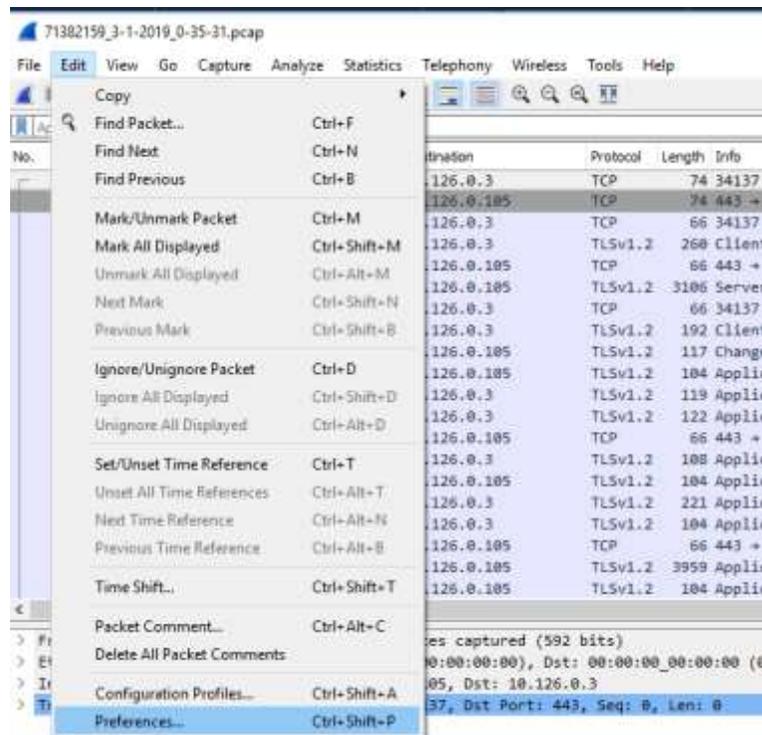


Objective--Network Traffic Forensics (Part 4)

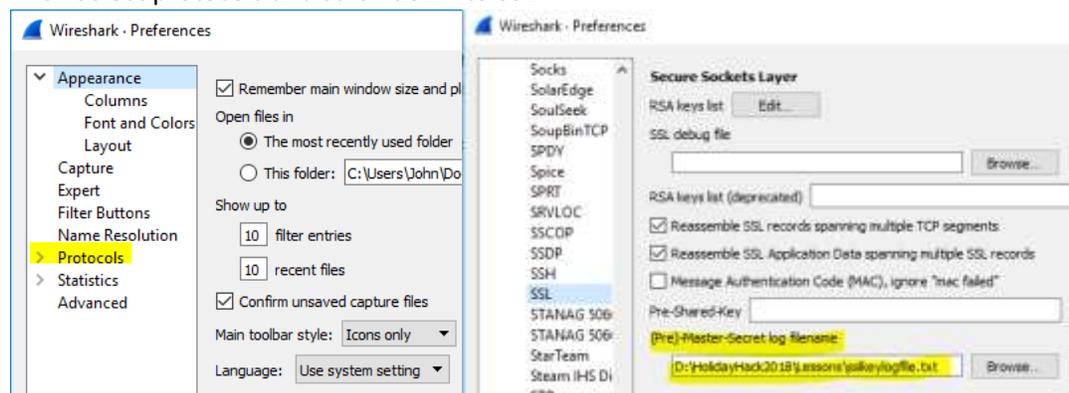
Solution (or part of it)

Now that we have sslkeylogfile.txt, or whatever you named it, we can decrypt the packet capture that the Packalyzer server gives us. Be sure to tell Packalyzer to sniff, then download the packet, and then use https://packalyzer.kringlecastle.com/dev/packalyzer_clientrandom_ssl.log/ to download the key file. Don't wait too long between taking the packet capture and downloading the key file; otherwise they won't match, and the traffic will not be decrypted.

When you open the pcap file from Packalyzer, you will see that the traffic is all encrypted by TLS 1.2. As in Chris' video, we need to edit Wireshark's preferences to include the key file. Select Edit > Preferences...



Then select protocols and scroll down to SSL.



With SSL selected, insert the path to the SSLKEYLOGFILE we found. The traffic will magically be decrypted. Remember, this only works because the browser (or other application) was recording the keys it used. We could not have decrypted the traffic just by intercepting it.

71382159_3-1-2019_0-35-31.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.126.0.105	10.126.0.3	TCP	74	34137 → 443 [SYN] Seq=0 Win=43696
2	0.000011	10.126.0.3	10.126.0.105	TCP	74	443 → 34137 [SYN, ACK] Seq=0 Ack=
3	0.000022	10.126.0.105	10.126.0.3	TCP	66	34137 → 443 [ACK] Seq=1 Ack=1 Win=
4	0.009025	10.126.0.105	10.126.0.3	TLSv1.2	260	Client Hello
5	0.009051	10.126.0.3	10.126.0.105	TCP	66	443 → 34137 [ACK] Seq=1 Ack=195 Win=
6	0.010766	10.126.0.3	10.126.0.105	TLSv1.2	3106	Server Hello, Certificate, Server Key
7	0.010773	10.126.0.105	10.126.0.3	TCP	66	34137 → 443 [ACK] Seq=195 Ack=304 Win=
8	0.011914	10.126.0.105	10.126.0.3	TLSv1.2	192	Client Key Exchange, Change Cipher
9	0.012784	10.126.0.3	10.126.0.105	TLSv1.2	117	Change Cipher Spec, Finished
10	0.012797	10.126.0.3	10.126.0.105	HTTP2	104	SETTINGS[0]
11	0.013035	10.126.0.105	10.126.0.3	HTTP2	119	Magic
12	0.013059	10.126.0.105	10.126.0.3	HTTP2	122	SETTINGS[0]
13	0.013065	10.126.0.3	10.126.0.105	TCP	66	443 → 34137 [ACK] Seq=3130 Ack=43
14	0.013069	10.126.0.105	10.126.0.3	HTTP2	108	WINDOW_UPDATE[0]
15	0.013183	10.126.0.3	10.126.0.105	HTTP2	104	SETTINGS[0]
16	0.013245	10.126.0.105	10.126.0.3	HTTP2	221	HEADERS[1]: GET /
17	0.013853	10.126.0.105	10.126.0.3	HTTP2	104	SETTINGS[0]
18	0.013901	10.126.0.3	10.126.0.105	TCP	66	443 → 34137 [ACK] Seq=3168 Ack=60
19	0.014380	10.126.0.3	10.126.0.105	HTTP2	3959	DATA[1]
20	0.014629	10.126.0.3	10.126.0.105	HTTP2	104	DATA[1] (text/html)

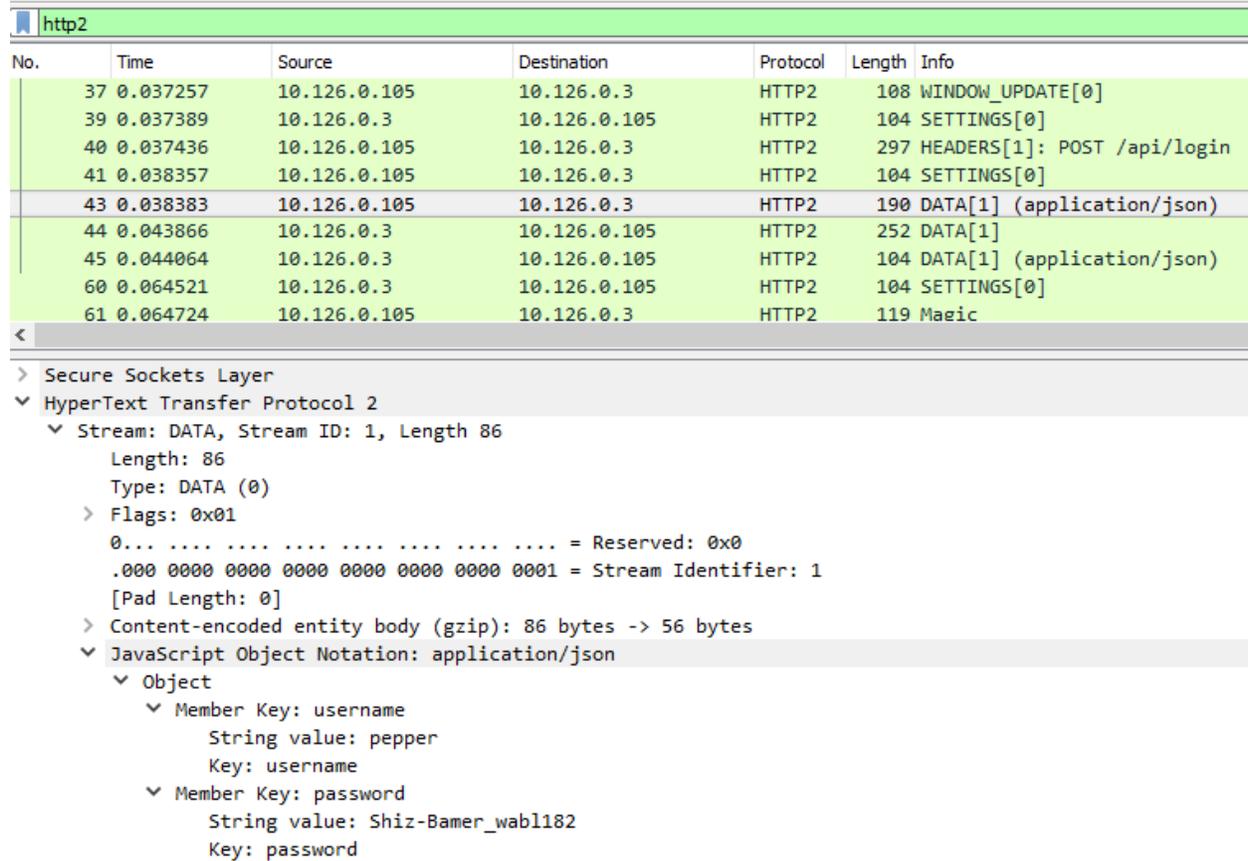
As we examine the traffic using the http2 Display Filter that Chris showed us, we do see something of interest. There is a Header with POST /api/login. Remember that HTTP/2 puts the headers and data into different frames.

http2

No.	Time	Source	Destination	Protocol	Length	Info
37	0.037257	10.126.0.105	10.126.0.3	HTTP2	108	WINDOW_UPDATE[0]
39	0.037389	10.126.0.3	10.126.0.105	HTTP2	104	SETTINGS[0]
40	0.037436	10.126.0.105	10.126.0.3	HTTP2	297	HEADERS[1]: POST /api/login
41	0.038357	10.126.0.105	10.126.0.3	HTTP2	104	SETTINGS[0]
43	0.038383	10.126.0.105	10.126.0.3	HTTP2	190	DATA[1] (application/json)
44	0.043866	10.126.0.3	10.126.0.105	HTTP2	252	DATA[1] (application/json)
45	0.044064	10.126.0.3	10.126.0.105	HTTP2	104	DATA[1] (application/json)
60	0.064521	10.126.0.3	10.126.0.105	HTTP2	104	SETTINGS[0]
61	0.064724	10.126.0.105	10.126.0.3	HTTP2	119	Magic
62	0.064747	10.126.0.105	10.126.0.3	HTTP2	122	SETTINGS[0]
63	0.064758	10.126.0.105	10.126.0.3	HTTP2	108	WINDOW_UPDATE[0]
65	0.064874	10.126.0.3	10.126.0.105	HTTP2	104	SETTINGS[0]

Remember that Follow > TCP Stream will just show us the encrypted traffic. Follow > SSL Stream is better in that shows the decrypted traffic, but it is still in gzip, so we cannot read it. The HTTP/2 section of the data pane is much better.

When we look at the DATA frames with application/json data, some of them bring joy to an attacker's heart.



No.	Time	Source	Destination	Protocol	Length	Info
37	0.037257	10.126.0.105	10.126.0.3	HTTP2	108	WINDOW_UPDATE[0]
39	0.037389	10.126.0.3	10.126.0.105	HTTP2	104	SETTINGS[0]
40	0.037436	10.126.0.105	10.126.0.3	HTTP2	297	HEADERS[1]: POST /api/login
41	0.038357	10.126.0.105	10.126.0.3	HTTP2	104	SETTINGS[0]
43	0.038383	10.126.0.105	10.126.0.3	HTTP2	190	DATA[1] (application/json)
44	0.043866	10.126.0.3	10.126.0.105	HTTP2	252	DATA[1]
45	0.044064	10.126.0.3	10.126.0.105	HTTP2	104	DATA[1] (application/json)
60	0.064521	10.126.0.3	10.126.0.105	HTTP2	104	SETTINGS[0]
61	0.064724	10.126.0.105	10.126.0.3	HTTP2	119	Maetic

```
> Secure Sockets Layer
v HyperText Transfer Protocol 2
  v Stream: DATA, Stream ID: 1, Length 86
    Length: 86
    Type: DATA (0)
    > Flags: 0x01
      0... .. = Reserved: 0x0
      .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
      [Pad Length: 0]
    > Content-encoded entity body (gzip): 86 bytes -> 56 bytes
  v JavaScript Object Notation: application/json
    v Object
      v Member Key: username
        String value: pepper
        Key: username
      v Member Key: password
        String value: Shiz-Bamer_wabl182
        Key: password
```

So, Pepper's credentials are pepper and Shiz-Bamer_wabl182. Cool.

The display filter Chris gave us, "http2.data.data && http2 contains username" does not work here. I'm not sure why, but perhaps we can make our own filter. The Wireshark feature that creates display filters when you right-click > Prepare a Filter > Selected is very powerful. Some of the

elements in the JSON data don't allow it, but the String value for pepper allows it.

```
> Frame 43: 190 bytes on wire (1520 bits), 190 bytes captured (1520 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: 10.126.0.105, Dst: 10.126.0.3
> Transmission Control Protocol, Src Port: 38337, Dst Port: 443, Seq: 741, Ack: 3168, Len: 124
> Secure Sockets Layer
▼ HyperText Transfer Protocol 2
  ▼ Stream: DATA, Stream ID: 1, Length 86
    Length: 86
    Type: DATA (0)
    > Flags: 0x01
    0... .. = Reserved: 0x0
    .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
    [Pad Length: 0]
    > Content-encoded entity body (gzip): 86 bytes -> 56 bytes
  ▼ JavaScript Object Notation: application/json
    ▼ Object
      ▼ Member Key: username
        String value: pepper
        Key: username
      ▼ Member Key: password
        String value: Shiz-Bamer_wabl182
        Key: password
```

That gives us a display filter.

No.	Time	Source	Destination	Protocol
35	0.037224	10.126.0.105	10.126.0.3	HTTP2

That helps, but we want something that shows use all the packets that have a username (or password would do.) After some fiddling, I arrived at the display filter `json.key==username`.

The image shows a Wireshark interface with a display filter `json.key == "username"` applied. A table of filtered packets is shown:

No.	Time	Source	Destination	Protocol	Length	Info
43	0.038383	10.126.0.105	10.126.0.3	HTTP2	190	DATA[1] (application/json)
120	3.056222	10.126.0.106	10.126.0.3	HTTP2	197	DATA[1] (application/json)
196	5.057790	10.126.0.104	10.126.0.3	HTTP2	202	DATA[1] (application/json)
272	9.059618	10.126.0.106	10.126.0.3	HTTP2	197	DATA[1] (application/json)

The detailed view of packet 196 shows the following structure:

- Frame 196: 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: 33697, Dst Port: 443, Seq: 741, Ack: 3168, Len: 136
- Secure Sockets Layer
- HyperText Transfer Protocol 2
 - 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]
 - Content-encoded entity body (gzip): 98 bytes -> 65 bytes
 - JavaScript Object Notation: application/json
 - Object
 - Member Key: username
 - String value: **alabaster**
 - Key: username
 - Member Key: password
 - String value: **Packer-p@re-turntable192**
 - Key: password

There are four packets that contain credentials, and one of them has Alabaster's. Perhaps Alabaster's credentials will get us into Packalyzer.

Look at the pcap Alabaster has stored! We are getting close to the end.

The screenshot shows the Packalyzer web interface. A modal window titled "Saved Pcaps" is open, displaying a table with the following content:

Name	Download	Reanalyze	Delete
super_secret_packet_capture.pcap			

At the bottom of the modal, there is a "CLOSE" button.

Hand In

Download the `super_secret_packet_capture.pcap` file and discover its secrets. You will have to extract a file from an SMTP attachment. Once you do, you can answer the question: What is the song that Alabaster and Holly are discussing? Thankfully, the packet capture is plain text SMTP.

- 1) How is the attached file encoded?
- 2) How did you extract the file from the SMTP stream?
- 3) What is the name of the song?