Limitless HTTP in an HTTPS World

Inferring the Semantics of the HTTPS Protocol without Decryption

Blake Anderson, PhD †; Andrew Chi †‡; Scott Dunlop †; and David McGrew, PhD †
† Cisco, ‡ University of North Carolina
contact: blake.anderson@cisco.com

ANRW 2019 (originally presented at CODASPY’19)
July 22, 2019
Goal: Given a stream of encrypted TLS applications records, infer:
- the underlying HTTP frames, and
- for HEADERS frames, identify fields/values

Higher level goals: Use these techniques to improve the detection of ...
- Defender: malicious communication/websites, data exfiltration
- Attacker: blocked domains
Motivation
Motivation

![Diagram showingClient to Server communication with Enterprise in the middle, where there is a Man-in-the-Middle (MiTM) attack blocking the TLS protocol]
TLS Application Data Records

Client

app_data

application_data

application_data

Server

application_data

application_data

application_data
Encrypted HTTP/2 Frames
Extracting TLS Key Material

```
struct ssl_session_st {
    int ssl_version;
    unsigned int key_arg_length;
    unsigned char key_arg[8];
    int master_key_length;
    unsigned char master_key[48];
    unsigned int session_id_length;
    unsigned char session_id[32];
    ...
```

```
03 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 00 00 00 44 0E 70 5C 1C 22 45 07 6C 1C ED 0D
E3 74 DF E2 C9 71 AF 41 2C 0B E6 AF 70 32 6E C3
A3 2C A0 E6 3A 7A FF 0E F3 70 A2 8A 88 52 B2 2D
D1 B3 F6 F2 20 00 00 00 CD 31 58 BF DF 97 B0 F8
C0 86 BA 48 47 93 B0 A5 BA C1 5B 4B 35 37 7F 98
```
Decrypting TLS

► Extracting Key Material
  ● SSLKEYLOGFILE environment variable when available
  ● Regular expressions for OpenSSL, BoringSSL, NSS, Schannel, Tor AES keys
  ● Regular expressions tuned to run in ~400ms for 1GB memory dump

► Decrypting TLS Sessions
  ● Bespoke python program supporting SSL 2.0 - TLS 1.3
  ● Support for HTTP/1.x, HTTP/2.0, Tor
  ● Write output as either JSON or a decrypted pcap
Decrypting Tor

"type": "application_data",
"length": 1052,

"protocol": "Tor",
"length": 1028,
"cells": [null]

"circ_id": "xxxxxxxx",
"cell_type": "RELAY",
"command": "RELAY_DATA",
"stream_id": "xxxx",
"digest": "xxxxxxxx",
"length": 340,

"decrypted_data": {
"tls_records": [null]

"type": "application_data",
"length": 335,

"method": "GET",
"uri": "/",
"v": "HTTP/1.1",
"headers": [null]
}
Decrypting Tor

"type": "application_data",
"length": 1052,
"decrypted_data": {
  "protocol": "Tor",
  "length": 1028,
  "cells": [
    {
      "circ_id": "xxxxxxxx",
      "cell_type": "RELAY",
      "command": "RELAY_DATA",
      "stream_id": "xxxx",
      "digest": "xxxxxxxx",
      "length": 340,
      "decrypted_data": {
      ...
    }
  ...
}

Tor Protocol
"type": "application_data",
"length": 1052,
"decrypted_data": {
  "protocol": "Tor",
  "length": 1028,
  "cells": [
    {
      "circ_id": "xxxxxxxx",
      "cell_type": "RELAY",
      "command": "RELAY_DATA",
      "stream_id": "xxxx",
      "digest": "xxxxxxxx",
      "length": 340,
      "decrypted_data": {
        "tls_records": [
          {
            "type": "application_data",
            "length": 335,
            "
          }
        ]
      }
    }
  ]
}
Decrypting Tor

```
"type": "application_data",
"length": 1052,
"decrypted_data": {
  "protocol": "Tor",
  "length": 1028,
  "cells": [
    {
      "circ_id": "xxxxxxxx",
      "cell_type": "RELAY",
      "command": "RELAY_DATA",
      "stream_id": "xxxx",
      "digest": "xxxxxxxx",
      "length": 340,
      "decrypted_data": {
        "tls_records": [
          {
            "type": "application_data",
            "length": 335,
            "decrypted_data": {
              "method": "GET",
              "uri": "/",
              "v": "HTTP/1.1",
              "headers": [
                ...
              ],
              ...
            }
          }
        ]
      }
    }
  ]
}
```
Decryption Lab

- Chrome, Firefox, Tor Browser
- Contact each site in the Alexa top-1,000 daily
- Record packet captures and key material
  - \{Firefox, Chrome\} → SSLKEYLOGFILE
  - Tor Browser → memory snapshots of the tor and firefox processes
Malware Sandbox

- Production malware analysis system running Windows 7 and 10
- Submitted samples ran for 5 minutes
- Key material extracted from memory dump post-run
  - ~80% of TLS connections successfully decrypted
## Datasets

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>TLS Connections</th>
<th>HTTP/1.1 TX’s</th>
<th>HTTP/2 TX’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>firefox</td>
<td>61,091</td>
<td>72,828</td>
<td>132,685</td>
</tr>
<tr>
<td>chrome</td>
<td>379,734</td>
<td>515,022</td>
<td>561,666</td>
</tr>
<tr>
<td>tor</td>
<td>6,067</td>
<td>50,799</td>
<td>0</td>
</tr>
<tr>
<td>malware</td>
<td>86,083</td>
<td>182,498</td>
<td>14,734</td>
</tr>
</tbody>
</table>
Data Features

We analyze the current, preceding 5, and following 5 TLS records; for each TLS record, we extract:

1. The number of packets
2. The number of packets with the TCP PUSH flag set
3. The average packet size in bytes
4. The type code of the TLS record
5. The TLS record size in bytes
6. The direction of the TLS record
Algorithm 1 Iterative HTTP Inference

1: procedure ITERATIVE_SEMANTICS_CLASSIFY
2:   given:
3:     conn := features describing connection
4:     alp ← application_layer_protocol(conn)
5:     recs ← classify_message_types(conn, alp)
6:     for rec ∈ recs do:
7:       if rec.type ≠ Headers then:
8:         continue
9:       get_record_features(rec, alp)
10:      classify_semantics(rec, alp)
11:     while not converged do:
12:       for rec ∈ recs do:
13:         if rec.type ≠ Headers then:
14:           continue
15:         get_record_features(rec, alp)
16:         get_inferred_features(rec, alp)
17:         classify_semantics(rec, alp)
## Interesting Inferences

<table>
<thead>
<tr>
<th>Problem</th>
<th>HTTP/1.1 Label Set</th>
<th>HTTP/2 Label Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>method (request)</td>
<td>GET, POST, OPTIONS, HEAD, PUT</td>
<td>GET, POST, OPTIONS, HEAD</td>
</tr>
<tr>
<td>Content-Type (request)</td>
<td>json, plain</td>
<td>json, plain</td>
</tr>
<tr>
<td>status-code (response)</td>
<td>100, 200, 204, 206, 302, 303, 301, 304, 307, 404</td>
<td>200, 204, 206, 301, 302, 303, 304, 307, 404</td>
</tr>
<tr>
<td>Content-Type (response)</td>
<td>html, javascript, image, video, css, octet, json, font, plain</td>
<td>html, javascript, image, video, css, octet, json, font, plain, protobuf</td>
</tr>
<tr>
<td>Server (response)</td>
<td>nginx-1.13/1.12/1.11/1.10, nginx-1.8/1.7/1.4, Apache cloudflare-nginx, nginx,</td>
<td>nginx-1.13/1.12/1.11/1.10, nginx-1.6/1.4/1.3, nginx cloudflare-nginx, Apache</td>
</tr>
<tr>
<td></td>
<td>AmazonS3, NetDNA/2.2, IIS-7.5/8.5, jetty-9.4/9.0, openresty, Coyote/1.1</td>
<td>Coyote/1.1, IIS/8.5, sfe, Golfe2, UploadServer, gws, Dreamlab, Tengine, Akamai,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cafe, Google, GSE, Dreamlab, Tengine, ESF, AmazonS3, NetDNA/2.2</td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>Dataset</th>
<th>HTTP/1.1</th>
<th></th>
<th></th>
<th>HTTP/2</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time-Based Split</td>
<td>SNI-Based Split</td>
<td></td>
<td>Time-Based Split</td>
<td>SNI-Based Split</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$F_1$ Score</td>
<td>Acc</td>
<td></td>
<td>$F_1$ Score</td>
<td>Acc</td>
<td></td>
</tr>
<tr>
<td>message-type</td>
<td>firefox</td>
<td>0.996</td>
<td>0.996</td>
<td></td>
<td>0.987</td>
<td>0.991</td>
<td></td>
</tr>
<tr>
<td></td>
<td>chrome</td>
<td>0.991</td>
<td>0.993</td>
<td></td>
<td>0.986</td>
<td>0.986</td>
<td></td>
</tr>
<tr>
<td></td>
<td>malware</td>
<td>0.995</td>
<td>0.996</td>
<td></td>
<td>0.981</td>
<td>0.989</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tor</td>
<td>0.869</td>
<td>0.878</td>
<td></td>
<td>0.845</td>
<td>0.848</td>
<td></td>
</tr>
<tr>
<td>method</td>
<td>firefox</td>
<td>0.943</td>
<td>0.995</td>
<td></td>
<td>0.989</td>
<td>0.997</td>
<td></td>
</tr>
<tr>
<td></td>
<td>chrome</td>
<td>0.978</td>
<td>0.998</td>
<td></td>
<td>0.936</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td></td>
<td>malware</td>
<td>0.705</td>
<td>0.996</td>
<td></td>
<td>0.831</td>
<td>0.981</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tor</td>
<td>0.846</td>
<td>0.965</td>
<td></td>
<td>0.865</td>
<td>0.973</td>
<td></td>
</tr>
<tr>
<td>Content-Type</td>
<td>firefox</td>
<td>0.967</td>
<td>0.978</td>
<td></td>
<td>0.982</td>
<td>0.985</td>
<td></td>
</tr>
<tr>
<td></td>
<td>chrome</td>
<td>0.977</td>
<td>0.993</td>
<td></td>
<td>0.998</td>
<td>0.998</td>
<td></td>
</tr>
<tr>
<td></td>
<td>malware</td>
<td>0.888</td>
<td>0.900</td>
<td></td>
<td>0.853</td>
<td>0.862</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tor</td>
<td>0.836</td>
<td>0.904</td>
<td></td>
<td>0.659</td>
<td>0.864</td>
<td></td>
</tr>
<tr>
<td>Cookie (b)</td>
<td>firefox</td>
<td>0.967</td>
<td>0.974</td>
<td></td>
<td>0.941</td>
<td>0.948</td>
<td></td>
</tr>
<tr>
<td></td>
<td>chrome</td>
<td>0.977</td>
<td>0.977</td>
<td></td>
<td>0.953</td>
<td>0.958</td>
<td></td>
</tr>
<tr>
<td></td>
<td>malware</td>
<td>0.916</td>
<td>0.918</td>
<td></td>
<td>0.876</td>
<td>0.876</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tor</td>
<td>0.756</td>
<td>0.823</td>
<td></td>
<td>0.657</td>
<td>0.740</td>
<td></td>
</tr>
</tbody>
</table>
Results - Content-Type

(a) chrome

(b) malware

(c) tor
Conclusions

- Detailed inferences about the encrypted HTTP protocol are possible with careful dataset construction and feature selection.
- Multiplexing and fixed-length records provide a valuable defense against these techniques.
- Results are client dependent; TLS fingerprinting can provide guidance.
THANK YOU