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

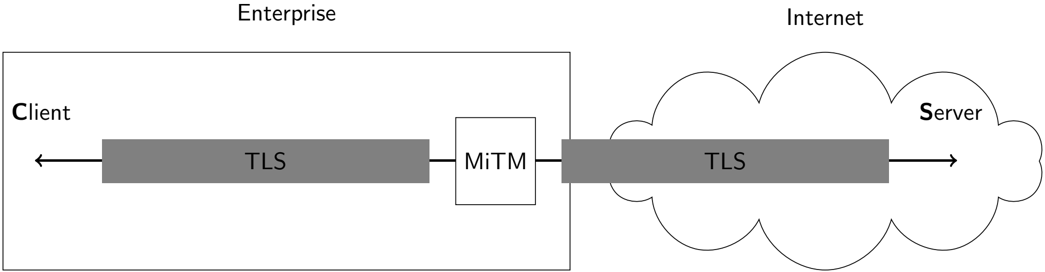
ANRW 2019 (originally presented at CODASPY'19)

July 22, 2019

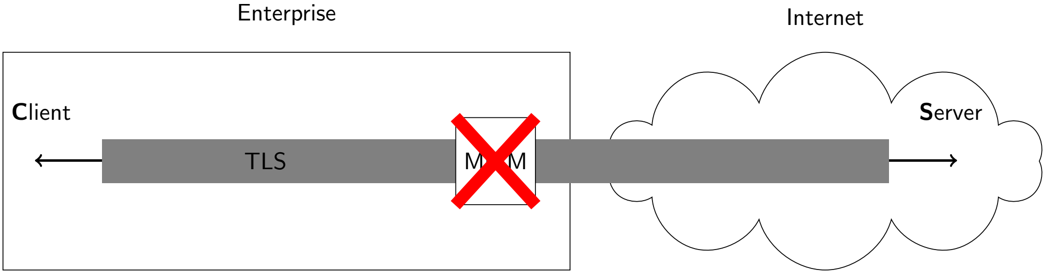
Overview

- 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

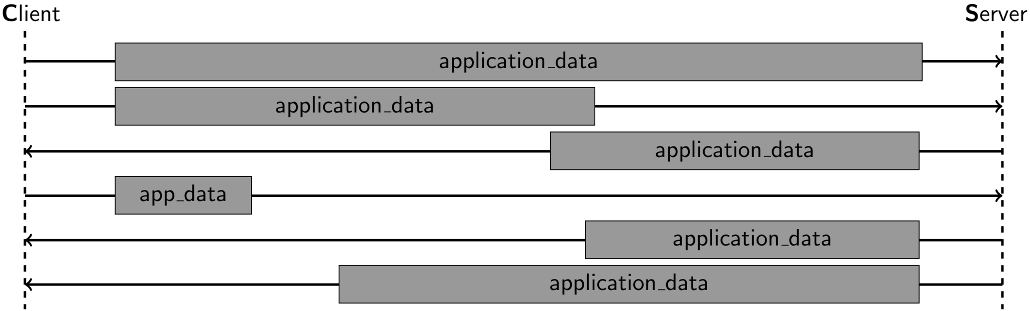
Motivation



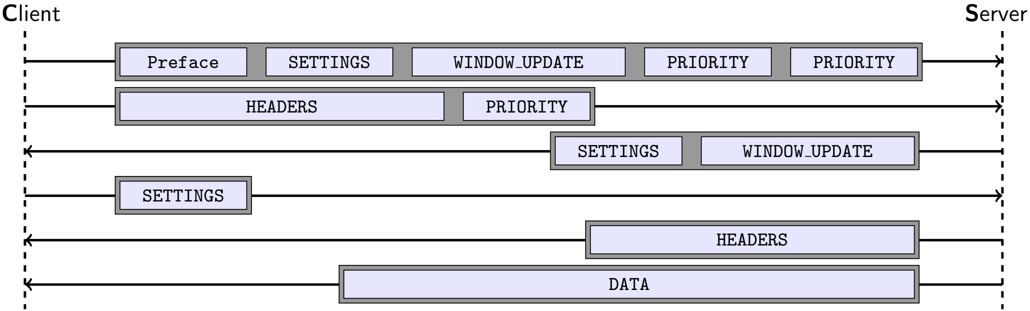
Motivation



TLS Application Data Records



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 $\sim$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,

[redacted]

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,
[redacted]
```


Tor Protocol

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,
```

[redacted]

Tor Protocol

TLS Protocol

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": [
                ...
              ],
       ...
```


Tor Protocol

TLS Protocol

HTTP Protocol

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
  - $\sim$80% of TLS connections successfully decrypted

Datasets

Dataset Name	TLS Connections	HTTP/1.1 TX's	HTTP/2 TX's
`firefox`	61,091	72,828	132,685
`chrome`	379,734	515,022	561,666
`tor`	6,067	50,799	0
`malware`	86,083	182,498	14,734

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

Iterative Classification

Algorithm 1 Iterative HTTP Inference

```
procedure ITERATIVE_SEMANTICS_CLASSIFY
    given:
        conn := features describing connection
    alp ← application_layer_protocol(conn)
    recs ← classify_message_types(conn, alp)
    for rec ∈ recs do:
        if rec.type ≠ Headers then:
            continue
        get_record_features(rec, alp)
        classify_semantics(rec, alp)
    while not converged do:
        for rec ∈ recs do:
            if rec.type ≠ Headers then:
                continue
            get_record_features(rec, alp)
            get_inferred_features(rec, alp)
            classify_semantics(rec, alp)
```

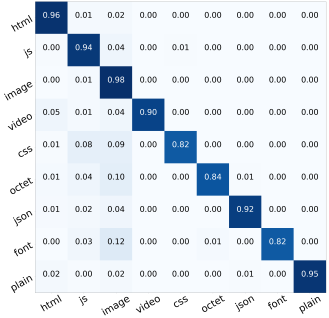
Interesting Inferences

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

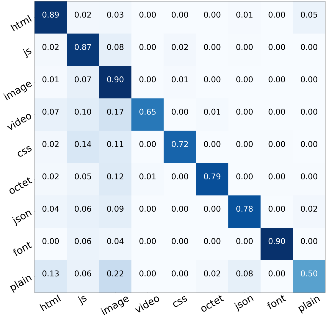
Results

Problem	Dataset	HTTP/1.1				HTTP/2			
		Time-Based Split		SNI-Based Split		Time-Based Split		SNI-Based Split	
		F_1 Score	Acc	F_1 Score	Acc	F_1 Score	Acc	F_1 Score	Acc
`message-type`	firefox	0.996	0.996	0.995	0.996	0.987	0.991	0.981	0.990
	chrome	0.991	0.993	0.989	0.991	0.986	0.986	0.982	0.984
	malware	0.995	0.996	0.995	0.996	0.981	0.989	0.979	0.986
	tor	0.869	0.878	0.845	0.848				
`method`	firefox	0.943	0.995	0.956	0.961	0.989	0.997	0.877	0.993
	chrome	0.978	0.998	0.947	0.957	0.936	0.999	0.913	0.993
	malware	0.705	0.996	0.831	0.981	0.687	0.985	0.807	0.987
	tor	0.846	0.965	0.865	0.973				
`Content-Type`	firefox	0.967	0.978	0.909	0.933	0.982	0.985	0.933	0.956
	chrome	0.977	0.993	0.874	0.875	0.998	0.998	0.842	0.864
	malware	0.888	0.900	0.853	0.862	0.711	0.887	0.811	0.890
	tor	0.836	0.904	0.659	0.864				
`Cookie` (b)	firefox	0.967	0.974	0.882	0.892	0.941	0.948	0.832	0.867
	chrome	0.977	0.977	0.929	0.934	0.953	0.958	0.856	0.941
	malware	0.916	0.918	0.876	0.876	0.898	0.913	0.850	0.861
	tor	0.756	0.823	0.657	0.740				

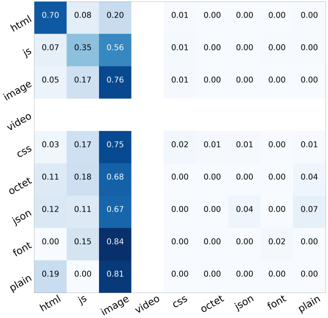
Results - Content-Type



(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