A TLS Interception Proxy with Real-Time Libpcap Export

Felix Erlacher*, Simon Woertz* and Falko Dressler†
* Dept. of Computer Science, University of Innsbruck, Austria
† Dept. of Computer Science and Heinz Nixdorf Institute, Paderborn University, Germany
simon.woertz@student.uibk.ac.at, {erlacher,dressler}@ccs-labs.org

***Abstract*—The usage of HTTPS and thereby the Transport Layer Security (TLS) protocol is constantly becoming more ubiquitous. This renders classic Deep Packet Inspection (DPI) approaches on the, now encrypted, application layer useless. The concept of TLS interception proxies is to decrypt the application layer data going through the proxy and to make it available for further processing. The decrypted data varies in form and content depending on the used proxy application. Most currently available tools store this data in plain text format consisting of the decrypted application layer data together with additional metadata describing the connection. This is useful if single connections have to be analyzed but generally fails for network monitoring tools such as Intrusion Detection System (IDS) due to missing lower layer network connection information. We developed a TLS interception proxy that exports in real-time not only the decrypted TLS application data records but also lower layer information using the well known libpcap format. This enables other network monitoring tools to further process the exported data and apply DPI techniques as well as detection rules that inspect the lower layers.**

I. Introduction

The usage of TLS connections in both local networks and the Internet is constantly increasing [1], [2]. This trend has been particularly fostered by the ever increasing privacy awareness as well as the strong need for data confidentiality. Given the fact that modern computer hardware (including mobile systems such as smartphones) is powerful enough to encrypt data streams with no impact on the data rates, many web services are changing their sites to default to TLS encryption. While this is desirable for a user of such services, it creates new challenges for network operators. All DPI methods such as the Snort [3] or Bro [4] IDS, which were used to secure campus networks, are now obsolete because they do not work on TLS encrypted connections. This raises the need for novel approaches that make it able to provide IDS security solutions but also other network monitoring techniques also for TLS encrypted traffic. Throughout this work we use the term TLS to denote both SSL and TLS encrypted connections.

There are already tools available that allow to intercept TLS encrypted traffic and several firewall manufacturers have a TLS interception feature in their products. However, they all lack two crucial features: First, it is not possible to forward the decrypted traffic in real-time in an open format to network monitoring tools like IDSs. Secondly, even though tools exist that export in some text format, these tools still omit lower layer network information.

In this work, we present a new tool chain filling exactly this gap. The novelty of our approach is that our TLS interception proxy exports the intercepted and decrypted application layer payload together with the lower layer information in real-time using the widely used libpcap format. This way, other network monitoring tools such as the IDS Snort [3] or Vermont [5] can be directly used without the need for tool specific modifications. Additionally, HTTP/1.1 flow monitoring concepts like [6] can be extended to also being able to enable flow monitoring on TLS encrypted sessions.

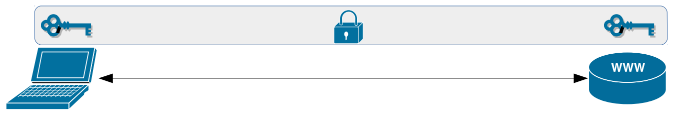
II. Related Work

There are many appliances that have TLS interception proxy capabilities, from web debugging tools (e.g., Fiddler[1]) that export the data to text files, to complex firewall solutions like Genua's GenuGate[2], which directly analyzes the decrypted application data. They all have in common that none of them exports the data in real-time in an open and generic format including the lower layers. This makes it difficult for third party applications to use the output. In the following, we describe two widely used tools.

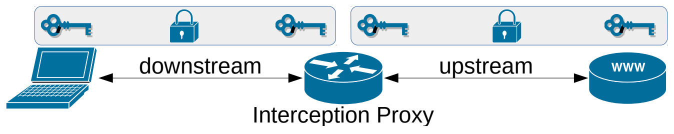
`mitmproxy`[3] is an interception proxy for HTTP with a console user interface written in Python. It is an interactive software which allows to intercept and modify HTTP requests and responses on the fly. It also allows the user to specify filters to trigger actions (e.g., logging or modifying payload), which are executed when the filter condition is met. The captured HTTP traffic can also be exported as serialized python objects and replayed with mitmproxy again. mitmproxy is also capable of acting as an intercepting TLS proxy. The decrypted connection information (time stamps, cookies, HTTP headers, decrypted payload, and other application layer info) can be exported into a text file. The tool comes bundled with a command line version of the tool called `mitmdump`. mitmproxy focuses on HTTP(S). While mitmproxy is also capable of intercepting non-HTTP TLS connections, information is currently only logged to the event log.

Wireshark[4] is a widely used network analyzer and debugger. Its main purpose is to analyze particular network traces

[1]https://www.telerik.com/fiddler
[2]https://www.genua.de/loesungen/high-resistance-firewall-genugate.html
[3]An interactive SSL-capable Intercepting HTTP Proxy for Penetration Testers and Software Developers, https://mitmproxy.org
[4]https://wireshark.org



(a) End-to-end cryptographic service



(b) Using a TLS proxy to split the cryptographic service

Fig. 1. TLS connection without and with a TLS interception proxy

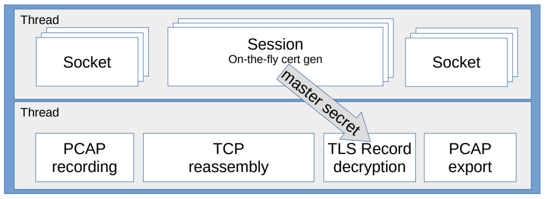


Fig. 2. Overview of the architecture of our new TLS interception proxy.

by capturing and presenting network traffic from a network interface or another libpcap input source. It can also be used to decrypt traffic from captured network traces. This can be done by setting the *(Pre)-Master-Secret log file* in the Wireshark preferences to a file which contains the master secrets in Network Security Services (NSS)[5] *Key Log Format*.[6] This file contains the session secrets and can be provided directly by the browser or by tools like the aforementioned mitmdump by setting the `KEYLOGFILE` environment variable when running the program. Wireshark is able to read this log file and to dissect TLS and possible layers on top of that. The dissected data can also be exported in various text formats. However, if the chosen export format is libpcap the packets are exported encrypted as they have been recorded “on the wire”. This means that the decrypted payload is not part of libpcap packets since the decryption happens after the packet is handed over from the operating system to the TLS library.

III. TLS Interception Proxy

TLS is a cryptographic protocol suite providing confidentiality and data integrity between two communication parties [7]. If implemented correctly, TLS makes it impossible to read the application layer payload by an observer which is able to intercept the traffic between the two TLS parties. Nevertheless, there are use cases where network operators want to inspect TLS traffic. A typical example is to avoid malicious code entering the corporate network.

Thus, network operators came up with TLS interception proxies that basically carry out legitimate man in the middle attacks. Legitimate in this case means that the clients are informed about the interception of their traffic and agree to install the Certificate Authority (CA) certificate of the interception proxy. If this is fulfilled it has to be ensured that all the traffic goes over the interception proxy. Figure 1 sketches this scenario. In contrast to scenario (a), where a client connects directly to a web server, in scenario (b) the traffic is routed over the interception proxy. This proxy establishes one TLS connection to the client and another TLS connection to the web server. This way the proxy is able to read the application layer payload in clear text. For the client the proxy behaves as being the web server, this requires the proxy to generate on the fly a valid certificate for the requested resource. Since the client has installed the CA certificate of the proxy the client will accept the generated certificate. The proxy acts as a TLS client at one connection and as a TLS server at the other. The incoming data at one connection is duplicated and forwarded to the respective other.

The operation of the proxy between the client and the server, which basically interrupts a connection, comes at a cost. The client’s assumptions about the TLS connection are no longer valid. One of these invalid assumptions can be the cryptographic trust in the connection. The client assumes to talk directly to the server and thus assumes to have knowledge about the cryptographic methods that are used within this TLS connection. This is not the case anymore as the interception proxy might, as a first example, decide to use weaker cryptographic protocols, or, as a second example, accept broken certificates from the server. It has to be made sure by the operators of the interception proxy that the standards expected by the clients are fulfilled.

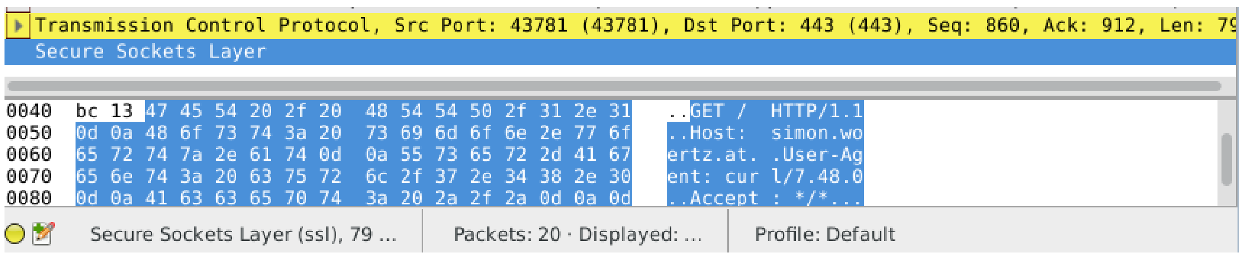
IV. Architecture

From an abstract point of view, our interception proxy listens to the configured interface, intercepts all TLS packets going over it and forwards all other traffic without touching it. The program consists mainly of two threads which are executed concurrently. The incoming TLS packets are duplicated and then fed to both threads. Figure 2 gives an overview of this architecture. In a nutshell, the upper thread manages the TLS connections in both directions and only cares about the transport layer. Additionally it hands over the master secret to the lower layer. The function of the lower layer is to reassemble out-of-order packets, to decrypt the TLS encrypted application layer payload, and to export all data.

In more detail, the upper layer first waits for an incoming connection, intercepts it and then initiates an additional TLS connection to the originally intended destination. In this work we call the incoming connection “downstream” connection and the outgoing one the “upstream” connection. To manage the TLS connections via a socket interface, the `boost.asio` library is used. In the upstream connection the interception proxy acts as the client endpoint whereas in the downstream connection it acts as the server endpoint. Therefore, the proxy has to authenticate itself to the client in the downstream connection. This is done by waiting until the original server authenticates itself to the proxy and then reissuing a copy of

[5]NSS is a set of cross-platform security libraries used by e.g., Firefox and Chrome.

[6]https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format



```
Transmission Control Protocol, Src Port: 43781 (43781), Dst Port: 443 (443), Seq: 860, Ack: 912, Len: 79
  Secure Sockets Layer

0040  bc 13 47 45 54 20 2f 20  48 54 54 50 2f 31 2e 31   ..GET /  HTTP/1.1
0050  0d 0a 48 6f 73 74 3a 20  73 69 6d 6f 6e 2e 77 6f   ..Host:  simon.wo
0060  65 72 74 7a 2e 61 74 0d  0a 55 73 65 72 2d 41 67   ertz.at. .User-Ag
0070  65 6e 74 3a 20 63 75 72  6c 2f 37 2e 34 38 2e 30   ent: cur l/7.48.0
0080  0d 0a 41 63 63 65 70 74  3a 20 2a 2f 2a 0d 0a 0d   ..Accept : */*...
```

Secure Sockets Layer (ssl), 79 ...
Packets: 20 · Displayed: ...
Profile: Default

Fig. 3. Screenshot of Wireshark presenting a libpcap dump of our interception proxy. The SSL content is shown in clear text.

the received certificate. Meanwhile the downstream connection is stalled. By reissuing (digitally signing) a certificate the proxy operates as a CA.

Instead of just forwarding incoming packets (like a router would do), the intercepting proxy has to maintain two connections (sockets) for every incoming TLS connection. All the data arriving at the upstream connection is forwarded to the corresponding downstream connection and vice-versa. The only exception to this is the certificate which has to be adapted before it can be handed over. Due to this approach, the proxy is in possession of all the necessary crypto material for all connections it maintains.

The lower part shown in Figure 2 is responsible for recording, reordering, dissecting, decrypting and exporting incoming packets. The library `libtins`[7] (a C++ wrapper for `libpcap`) is used to record and reorder (if necessary) incoming packets. Reordering is necessary since TLS is usually layered on top of TCP as a transport protocol. A network packet sent over a large network can take different routes on the way from the source to the destination. Therefore, packets can arrive out-of-order and we have to reorder them.

The lower part shown in Figure 2 is also responsible of recording the “client random” and “server random” information, which are exchanged during the TLS Handshake. Together with the “master secret”, which is stored during the handshakes performed in the “upper” part, it is possible to calculate all the necessary crypto material to decrypt the received TLS records. “client random”, “server random”, and the “master secret” are stored together and are identified by their source and destination IP addresses and TCP ports. This information is stored in a shared memory and is the only way the two threads communicate with each other.

The final step is to combine decrypted TLS records with the lower layers of the recorded packets. This is done by exchanging the payload of a TCP packet which originally contained an encrypted TLS record by the decrypted TLS application data. The exchange of the payload and the export of the packets into libpcap format files is done using libtins.

Exporting the decrypted payload atop of the original lower layers has the downside that network monitoring tools (e.g., Wireshark) will complain when loading the exported libpcap file because if a header contains checksums like Cyclic Redundancy Check (CRC), they will obviously be wrong. This is because the application payload now is decrypted and thus can not match the CRC checksum of the encrypted payload.

[7]https://libtins.github.io/

Until now only the most used TLS cipher suites have been implemented, but more will follow. The source code of our interception proxy is available at https://bitbucket.org/swoertz/master-project.

V. Demonstration Setup

To demonstrate our TLS interception proxy we will use a laptop with Internet access acting as proxy and a Raspberry Pi as TLS client. The proxy (on the laptop) will export the traffic in libpcap format to another network monitoring tool that will display the encrypted TLS records.

Figure 3 shows a sample libpcap output of our interception proxy. While Wireshark displays the packet content as being SSL it can be seen that the payload is readable in clear text.

VI. Conclusion

In this demo we propose a TLS interception proxy that is not only able to export the decrypted application layer of TLS encrypted connections in real-time in the well known libpcap format. The exported data also include lower network layer information. This allows third party tools like IDS not only to apply DPI methods on it but also analyze the lower layers.

References

[1] S. Dyllan, H. Dahimene, P. Wright, and P. Xiao, “Analysis of HTTP and HTTPS Usage on the University Internet Backbone Links,” *Journal of Industrial and Intelligent Information*, vol. 2, no. 1, pp. 67–70, Mar. 2014.

[2] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munaf‘o, K. Papagiannaki, and P. Steenkiste, “The Cost of the ”S” in HTTPS,” in *10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT 2014)*. Sydney, Australia: ACM, Dec. 2014, pp. 133–140.

[3] M. Roesch, “Snort: Lightweight Intrusion Detection for Networks,” in *13th USENIX Conference on System Administration (LISA 1999)*, Seattle, WA, Nov. 1999, pp. 229–238.

[4] V. Paxson, “Bro: A System for Detecting Network Intruders in Real-Time,” *Elsevier Computer Networks*, vol. 31, no. 23-24, pp. 2435–2463, Dec. 1999.

[5] R. T. Lampert, C. Sommer, G. Münz, and F. Dressler, “Vermont - A Versatile Monitoring Toolkit for IPFIX and PSAMP,” in *IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation (MonAM 2006)*. Tübingen, Germany: IEEE, September 2006, pp. 62–65.

[6] F. Erlacher, W. Estgfaeller, and F. Dressler, “Improving Network Monitoring Through Aggregation of HTTP/1.1 Dialogs in IPFIX,” in *41st IEEE Conference on Local Computer Networks (LCN 2016)*. Dubai, UAE: IEEE, November 2016.

[7] T. Dierks, “The Transport Layer Security (TLS) Protocol Version 1.2,” IETF, Tech. Rep. 5246, Oct. 2015.