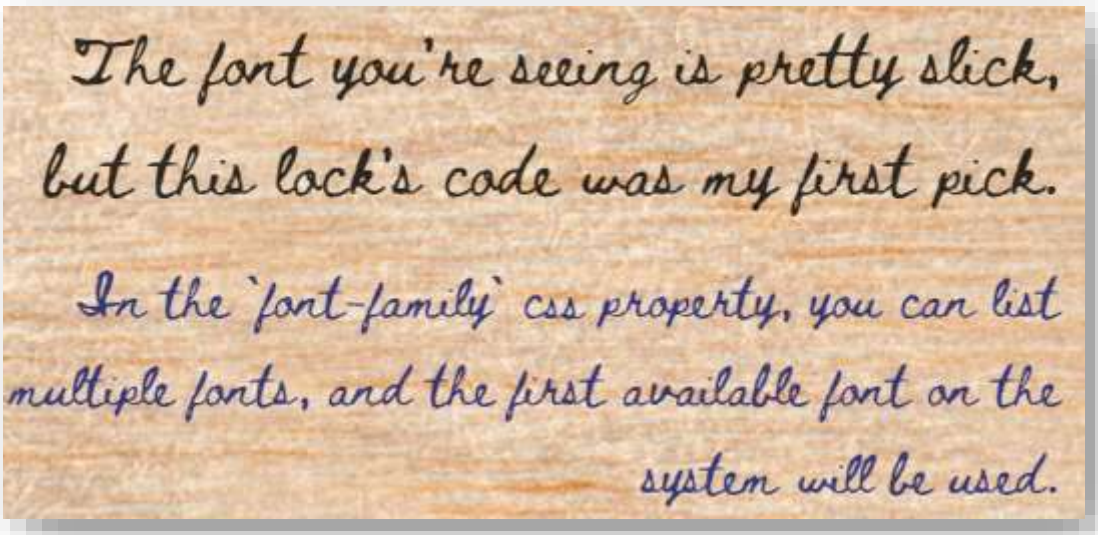


Figure 186 - Crate Screenshot

Another simple one – inspecting the font, specifically the `font-family`[64] property gives us the code for this lock.

[64] https://developer.mozilla.org/en-US/docs/Web/CSS/font-family

```
.instructions {
  font-family: 'ND5FVJ57', 'Beth Ellen', cursive;
}
```

Figure 187 - Crate Screenshot

Question 8

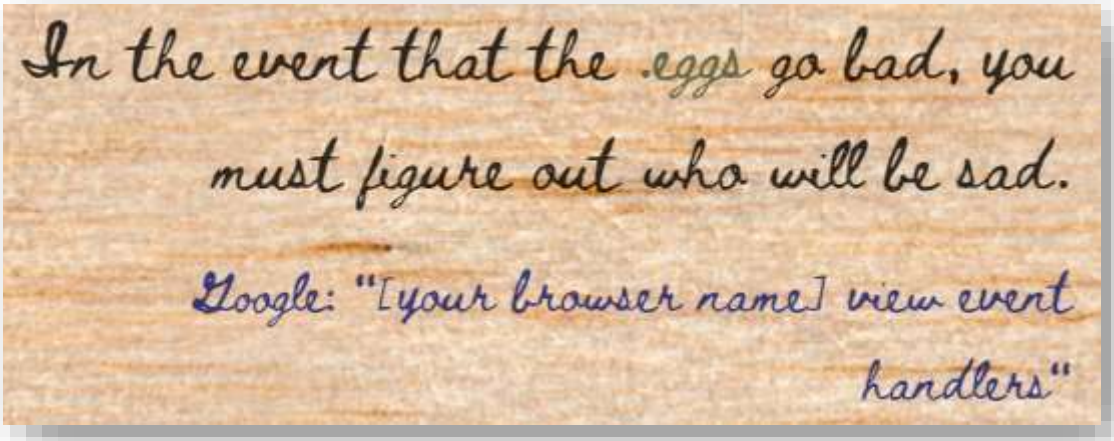


Figure 188 - Crate Screenshot

In CSS specifically, `.eggs` would indicate a class[65] of `eggs` has been given to one or more elements. Inspecting the page once again, we can see that there is only one instance.

```
▼<div class="instructions">
    "In the event that the "
    <span class="eggs">.eggs</span>
    " go bad, you must figure out who will be sad."
  </div>
```

Figure 189 - Crate Screenshot

With the specific element selected, going to the right-hand pane and selecting event listeners, we see a synonym of “go bad” in “spoil”, expanding the tree gives us the code for this lock.

---

[65] https://developer.mozilla.org/en-US/docs/Web/CSS/Class_selectors

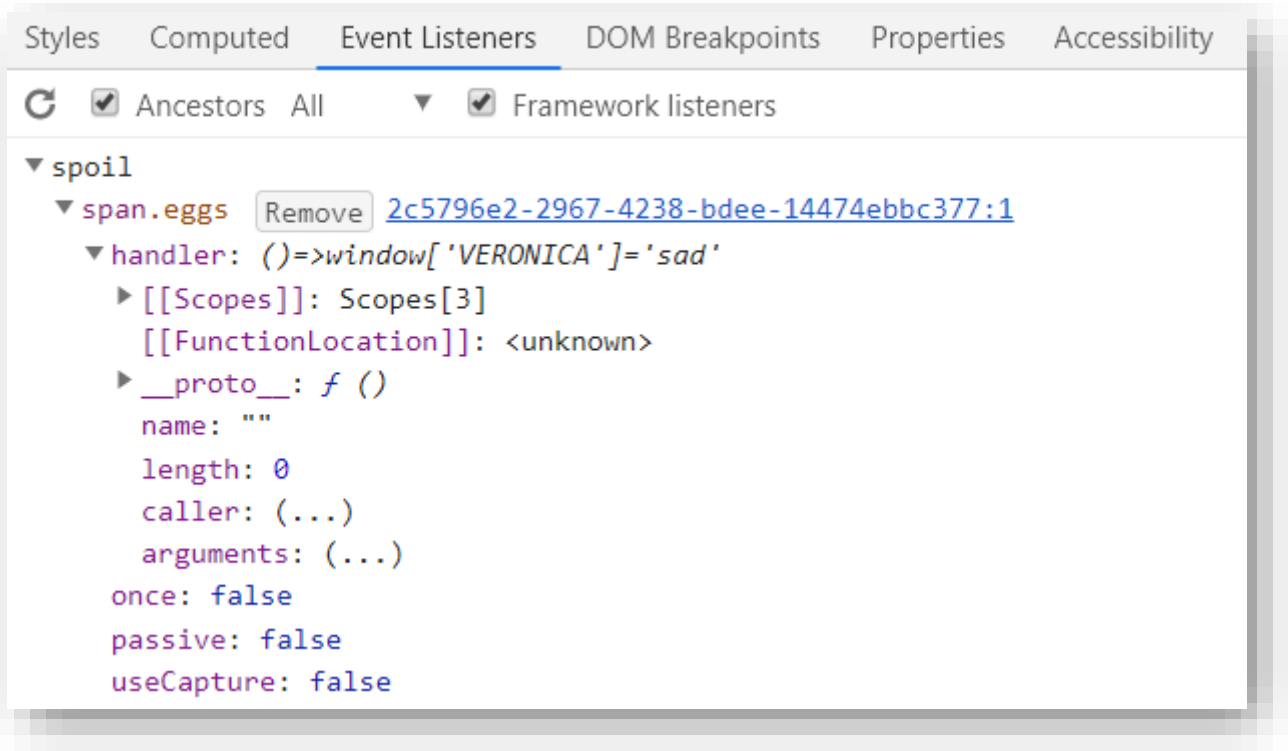


Figure 190 - Crate Screenshot

Question 9

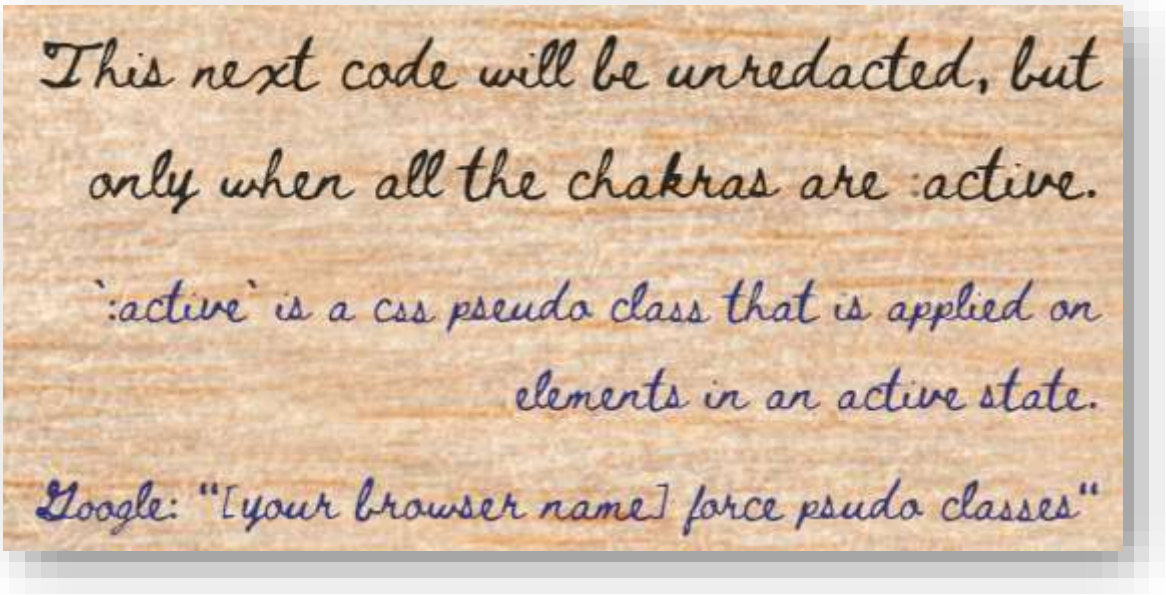


Figure 191 - Crate Screenshot

Inspecting the page shows up several elements with the class `chakra` attached to them.

```
▼<div class="instructions">
    "This "
    <span class="chakra">next</span> == $0
    " code will be "
    <span class="chakra">unredacted</span>
    ", but "
    <span class="chakra">only</span>
    <span class="chakra">when</span>
    " all the "
    <span class="chakra">chakras</span>
    " are "
    <span class="subtle">:</span>
    "active."
  </div>
```

Figure 192 - Crate Screenshot

Using the right-hand pane and the styles tab, we can see what CSS styles are being associated with the class.

```
241 span.chakra {
242   position: relative;
243 }
244
245 span.chakra:active:after {
246   content: '';
247   position: absolute;
248   font-family: monospace;
249   font-weight: bold;
250   color: #bb0000;
251   font-size: 1.5em;
252   top: -12px;
253 }
254
255 span.chakra:nth-child(1):active:after {
256   content: 'L7';
257 }
258 span.chakra:nth-child(2):active:after {
259   content: 'U4';
260 }
261 span.chakra:nth-child(3):active:after {
262   content: 'I';
263 }
264 span.chakra:nth-child(4):active:after {
265   content: 'PI';
266 }
267 span.chakra:nth-child(5):active:after {
268   content: '2';
269 }
```

Figure 193 - Crate Screenshot

The code for the lock can be seen here but there is another way to do it. In a previous screenshot, simple right-hand clicking on the element in question and selecting **Force State > :active**, will, as should be clear, force the active state[66] upon the element, thus initiating the relevant CSS style.

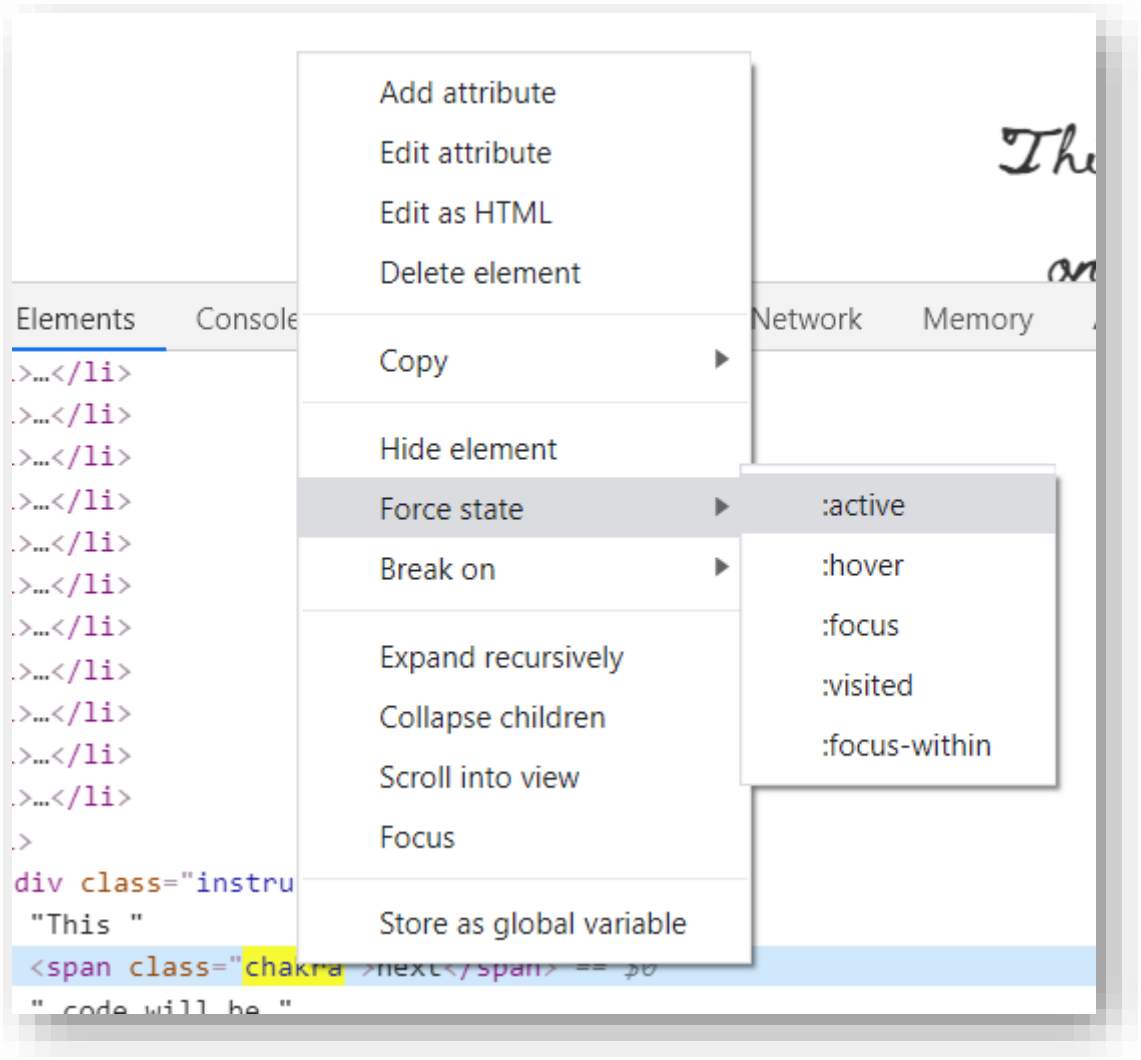


Figure 194 - Crate Screenshot

Doing this on all the relevant elements will again, show the code for this lock.

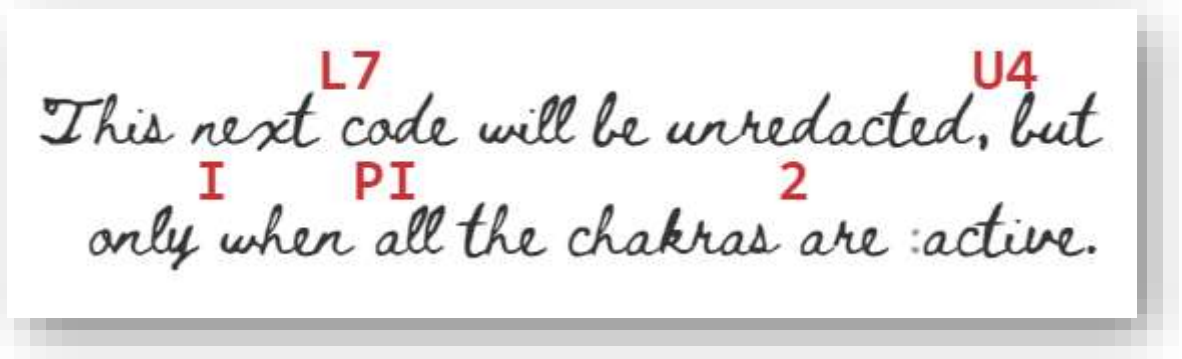


Figure 195 - Crate Screenshot

[66] https://developer.mozilla.org/en-US/docs/Web/CSS/:active

Question 10

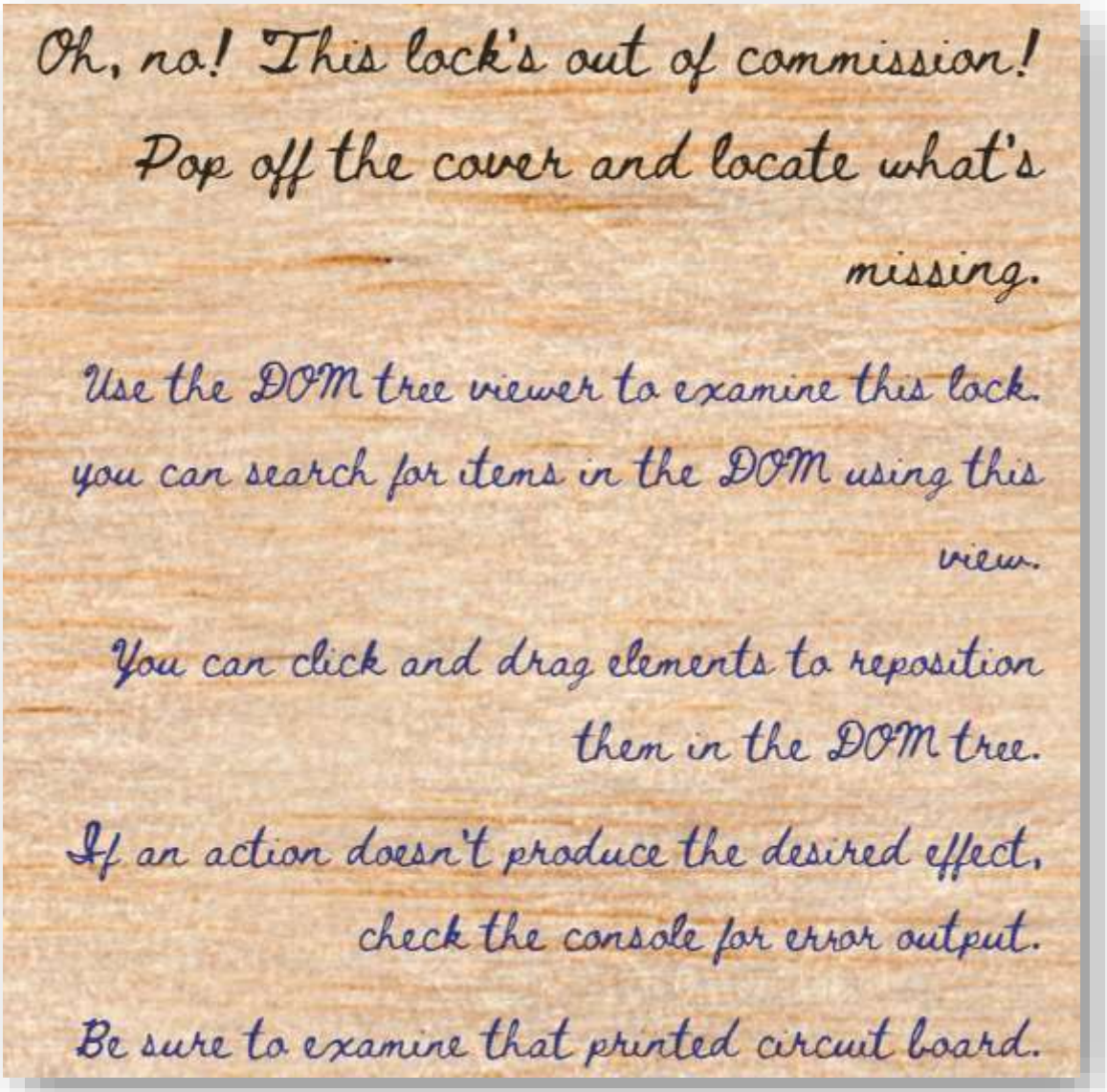


Figure 196 - Crate Screenshot

Inspecting the page gives us only one reference to cover.

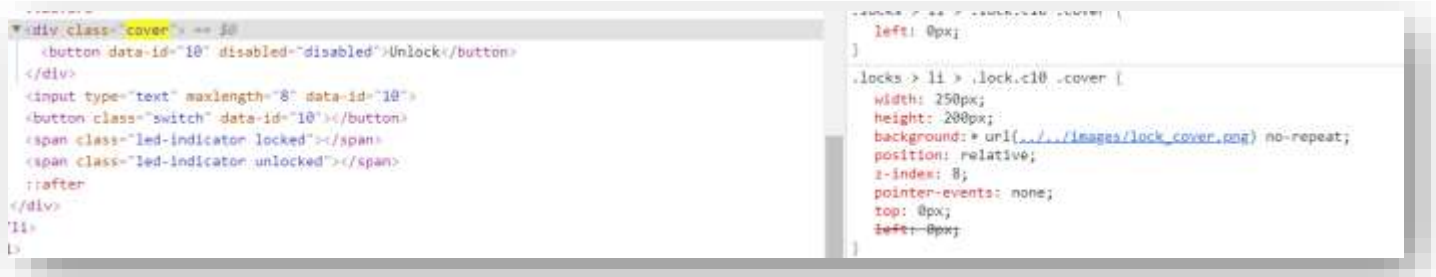


Figure 197 - Crate Screenshot

The clue says to pop the cover which we do a number of ways but easiest here would be to do what was done before and simply uncheck the `background`[67] property in the right-hand pane. Doing so, reveals a circuit board.

Figure 198 - Crate Screenshot

The last clue says to examine the circuit board and in doing so, we can see an 8 digit code in the bottom right hand corner.

Figure 199 - Crate Screenshot

However, adding **KD29XJ37** into the lock doesn't pop it open. Instead, we get errors in the console that the hints point towards.

Figure 200 - Crate Screenshot

[67] https://developer.mozilla.org/en-US/docs/Web/CSS/background

The link takes us the place in the code where the error gets thrown but unfortunately it's not easily readable.

Figure 201 - Crate Screenshot

Fortunately, we can prettify the code using the **Pretty print** option at the bottom.

```
try {
    const _0x271ae8 = document[_0xedf7('0x2e')](_0xedf7('0x5d'));
    if (!_0x271ae8)
        throw Error('\x4d\x69\x73\x73\x69\x6e\x67\x20\x6d\x61\x63\x61\x72\x6f\x6e\x69\x21');
    _0x271ae8[_0xedf7('0x2b')][_0xedf7('0x5e')]['\x76\x61\x6c\x75\x65'];
    const _0x19db5d = document['\x71\x75\x65\x72\x79\x53\x65\x6c\x65\x63\x74\x6f\x72'](_0xedf7('0x5f'));
    if (!_0x19db5d)
        throw Error('\x4d\x69\x73\x73\x69\x6e\x67\x20\x63\x6f\x74\x74\x6f\x6e\x20\x73\x77\x61\x62\x21');
    _0x19db5d['\x61\x74\x74\x72\x69\x62\x75\x74\x65\x73'][_0xedf7('0x5e')]['\x76\x61\x6c\x75\x65'];
    const _0x2caef2 = document[_0xedf7('0x2e')](_0xedf7('0x60'));
    if (!_0x2caef2)
        throw Error(_0xedf7('0x61'));
    _0x2caef2[_0xedf7('0x2b')][_0xedf7('0x5e')][_0xedf7('0x55')];
    _0x1acb20(_0x4664e7, {
        '\x69\x64': _0x4664e7,
        '\x63\x6f\x64\x65': _0xcb5497[_0xedf7('0x55')]
    });
} catch (_0x3c48f6) {
    console[_0xedf7('0x42')](_0x3c48f6);
}
```

Figure 202 - Crate Screenshot

A little more readable but the code is obfuscated. We can still work with this however.

`_0xedf7('0x2e')` will contain a value and we can view values, similar to how we got do`cument.title` by just writing it to the console as follows:

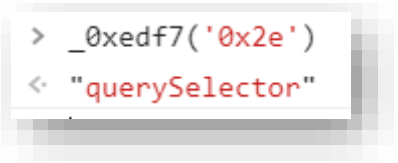


Figure 203 - Crate Screenshot

This is also repeated for the 3rd `if` statement. The value is querySelector[68]. Moving on to the next part, we get the final piece of the command.

Figure 204 - Crate Screenshot

querySelector is looking for one or more selectors to match. The selector in this example is `.locks > li > .lock.c10 > .component.macaroni`. The error is thrown because this selector doesn't exist.

Going onto the 3rd `if` statement which has the same start, we can see it's looking for another selector.

Figure 205 - Crate Screenshot

Finally, the middle one is formatted slightly differently as it has a load of hex values thrown in. It actually has a similar format to the other two so it's safe to presume it'll be `querySelector` as well and when decoded it, sure enough, it decodes to `querySelector`.

Figure 206 - Crate Screenshot

What we now need to do is ensure there exists 3 elements underneath the element with a class of `lock` and `c10` that all have class of `component` and `macaroni`, `gnome` and `swab` respectively.

Inspecting the code one last time shows all elements exist already.

---

[68] https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector

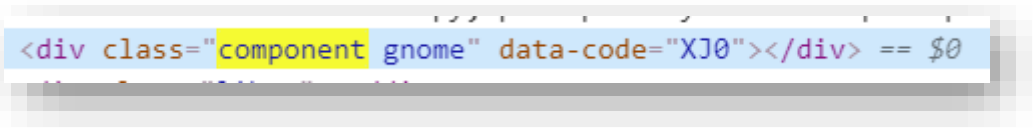


Figure 207 - Create Screenshot

Now it's just a matter of ensuring they exist in the proper place.

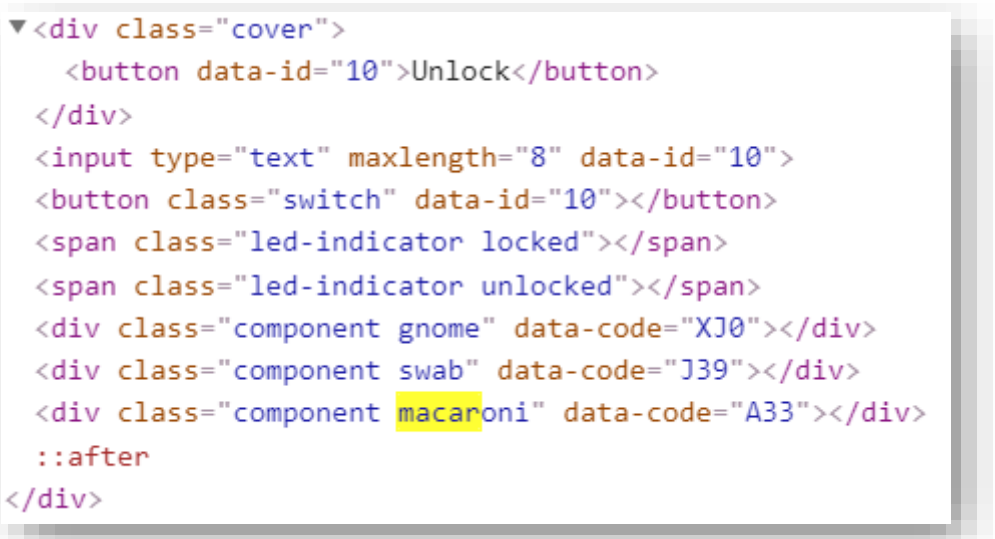


Figure 208 - Crate Screenshot

This enables us to proceed and complete the challenge. In completing it, we get the following console messages

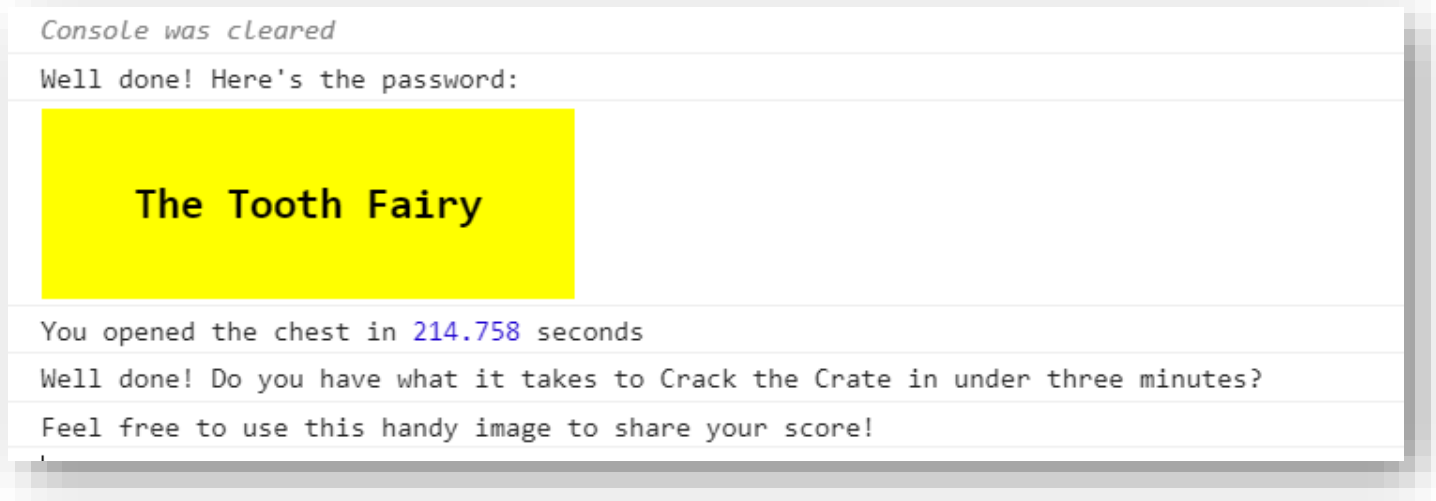


Figure 209 - Crate Screenshot

In fact, we can do better than 3 minutes by some margin – the code and explanation of which are in the appendix but this is what you see when you do get it in a quick time.

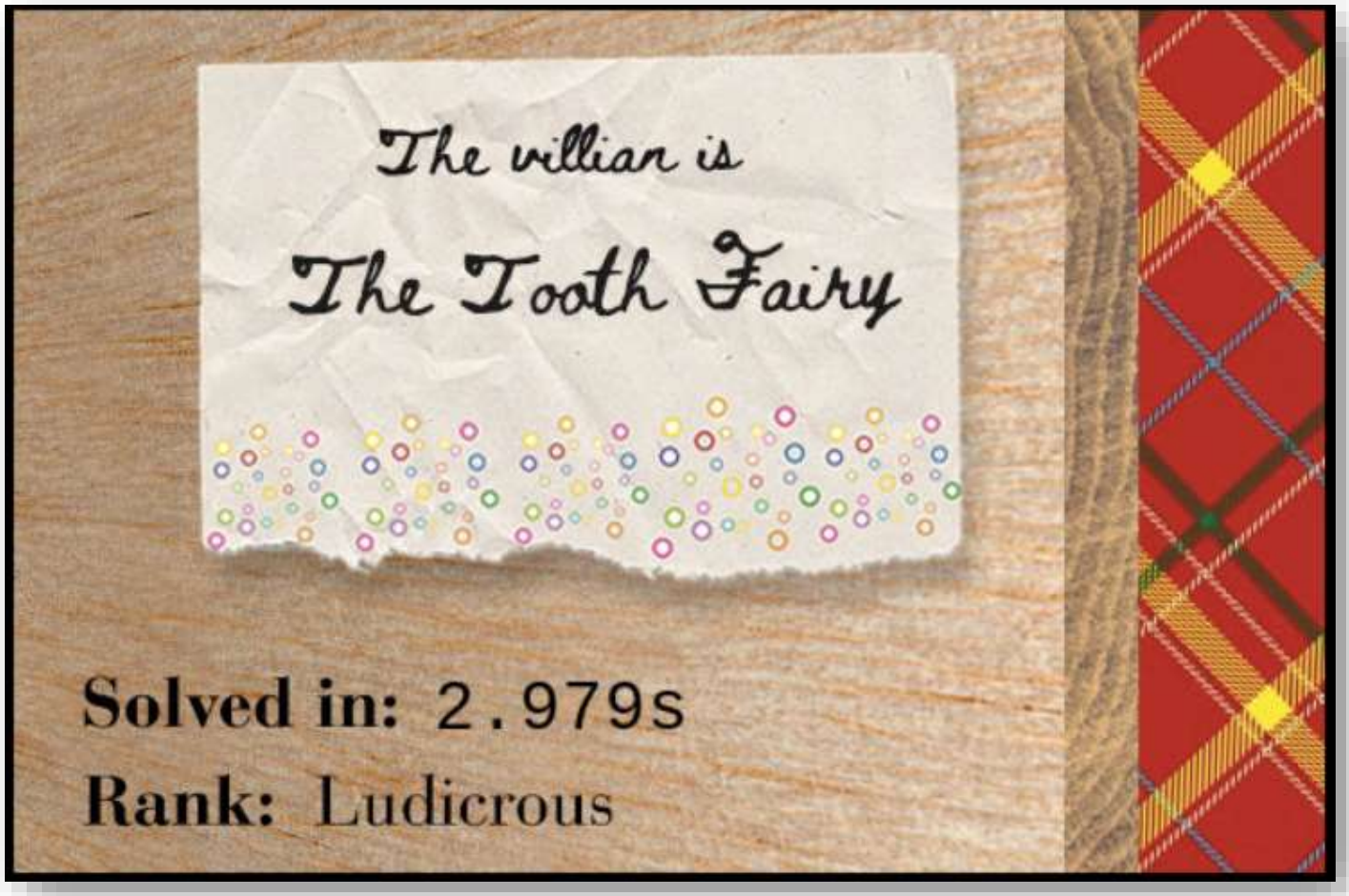


Figure 210 - Crate Screenshot

Filter Out Poisoned Sources of Weather Data

Synopsis

Use the data supplied in the Zeek JSON logs to identify the IP addresses of attackers poisoning Santa's flight mapping software. Block the 100 offending sources of information to guide Santa's sleigh through the attack. Submit the Route ID ("RID") success value that you're given. For hints on achieving this objective, please visit the Sleigh Shop and talk with Wunorse Openslae.

https://downloads.elfu.org/http.log.gz

https://srf.elfu.org

Hint

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.

If you find a log entry that definitely looks bad, try pivoting off other unusual attributes in that entry to find more bad IPs.

The sleigh's machine learning device (SRF) needs most of the malicious IPs blocked in order to calculate a good route.

Try not to block many legitimate weather station IPs as that could also cause route calculation failure.

Remember, when looking at JSON data, jq is the tool for you!

Solution

Going to https://srf.elfu.org/presents us with a login page.

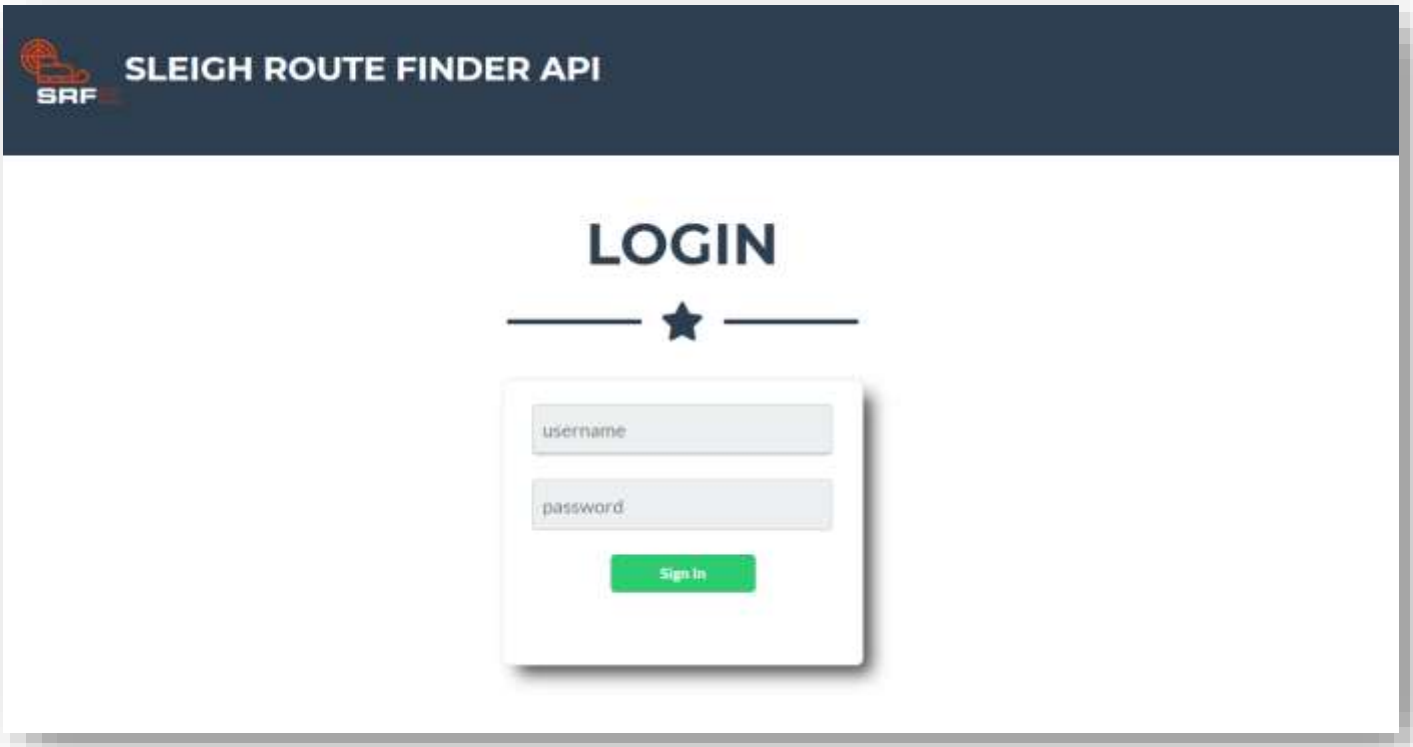


Figure 211 - Weather Data Screenshot

At this point there is no signs of credentials to use of weaknesses to bypass authentication. We review what we have got so far from all objectives.

3. SRF - Sleigh Route Finder Web API

The SRF Web API is started up on Super Sled-O-Matic device bootup and by default binds to 0.0.0.0:1225:

The default login credentials should be changed on startup and can be found in the readme in the ElfU Research Labs git repository.

The Sleigh Route Finder has a weather map showing Elf weather stations around the globe reporting their local conditions. The website also contains a simple IP Firewall for filtering out improper weather traffic from being ingested. These two features can be seen below:

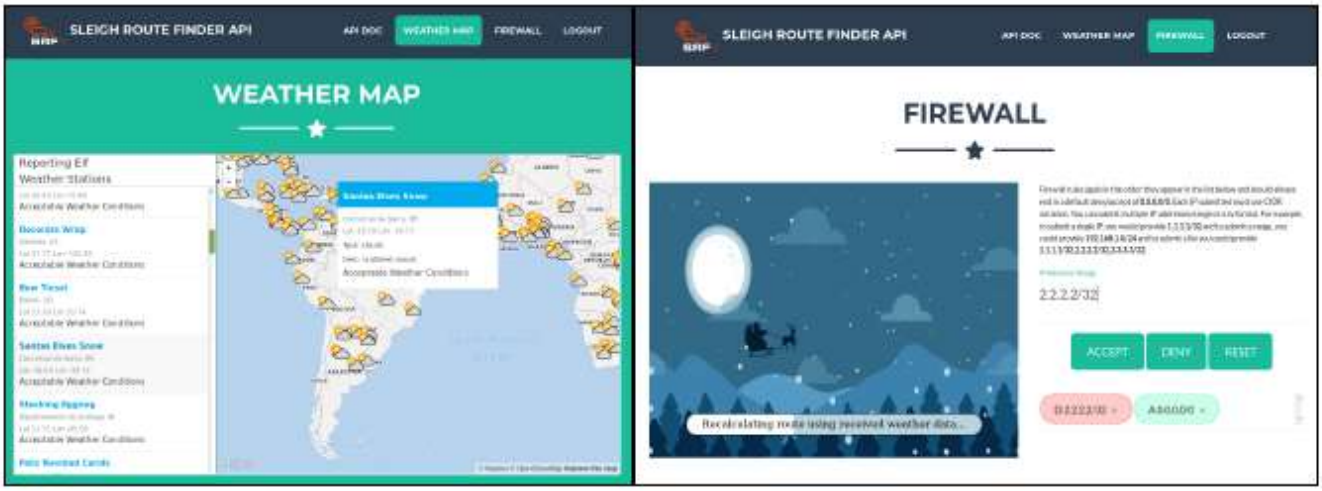


Figure 212 - Weather Data Screenshot

We find a clue in the document we got from decrypting the file in an earlier objective. Everything points to there being a readme file.

Going to https://srf.elfu.org/README.md (also in the downloaded logs for this objective) gives us these credentials.

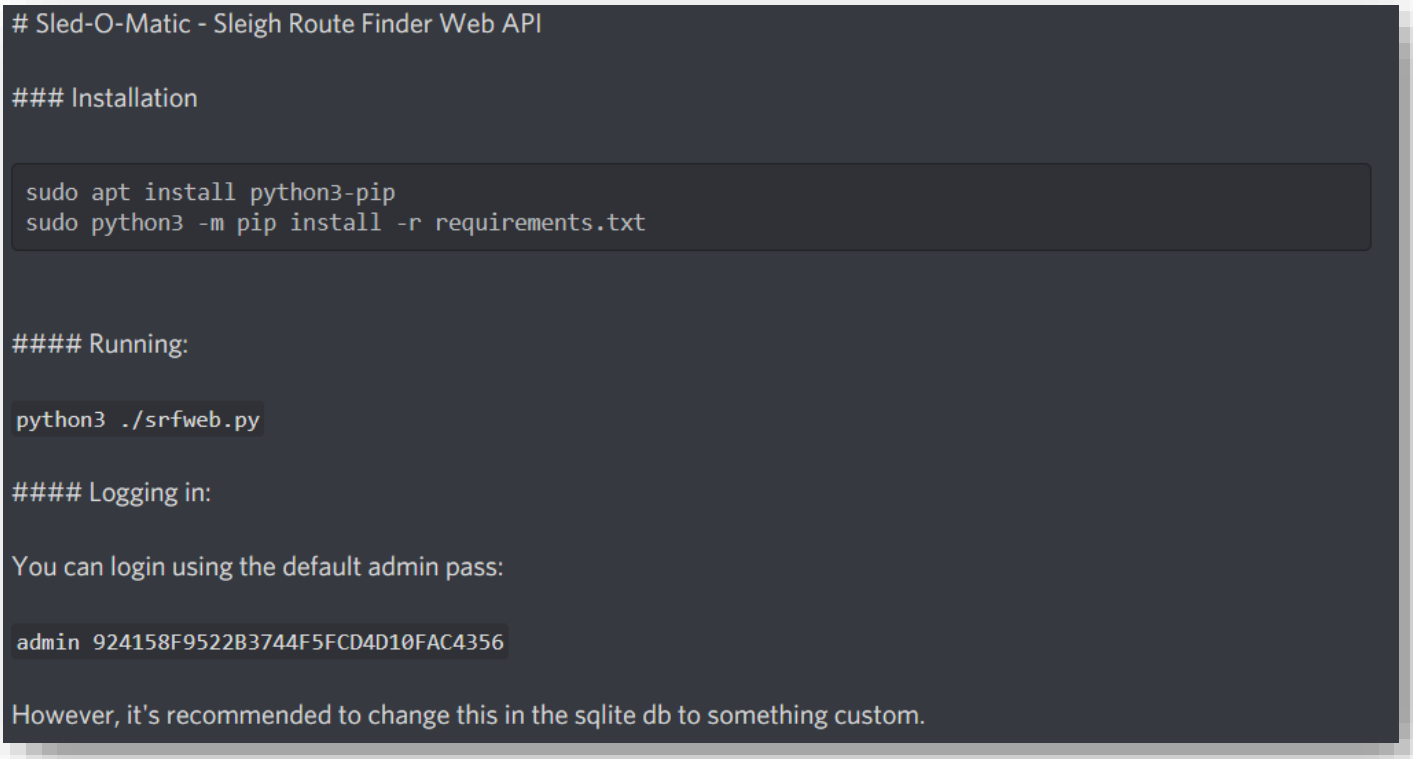


Figure 213 - Weather Data Screenshot

As per the document we used to find the clue to get us the credentials, the website has a section about API docs, the firewall section a weather map and an about us section.

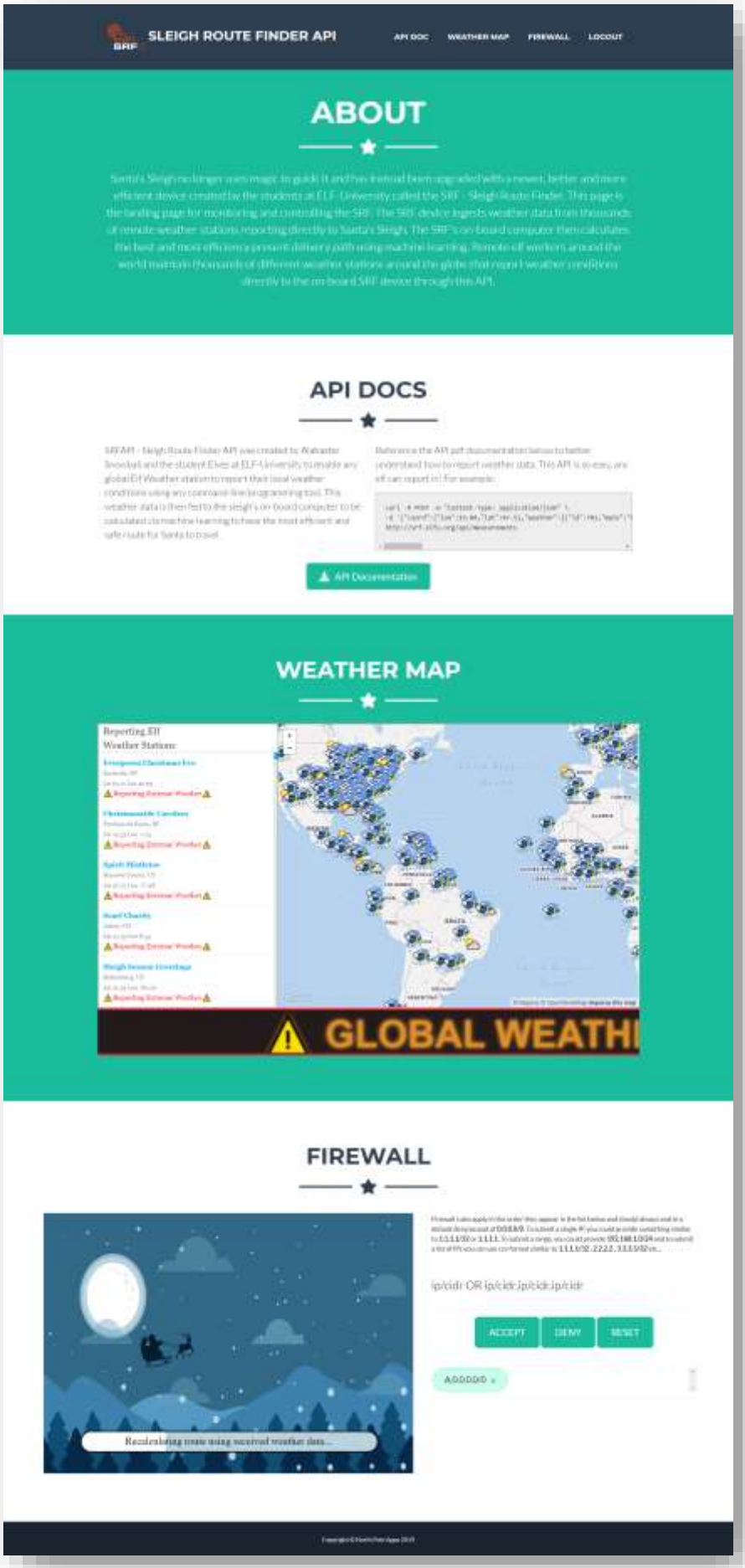


Figure 214 - Weather Data Screenshot

Using jq seems the obvious route to go down given the hints and the use of it to parse json previously. There are also utils out there that convert the JSON into a TSV format that can be consumed by RITA.

However, as is tradition[69], we opt for Excel. In order to do that, the JSON file is converted into CSV using a small bit of Python.

```python
import csv, json, sys
fileInput = "http.log"
fileOutput = "http.csv"
inputFile = open(fileInput) #open json file
outputFile = open(fileOutput, 'w') #load csv file
data = json.load(inputFile) #load json content
inputFile.close() #close the input file
output = csv.writer(outputFile) #create a csv.write
output.writerow(data[0].keys())  # header row
for row in data:
    output.writerow(row.values()) #values row
```

Figure 215 - Weather Data Screenshot

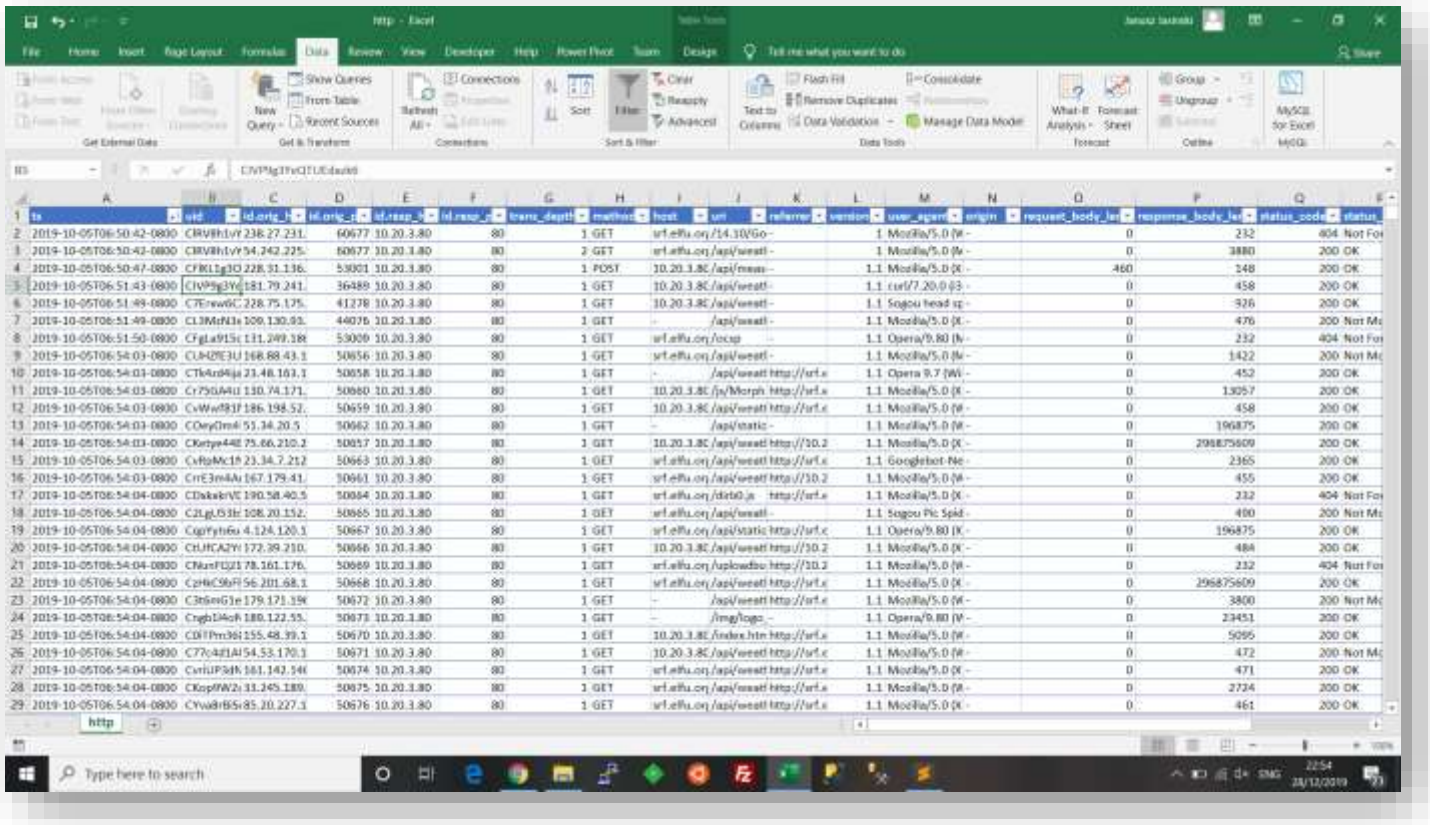
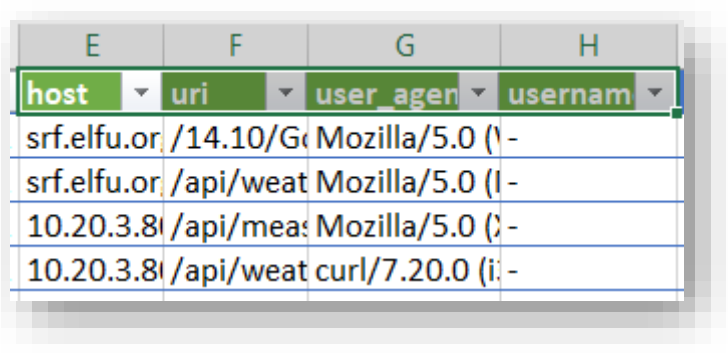


Figure 216 - Weather Data Screenshot

[69] We do a lot of the challenges and write-up at work where we have no access to 3rd party tools such as jq, linux, burp etc so we make do with what we can

Since the hints heavily suggested LFI[70], XSS[71], Shellshock[72] and SQLi[73], we can visually make the data more relevant by stripping out the Zeek[74] specific columns to just display what the end user would ultimately have control over. This leaves `host`, `uri`, `user_agent` and `username`.



E	F	G	H
host	uri	user_agen	usernam
srf.elfu.or	/14.10/G	Mozilla/5.0 (\	-
srf.elfu.or	/api/weat	Mozilla/5.0 (I	-
10.20.3.8	/api/mea	Mozilla/5.0 (	-
10.20.3.8	/api/weat	curl/7.20.0 (i	-

Figure 217 - Weather Data Screenshot

The next several screenshots show what sort of filters are being used to identify the different type of attacks being sent.

[70] https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion
[71] https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
[72] https://www.owasp.org/images/1/1b/Shellshock_-_Tudor_Enache.pdf
[73] https://www.owasp.org/index.php/SQL_Injection
[74] https://www.zeek.org/

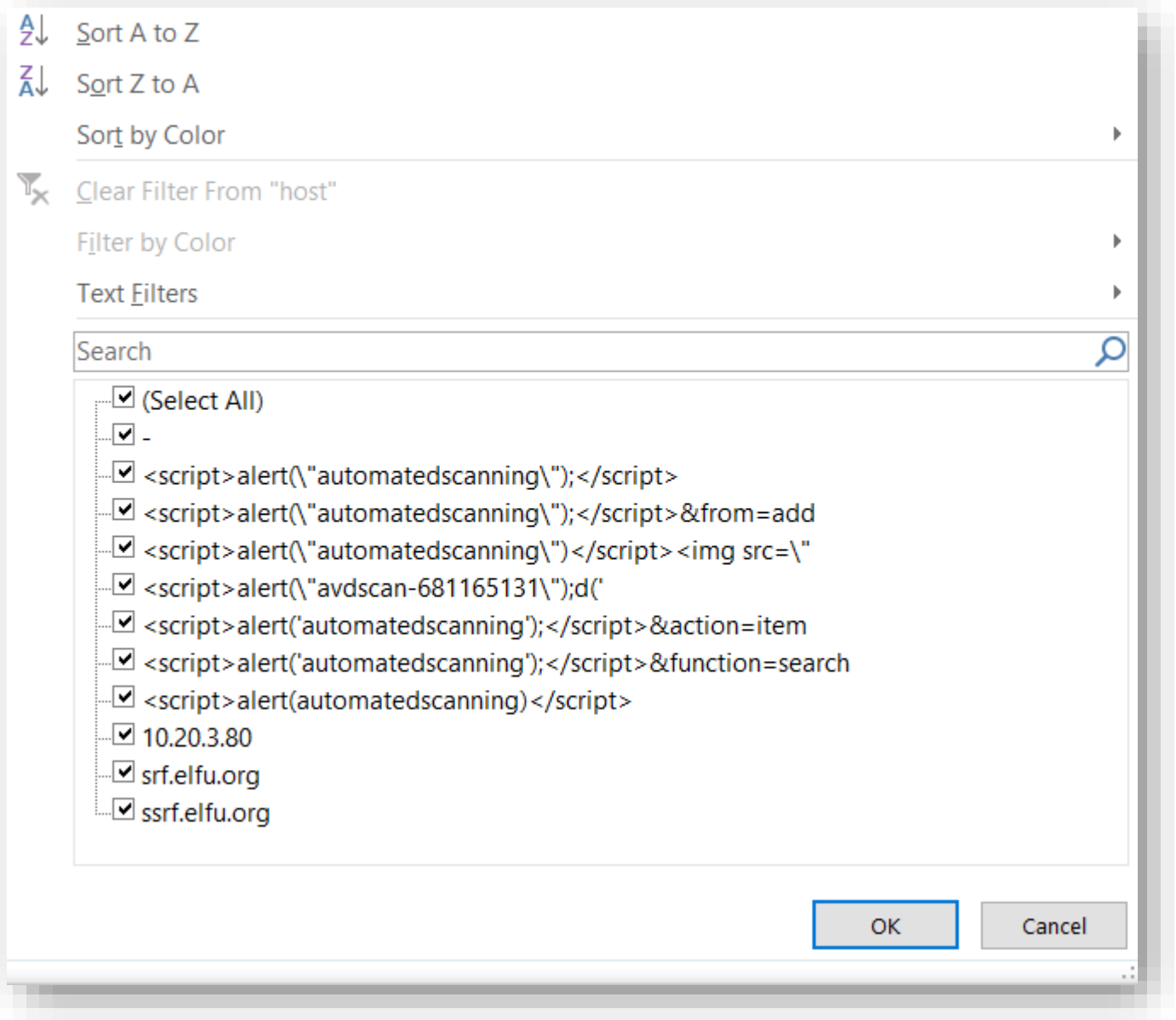


Figure 218 - Weather Data Screenshot

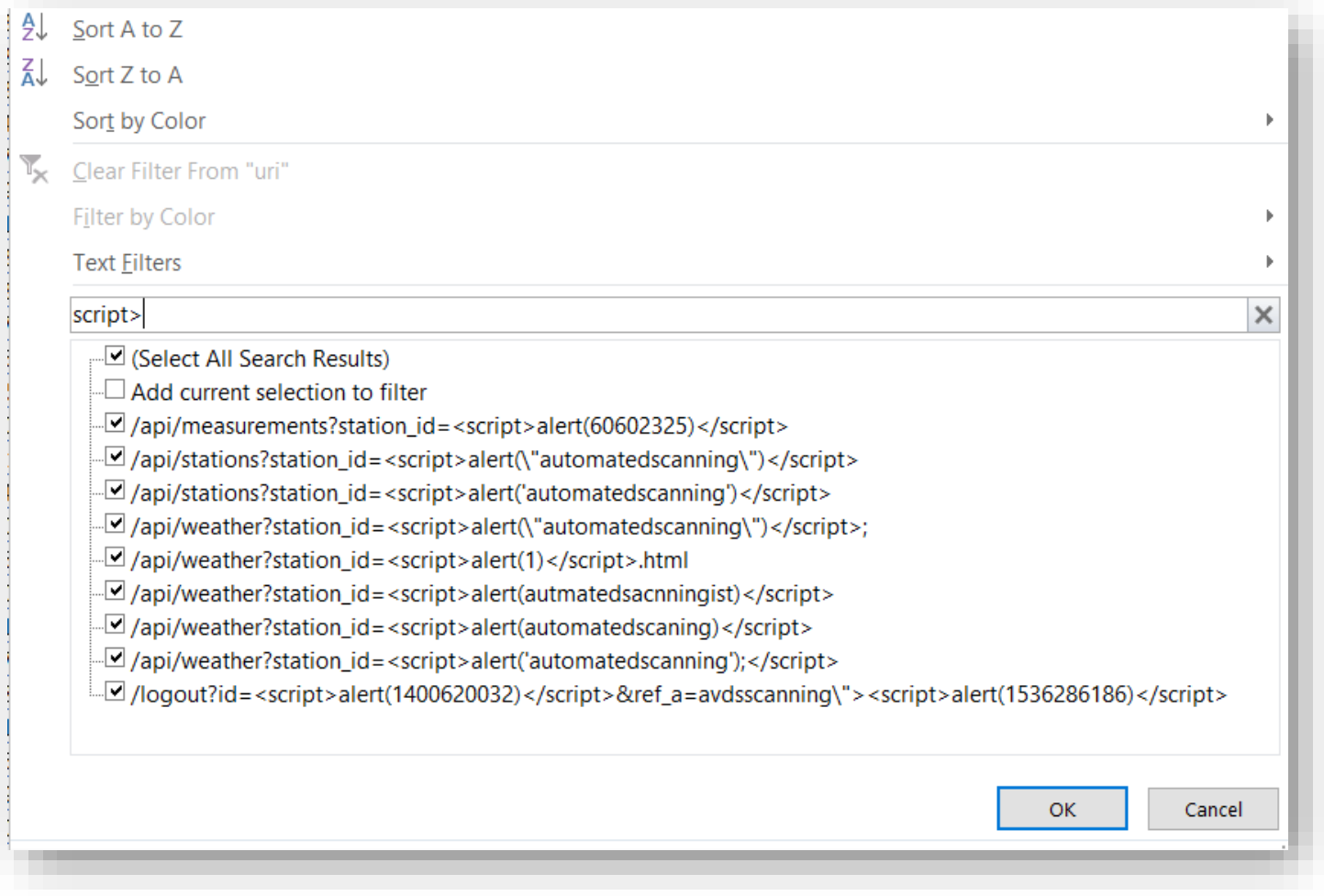


Figure 219 - Weather Data Screenshot

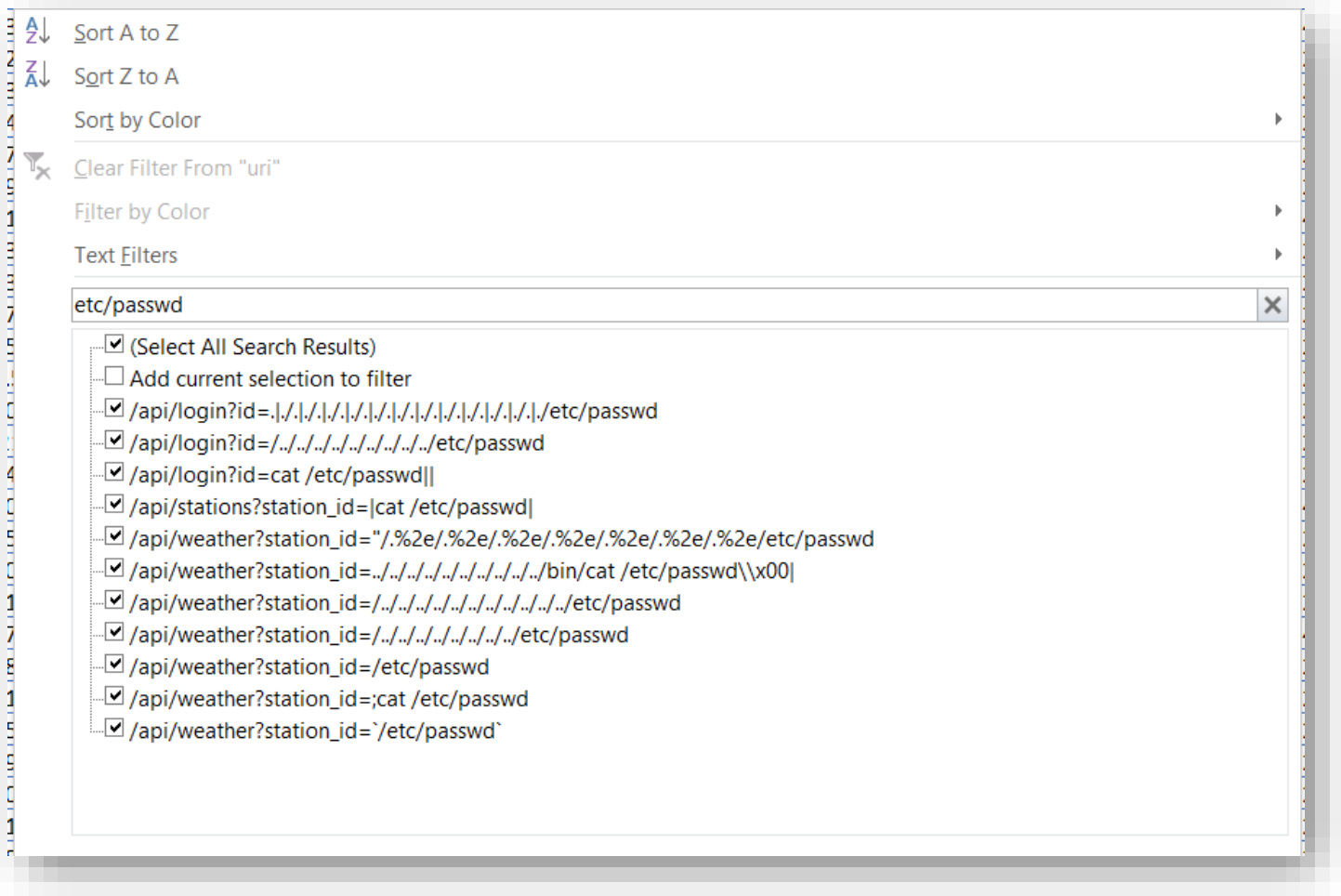


Figure 220 - Weather Data Screenshot

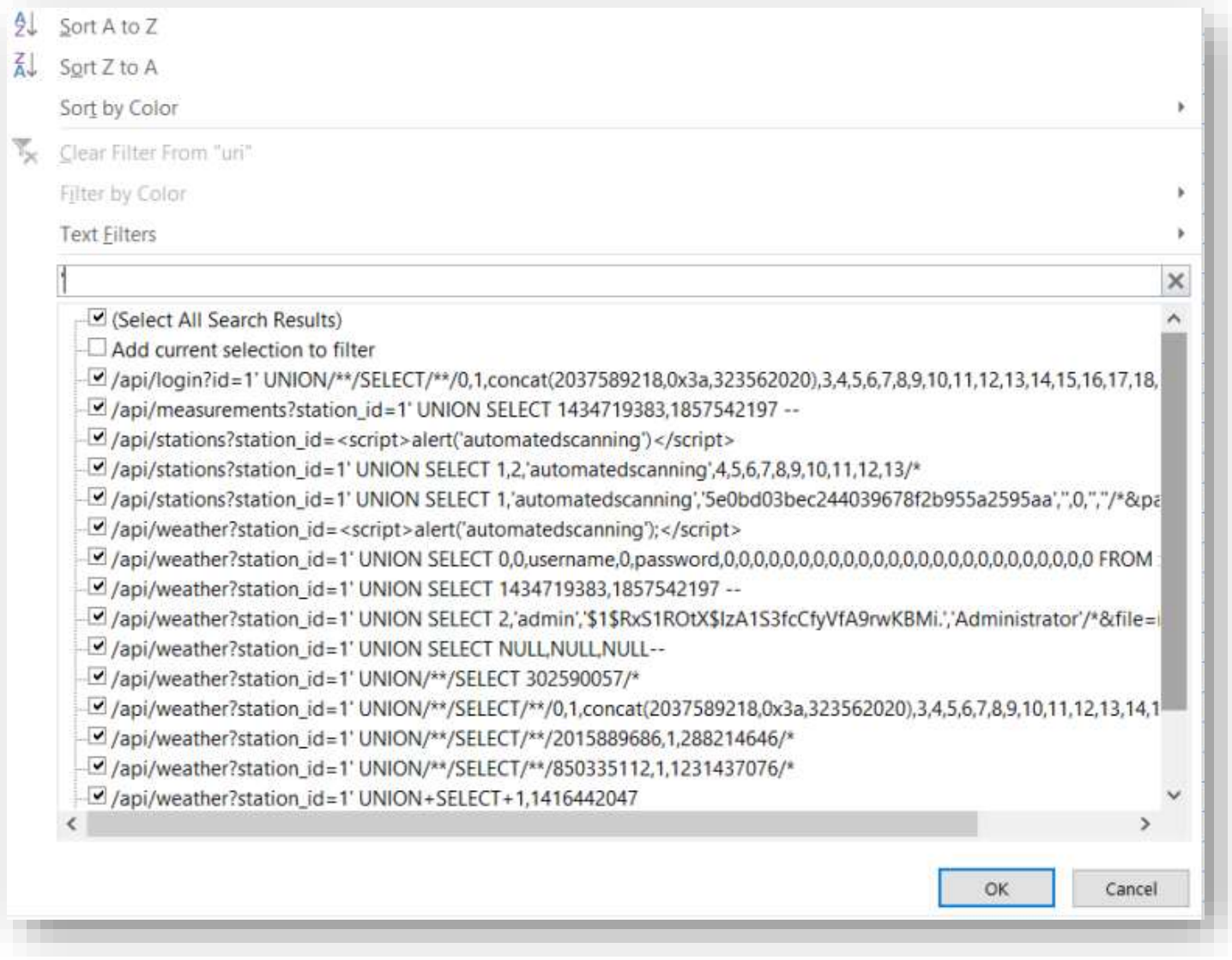


Figure 221 - Weather Data Screenshot

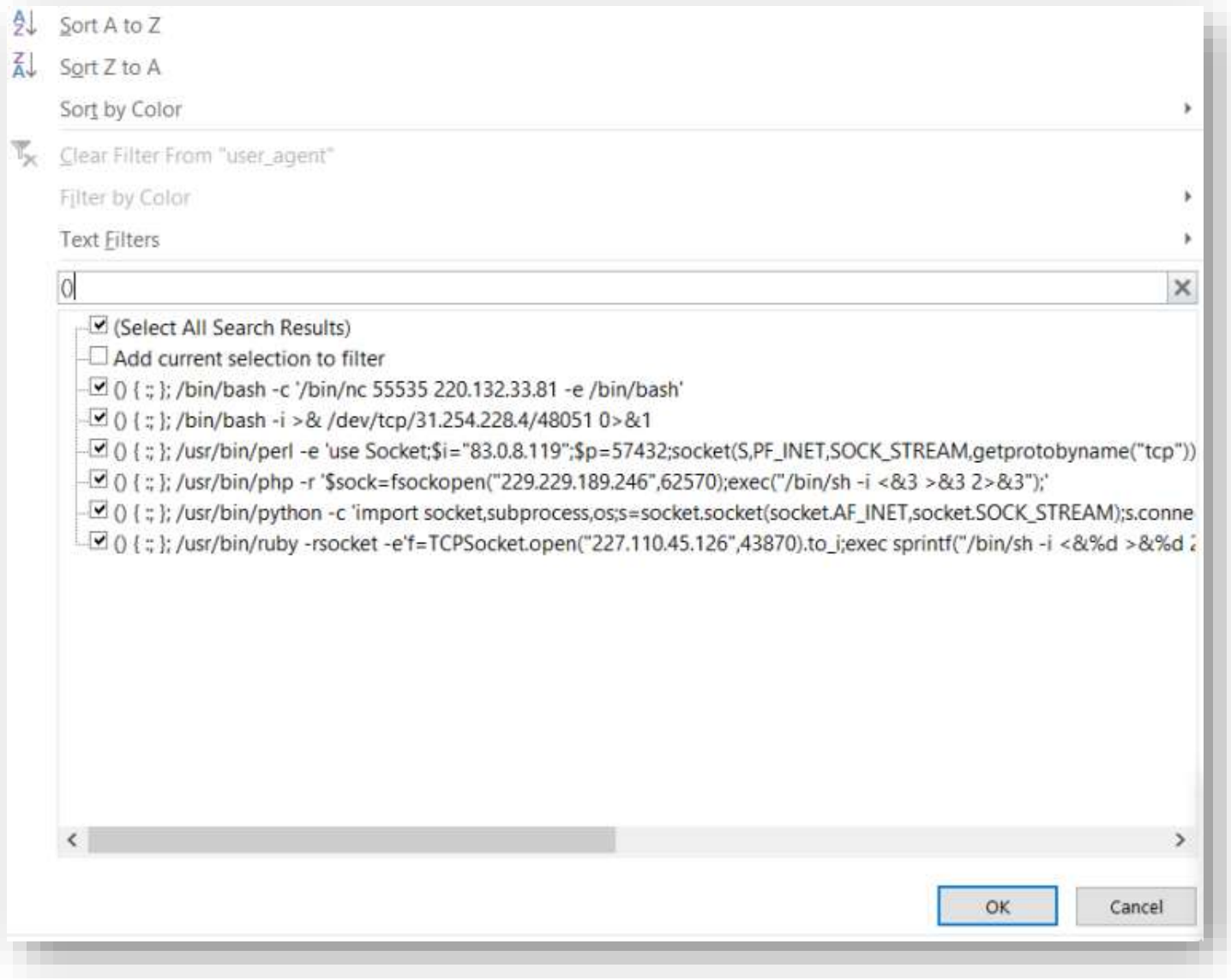


Figure 222 - Weather Data Screenshot

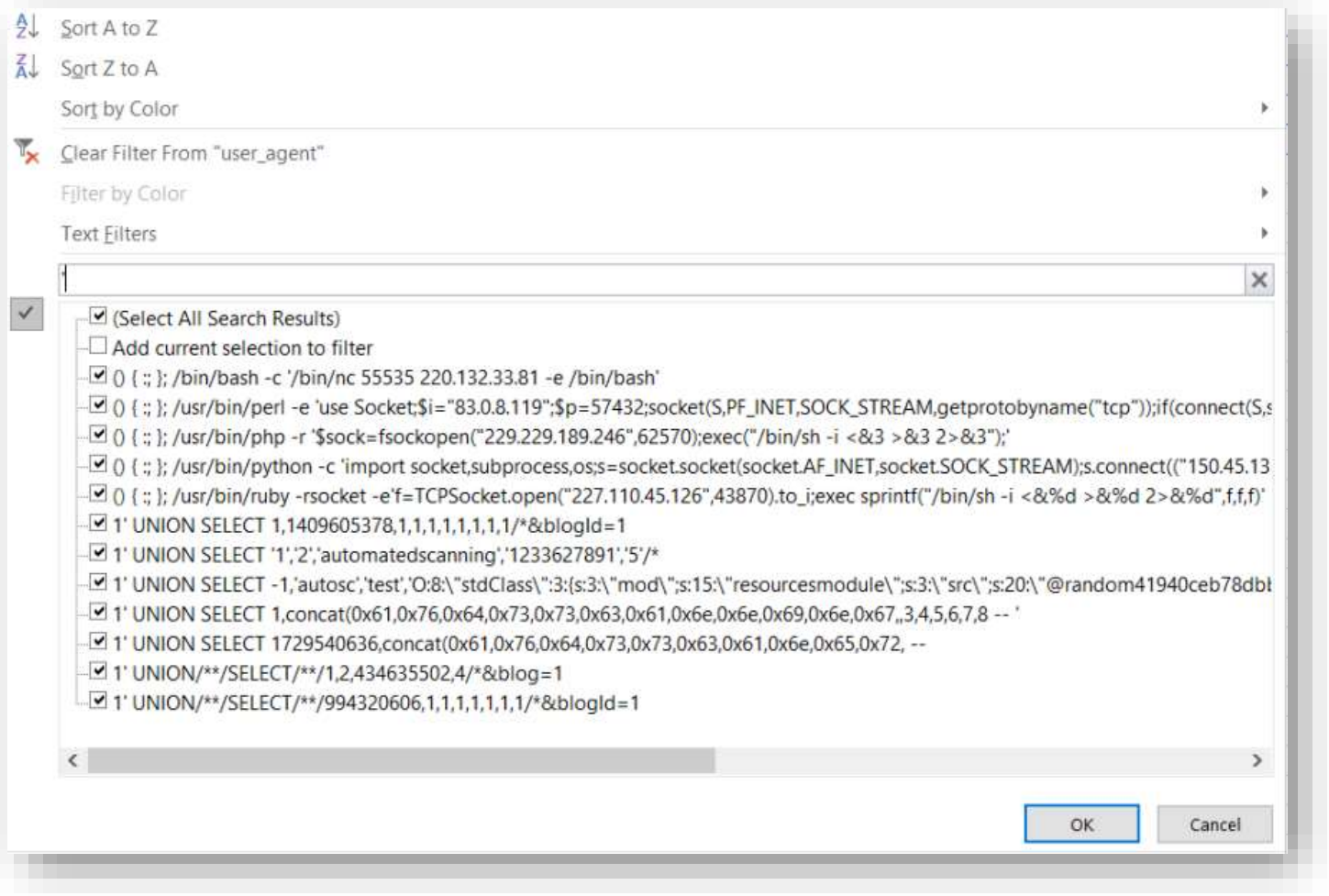


Figure 223 - Weather Data Screenshot

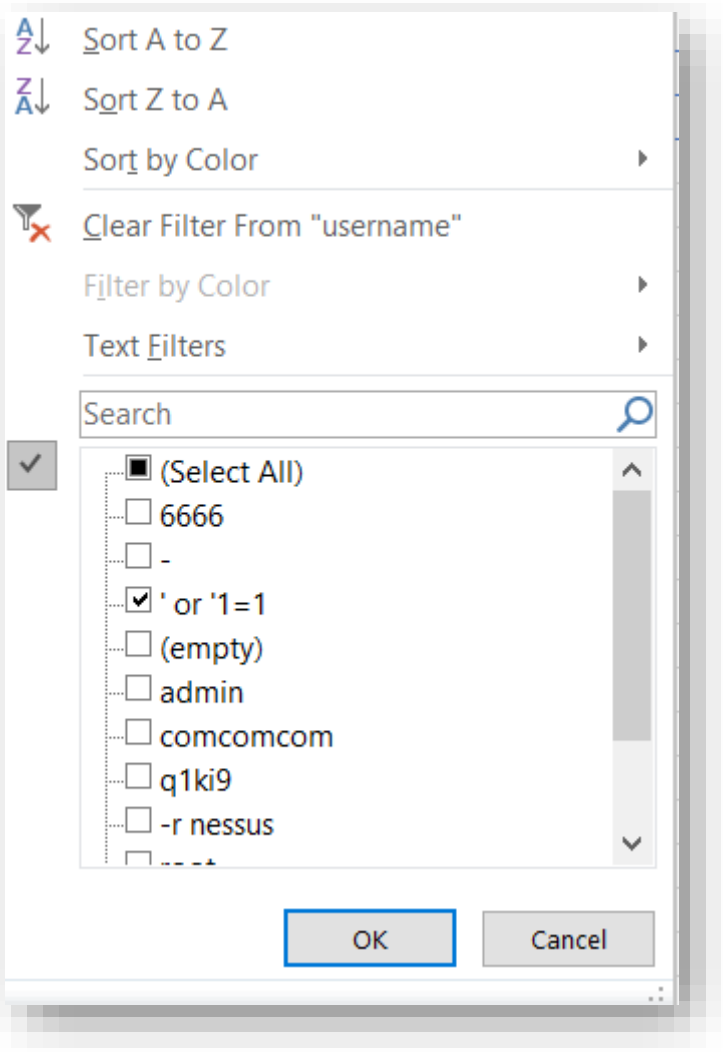


Figure 224 - Weather Data Screenshot

Doing it this way in Excel, helps us visualise it much easier and we can see a lot of it comes from automated scanning. Now to count how many of each type we have got.

Row Labels	Count of Type
LFI	11
Shellshock	6
SQLi	29
XSS	16
(blank)	
Grand Total	**62**

Figure 225 - Weather Data Screenshot

A bit low on numbers, we remember the line:

> *If you find a log entry that definitely looks bad, try pivoting off other unusual attributes in that entry to find more bad IPs.*

Since the only reasonable attribute we can pivot off now is the user agent, that's what we'll do. We're helped by seeing a user-agent for Metasploit[75] and malware[76].

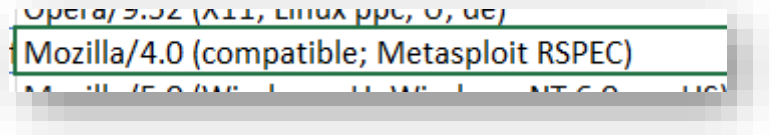


Figure 226 - Weather Data Screenshot

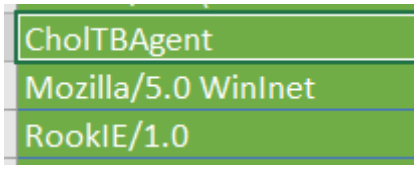


Figure 227 - Weather Data Screenshot

We perform an INDEX MATCH[77] to only bring rows back where the user agent is equal to the ones found from the initial reconnaissance. This brings back 146 which is too many. Sifting through the data, the majority seem to only have 1 or 2 records, so we exclude any IPs that appear several times (except if one has been caught already) and we now have 100 sources of poisoning (with a few duplicate IPs)

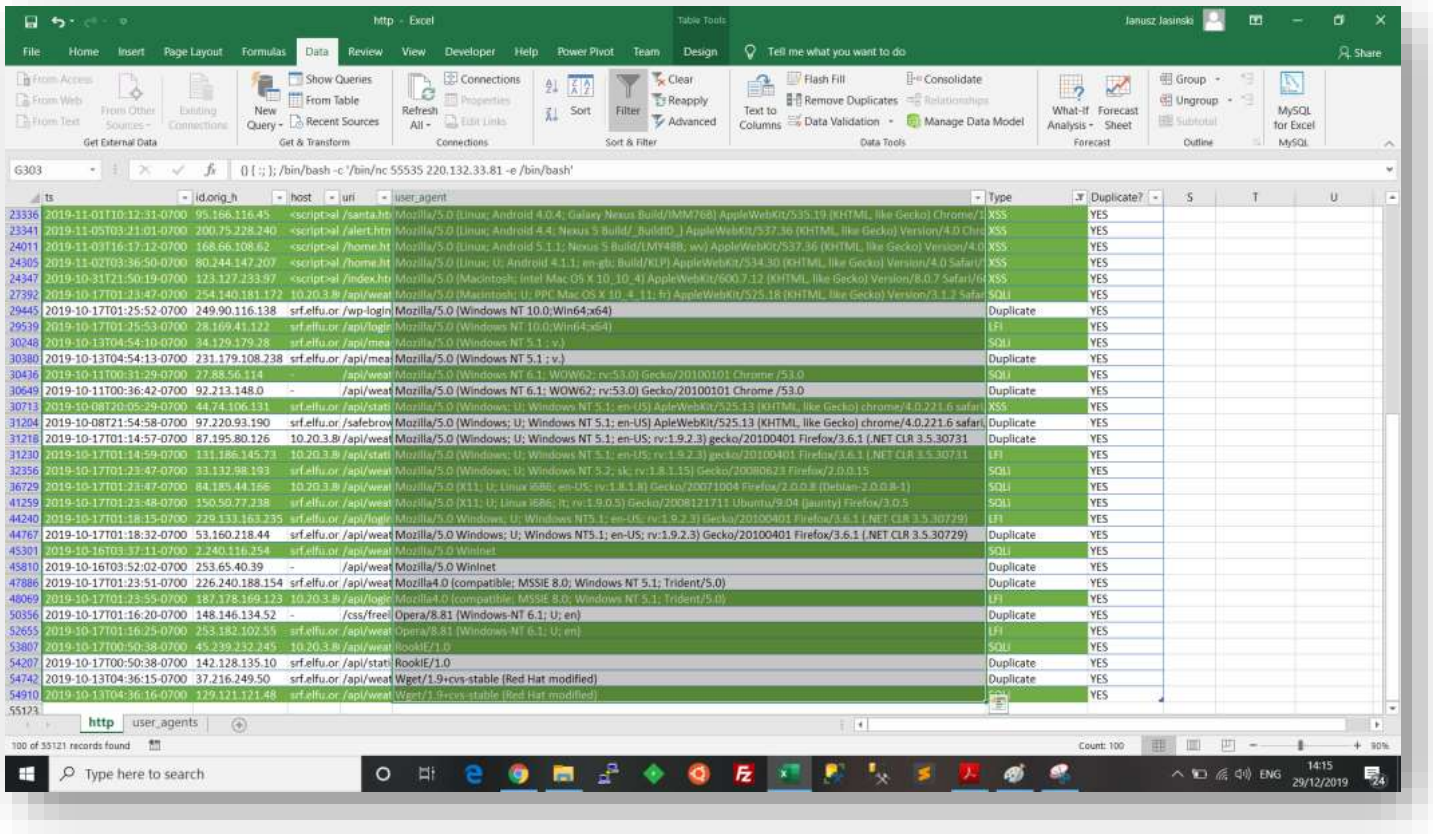


Figure 228 - Weather Data Screenshot

[75] https://www.metasploit.com/

[76] https://github.com/Neo23x0/sigma/blob/master/rules/proxy/proxy_ua_malware.yml

[77] https://exceljet.net/index-and-match

The list of IPs are collected and submitted onto https://srf.elfu.org/

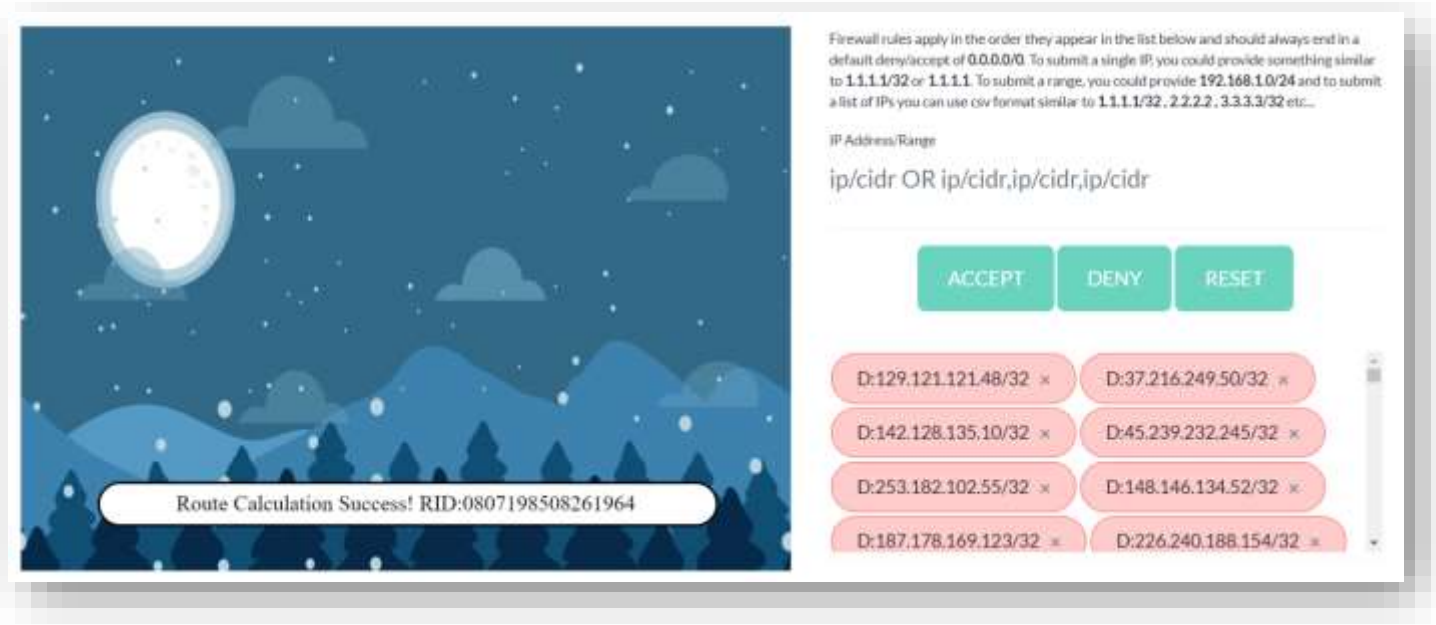


Figure 229 - Weather Data Screenshot

Success and objective completed.

Chats

Santa

Initial Chat

Completed Chat

Bushy Evergreen

Initial Chat

Completed Chat

Sugarplum Mary

Initial Chat

Completed Chat

Holly Evergreen

Initial Chat

Completed Chat

Alabaster Snowball

Initial Chat

Completed Chat

Professor Banas

Initial Chat

Completed Chat

Sparkle Redberry

Initial Chat

Completed Chat

Michael and Jane – Two Turtle Doves

Chat

Kent Tinseltooth

Initial Chat

Completed Chat

The Tooth Fairy

Initial Chat

Completed Chat

Wurnose Openslae

Initial Chat

Completed Chat

Krampus

Initial Chat

Completed Chat

Tangle Coalbox

Initial Chat

Completed Chat

Pepper Minstix

Initial Chat

Completed Chat

Minty Candycane

Initial Chat

Completed Chat

Shinny Upatree

Initial Chat

Completed Chat

Appendix

Splunk – Empire downloads (https://medium.com/@netscylla/powershell-that-looks-smells-like-empire-payloads-7f9bfdd39e5b)