Live Detection and Analysis of HTTPS Interceptions

Tobias Brunnwieser Oliver Gasser Sree Harsha Totakura Georg Carle
Technical University of Munich
{brunnwie,gasser,totakura,carle}@net.in.tum.de

ABSTRACT

In 2016, field tests by Google, Facebook, and Mozilla revealed compatibility issues with the new TLS 1.3 protocol. The issues arose from HTTPS interception middleboxes unable to cope with the new TLS handshake. While most interception systems have benevolent intent, research has shown that they can negatively impact connection security.

In this paper, we present a system to passively detect HTTPS interceptions on live traffic and analyze the impact on connection security. We design and implement a component to continuously learn new browser versions and cipher suites. Our analysis on one month of HTTPS connections to a popular website shows that on average, 4.2% of all observed connections are intercepted. We could classify those interceptions into 577 unique patterns.

1. INTRODUCTION

HTTPS is one of the important building blocks of a secure Web. Its security layer, TLS, not only provides encryption, authentication, and integrity protection, but also prevents man-in-the-middle attacks through the use of certificates. These are issued and signed by Certificate Authorities (CAs). Trusted CA certificates are then included in a browser's root store which allows it to check whether a certificate was issued by a trusted CA.

HTTPS Interception: While this system effectively prevents man-in-the-middle attacks with forged certificates, it also blocks legitimate inspection of traffic, e.g. from antivirus software. To solve this such software generates a new root certificate and installs it locally at the client. When the client connects to a web site over HTTPS, a new certificate for the requested domain is dynamically generated and signed by the installed root CA. This way, the client is fooled to trust the forged certificate. The interception device or software can then inspect the cleartext traffic for malicious or unwanted communication. Previous work shows that such interceptions can occur anywhere from less than 1% up to over 10% of observed connections [1, 2, 3].

TLS Fingerprinting: When establishing an HTTPS connection, client and server need to negotiate parameters for the underlying TLS connection. This includes the selection of a so called *cipher suite* to be used for encryption, integrity checking, and authentication. A client advertises its preference in the first handshake message of a TLS connection, namely the `ClientHello` message. Since these settings affect user security, most clients use the same parameters in every `ClientHello` message. We can use this information to create a unique TLS *fingerprint* for each client [1].

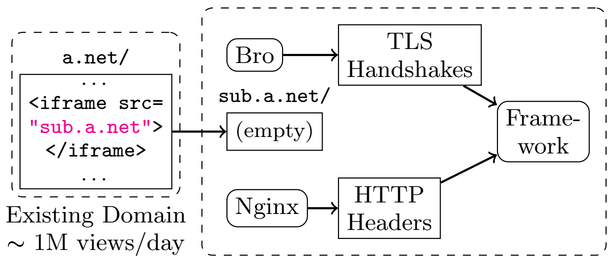

Figure 1: Data collection architecture.

2. DATA COLLECTION

In the following we describe our data collection setup (see Figure 1). We embed an `iframe` on an existing web site which points to a subdomain served by a dedicated server with its own certificate. On this server we log every request in detail.

On the same machine we use Bro [5] for collecting TLS handshakes. We develop a Bro script that records all fields available in the `ClientHello` message of every HTTPS connection. We ensure to collect only information from newly established connections which are used to transfer data.

Finally, we combine web server logs and Bro logs by matching timestamp, remote IP address, and port and feed this information into our detection and analysis framework.

This architecture requires only minimal changes to existing infrastructure with production traffic and operates completely isolated from it.

Ethical Considerations: We follow measurement best practices [4] and do not store user identifiable information.

3. DETECTION AND LEARNING

Our detection approach builds upon prior work from Durumeric et al. [1]. They use a *mismatch* between a client's user agent (UA) and its TLS handshake fingerprint as an indicator for interception. We extend this approach to perform live detection, add a learning component for fingerprints, and analyze found interceptions in-depth.

Fingerprint Variants: Initially, we do not know whether the TLS fingerprint of a client is legitimate or is altered due to interception. Therefore, we collect all fingerprints of each client, which we call fingerprint *variants*. We periodically re-evaluate which variant is trusted and may therefore be used for detection.

Evaluating Trust: Our approach is based on the assumption that the majority of clients is not intercepted. Therefore, we record the number of distinct IP subnets observed for every fingerprint variant, which we call *subnet diversity*. If a variant surpasses a threshold for subnet diversity, we regard this variant as trusted. Our experiments show that a variant can be trusted if it appears in 15% or more of all observed subnets, but in at least 3 subnets. We perform this check at regular intervals for each fingerprint and count the number of trusted variants. Interception detection can only be performed if a single variant is trusted. Cases where no variant is trusted indicate that the fingerprint is observed from few clients with low subnet diversity. Multiple trusted variants indicate that the parameters for this client's fingerprint are not stable (e.g. random cipher suite ordering) and need manual investigation.

Processing Connections: For every observed connection, we look up the corresponding fingerprint variants by its UA and distinguish the following three cases. First, if the observed fingerprint matches the single trusted variant, the connection is categorized as not intercepted. Second, if we match any other variant except the single trusted one, the connection is categorized as intercepted. Third, in any other case no conclusive decision can be made.

4. INTERCEPTION ANALYSIS

In addition to live interception detection, we propose several analyses to asses the security impact of HTTPS interception. We perform these analyses on the sever as well on the client, the latter by delivering JavaScript in the `iframe`.

Identifying the Intercepting System: To identify the intercepting system we compare its TLS fingerprint against publicly available databases [1]. In addition, we plan to use `nmap` and inspect HTTP headers for identification purposes.

Certificate Validation: Intercepting middleboxes need to validate the certificates presented by web servers. Therefore we create a set of unique test domains, each with a certificate that has a unique flaw (e.g. domain name mismatch). We instruct clients to test connectivity to those domains. Any successful connection indicates a failure of the middlebox in validating server certificates.

Content Modification: We use the approach proposed by Reis et al. [6] to detect web content modification by intercepting middleboxes. Therefore, we embed the test script in the `iframe` and add content to the downloaded web page which can be checked for modification.

5. FIRST RESULTS

Our results are based on measurements performed between December 12, 2017 and March 2, 2018. We observed 2.17M HTTPS connections from 5749 unique browser versions with 9488 different variants in total. By inspecting the UA, we found that 65.5% of all connections were made by mobile devices and 34.5% by desktop clients.

From all connections, we find that 4.2% are intercepted and are 9.3% undecided. For mobile devices, we observe significantly lower interception rates (0.8%), while desktop clients tend to be affected more (10.6%). Figure 2 shows a time series of rates of intercepted and undecidable clients. For each point in the diagram we aggregated the number of intercepted and undecidable connections within 24 hours. In total, we could classify 577 unique interceptions.

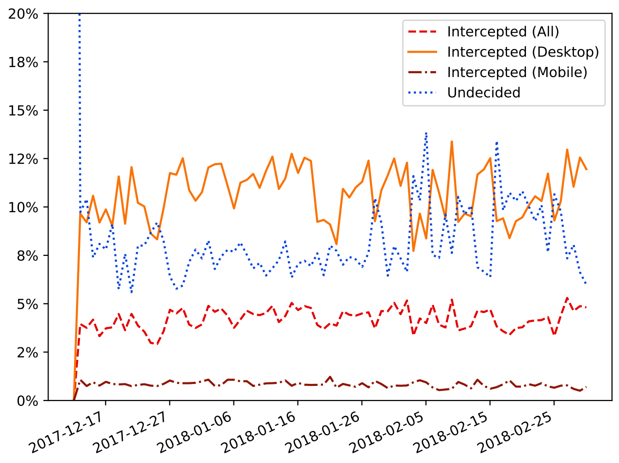

Figure 2: Time series of the interception rates for all clients, mobile clients, and desktop clients.

6. CONCLUSION AND FUTURE WORK

In this paper we presented an approach to perform live HTTPS interception. We created a dynamic fingerprint learning component, which eliminates the need for manually assembling a database of trusted fingerprints. Furthermore, we designed our approach to be simple to deploy and it can be integrated into multiple web pages at once and therefore eases the process of acquiring data. In the future, we plan to perform a long term study of HTTPS interception and to implement in-depth analysis of intercepted clients.

Acknowledgments: This work was supported by PPRO Financial Ltd.; we want to especially thank Roland Bock and Andrew Edem. We thank Florian Wohlfart for the valuable discussions. This work was partially funded by the German Federal Ministry of Education and Research under project X-Check, grant 16KIS0530, and project DecADe, grant 16KIS0538.

7. REFERENCES

[1] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Nailey, J. A. Halderman, and V. Paxson. The Security Impact of HTTPS Interception. In *NDSS'17*.

[2] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson. Analyzing Forged SSL Certificates in the Wild. In *S&P'14*.

[3] M. O'Neill, S. Ruoti, K. Seamons, and D. Zappala. TLS Proxies: Friend or Foe? In *IMC'16*.

[4] C. Partridge and M. Allman. Ethical Considerations in Network Measurement Papers. *Communications of the ACM*, 2016.

[5] V. Paxson. Bro: a System for Detecting Network Intruders in Real-Time. *Computer Networks*, 1999.

[6] C. Reis, S. D. Gribble, T. Kohno, and N. C. Weaver. Detecting In-Flight Page Changes with Web Tripwires. In *NSDI'08*.