The Security Impact of HTTPS Interception

Zakir Durumeric[*][∨], Zane Ma[†], Drew Springall[*], Richard Barnes[‡], Nick Sullivan[§],
Elie Bursztein[¶], Michael Bailey[†], J. Alex Halderman[*], Vern Paxson[‖][∨]

[*] University of Michigan [†] University of Illinois Urbana-Champaign [‡] Mozilla [§] Cloudflare
[¶] Google [‖] University of California Berkeley [∨] International Computer Science Institute

Abstract—**As HTTPS deployment grows, middlebox and antivirus products are increasingly intercepting TLS connections to retain visibility into network traffic. In this work, we present a comprehensive study on the prevalence and impact of HTTPS interception. First, we show that web servers can detect interception by identifying a mismatch between the HTTP User-Agent header and TLS client behavior. We characterize the TLS handshakes of major browsers and popular interception products, which we use to build a set of heuristics to detect interception and identify the responsible product. We deploy these heuristics at three large network providers: (1) Mozilla Firefox update servers, (2) a set of popular e-commerce sites, and (3) the Cloudflare content distribution network. We find more than an order of magnitude more interception than previously estimated and with dramatic impact on connection security. To understand why security suffers, we investigate popular middleboxes and client-side security software, finding that nearly all reduce connection security and many introduce severe vulnerabilities. Drawing on our measurements, we conclude with a discussion on recent proposals to safely monitor HTTPS and recommendations for the security community.**

I. Introduction

When it comes to HTTPS, the security community is working at cross purposes. On the one hand, we are striving to harden and ubiquitously deploy HTTPS in order to provide strong end-to-end connection security [5], [20], [22], [23], [34], [51]. At the same time, middlebox and antivirus products increasingly intercept (i.e., terminate and re-initiate) HTTPS connections in an attempt to detect and block malicious content that uses the protocol to avoid inspection [6], [12], [15], [27]. Previous work has found that some specific HTTPS interception products dramatically reduce connection security [7], [12], [58]; however, the broader security impact of such interception remains unclear. In this paper, we conduct the first comprehensive study of HTTPS interception in the wild, quantifying both its prevalence in traffic to major services and its effects on real-world security.

We begin by introducing a novel technique for passively detecting HTTPS interception based on handshake characteristics. HTTPS interception products typically function as transparent proxies: they terminate the browser's TLS connection, inspect the HTTP plaintext, and relay the HTTP data over a new TLS connection to the destination server. We show that web servers can detect such interception by identifying a *mismatch* between the HTTP User-Agent header and the behavior of the TLS client. TLS implementations display varied support (and preference order) for cipher suites, extensions, elliptic curves, compression methods, and signature algorithms. We characterize these variations for major browsers and popular interception products in order to construct heuristics for detecting interception and identifying the responsible product.

Next, we assess the prevalence and impact of HTTPS interception by applying our heuristics to nearly eight billion connection handshakes. In order to avoid the bias inherent in any single network vantage point, we analyzed connections for one week at three major Internet services: (1) Mozilla Firefox update servers, (2) a set of popular e-commerce websites, and (3) the Cloudflare content distribution network. These providers serve different types of content and populations of users, and we find differing rates of interception: 4.0% of Firefox update connections, 6.2% of e-commerce connections, and 10.9% of U.S. Cloudflare connections were intercepted. While these rates vary by vantage point, all are more than an order of magnitude higher than previous estimates [27], [46].

To quantify the real-world security impact of the observed interception, we establish a grading scale based on the TLS features advertised by each client. By applying the metric to unmodified browser handshakes and to the intercepted connections seen at each vantage point, we calculate the change in security for intercepted connections. While for some older clients, proxies increased connection security, these improvements were modest compared to the vulnerabilities introduced: 97% of Firefox, 32% of e-commerce, and 54% of Cloudflare connections that were intercepted became less secure. Alarmingly, not only did intercepted connections use weaker cryptographic algorithms, but 10–40% advertised support for known-broken ciphers that would allow an active man-in-the-middle attacker to later intercept, downgrade, and decrypt the connection. A large number of these severely broken connections were due to network-based middleboxes rather than client-side security software: 62% of middlebox connections were less secure and an astounding 58% had severe vulnerabilities enabling later interception.

Finally, we attempt to understand why such a large number of intercepted connections are vulnerable by testing the security of a range of popular corporate middleboxes, antivirus products, and other software known to intercept TLS. The default settings for eleven of the twelve corporate middleboxes we evaluated expose connections to known attacks, and five introduce severe vulnerabilities (e.g., incorrectly validate certificates). Similarly, 18 of the 20 client-side security products we tested reduce

NDSS'17, 26 February–1 March, 2017, San Diego, CA, USA
http://dx.doi.org/10.14722/ndss.2017.23456

connection security, and half introduce severe vulnerabilities. In some cases, manufacturers attempted to customize libraries or re-implement TLS, introducing negligent vulnerabilities. In other cases, products shipped with libraries that were years out of date. Across the board, companies are struggling to correctly deploy the base TLS protocol, let alone implement modern HTTPS security features.

Our results indicate that HTTPS interception has become startlingly widespread, and that interception products as a class have a dramatically negative impact on connection security. We hope that shedding light on this state of affairs will motivate improvements to existing products, advance work on recent proposals for safely intercepting HTTPS [26], [38], [44], [54], and prompt discussion on long-term solutions.

II. Background

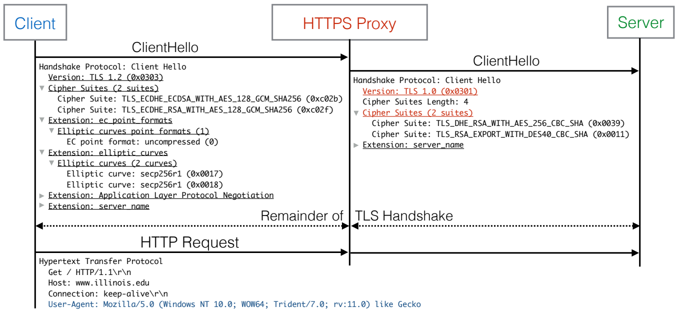
In this section, we provide a brief background on HTTPS interception and describe the aspects of HTTP and TLS that are relevant to our fingerprinting techniques. We refer the reader to RFC 5280 [14] for a detailed description of TLS.

A. TLS Interception

Client-side software and network middleboxes that inspect HTTPS traffic operate by acting as transparent proxies. They terminate and decrypt the client-initiated TLS session, analyze the inner HTTP plaintext, and then initiate a new TLS connection to the destination website. By design, TLS makes such interception difficult by encrypting data and defending against man-in-the-middle attacks through certificate validation, in which the client authenticates the identity of the destination server and rejects impostors. To circumvent this validation, local software injects a self-signed CA certificate into the client browser's root store at install time. For network middleboxes, administrators will similarly deploy the middlebox's CA certificate to devices within their organization. Subsequently, when the proxy intercepts a connection to a particular site, it will dynamically generate a certificate for that site's domain name signed with its CA certificate and deliver this certificate chain to the browser. Unless users manually verify the presented certificate chain, they are unlikely to notice that the connection has been intercepted and re-established.[1]

B. TLS Feature Negotiation

TLS clients and servers negotiate a variety of protocol parameters during a connection handshake [14]. In the first protocol message, Client Hello, the client specifies what TLS version and features it supports. It sends ordered lists of cipher suites, compression methods, and extensions — which themselves frequently contain additional parameters, such as supported elliptic curves and signature algorithms. The server then selects a mutually agreeable choice from each list of options. This extensibility facilitates the continuing evolution of features and provides adaptability in the wake of new attacks.



```
Handshake Protocol: Client Hello
  Version: TLS 1.2 (0x0303)
▼ Cipher Suites (2 suites)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
▼ Extension: ec_point_formats
  ▼ Elliptic curves point formats (1)
      EC point format: uncompressed (0)
▼ Extension: elliptic_curves
  ▼ Elliptic curves (2 curves)
      Elliptic curve: secp256r1 (0x0017)
      Elliptic curve: secp256r1 (0x0018)
▶ Extension: Application Layer Protocol Negotiation
▶ Extension: server_name
```

```
Handshake Protocol: Client Hello
  Version: TLS 1.0 (0x0301)
  Cipher Suites Length: 4
▼ Cipher Suites (2 suites)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
    Cipher Suite: TLS_RSA_EXPORT_WITH_DES40_CBC_SHA (0x0011)
▶ Extension: server_name
```

```
Hypertext Transfer Protocol
  Get / HTTP/1.1\r\n
  Host: www.illinois.edu
  Connection: keep-alive\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
```

Fig. 1: **HTTPS Interception** — Products monitor HTTPS connections by acting as transparent proxies that terminate the browser TLS session, inspect content, and establish a new connection to the destination server. These proxies use different TLS libraries than popular browsers, which allows us to detect interception by identifying a mismatch between the HTTP User-Agent header and TLS client behavior.

As of early 2016, there exist more than 340 cipher suites, 36 elliptic curves, 3 elliptic curve point formats, 28 signature algorithms, and 27 extensions that clients can advertise [29], [30]. In practice, browsers and security products use varying TLS libraries and advertise different handshake parameters. As we will show in Section III, these characteristic variations allow us to uniquely identify individual TLS implementations based on their handshakes.

C. HTTP User-Agent Header

The HTTP protocol allows the client and server to pass additional information during a connection by including header fields in their messages. For example, the client can include the `Accept-Charset: utf-8` header to indicate that it expects content to be encoded in UTF-8. One standard client header is the *User-Agent* header, which indicates the client browser and operating system in a standardized format. There has been significant prior study on User-Agent header spoofing. These studies have largely found that end users do not spoof their own User-Agent header [18], [45], [61]. For example, Eckersley found that only 0.03% of connections with a Firefox User-Agent supported features unique to Internet Explorer, indicating spoofing [18]. Fingerprinting studies commonly trust the User-Agent string [40], and we follow suit in this work.

III. TLS Implementation Heuristics

Our methodology for identifying interception is based on detecting a *mismatch* between the browser specified in the HTTP User-Agent header and the cryptographic parameters advertised during the TLS handshake (Figure 1). In this section, we characterize the handshakes from popular browsers and develop heuristics that determine whether a TLS handshake is consistent with a given browser. We then go on to fingerprint the handshakes produced by popular security products in order to identify the products responsible for interception in the wild.

A. Web Browsers

We captured the TLS handshakes generated by the four most popular browsers: Chrome, Safari, Internet Explorer, and Firefox [57]. To account for older versions, differing

[1] Contrary to widespread belief, public key pinning [19] — an HTTPS feature that allows websites to restrict connections to a specific key — does not prevent this interception. Chrome, Firefox, and Safari only enforce pinned keys when a certificate chain terminates in an authority shipped with the browser or operating system. The extra validation is skipped when the chain terminates in a locally installed root (i.e., a CA certificate installed by an administrator) [34]. Internet Explorer and Edge do not support key pinning [39].

operating systems, and varying mobile hardware, we generated and captured handshakes in different environments using BrowserStack [4], a cloud service that provides developers with a variety of virtual machines for testing websites.[2]

We analyzed the non-ephemeral parameters advertised in the TLS handshakes, finding that each browser family selects a unique set of options, and that these options differ from those used by both common libraries (e.g., OpenSSL) and popular interception products.[3] However, while each browser, library, and security product produces a unique Client Hello message, the parameters selected by browsers are not statically defined. Instead, browsers alter their behavior based on hardware support, operating system updates, and user preferences.

Instead of generating all possible permutations, we analyzed *when* browsers select different parameters and developed a set of heuristics that determine whether a specific handshake *could have* been generated by a given browser. For example, none of the four browsers have ever supported the TLS Heartbeat extension [53]. If a browser connection advertises its support, we know that the session was intercepted. On the other hand, despite the fact that all four browsers have default support for AES-based ciphers, the lack of these ciphers does not indicate interception, because browsers allow users to disable specific cipher suites. This methodology has the advantage of excluding false positives that arise from uncommon user configurations. However, it can yield false negatives if a proxy copies TLS parameters from the original client connection.

We describe the heuristics for each browser below:

Mozilla Firefox Firefox was the most consistent of the four browsers, and by default, each version produces a nearly identical Client Hello message regardless of operating system and platform. All parameters, including TLS extensions, ciphers, elliptic curves, and compression methods are predetermined and hard-coded by the browser. Users can disable individual ciphers, but they cannot add new ciphers nor reorder them. To determine whether a Firefox session has been intercepted, we check for the presence and order of extensions, cipher suites, elliptic curves, EC point formats, and handshake compression methods. Mozilla maintains its own TLS implementation, Mozilla Network Security Services (NSS) [42]. NSS specifies extensions in a different order than the other TLS libraries we tested, which allows it to be easily distinguished from other implementations. The library is unlikely to be directly integrated into proxies because it is seldom used in server-side applications.

Google Chrome Chrome was one of the most challenging browsers to fingerprint because its behavior is dependent on hardware support and operating system. For example, Chrome prioritizes ChaCha-20-based ciphers on Android devices that lack hardware AES acceleration [10] and Chrome on Windows XP does not advertise support for ECDSA keys at all. These optimizations result in several valid cipher and extension orderings for each version of Chrome, and furthermore, users can disable individual cipher suites. We check for ciphers and extensions that Chrome is known to not support, but do not check for the inclusion of specific ciphers or extensions, nor do we validate their order. When appropriate, we check the presence and order of elliptic curves, compression methods, and EC point formats. We note that because Chrome uses BoringSSL, an OpenSSL fork, connections have a similar structure to OpenSSL. However, Chrome supports considerably fewer options, including a subset of the default extensions, ciphers, and elliptic curves advertised by OpenSSL.

Microsoft Internet Explorer and Edge Internet Explorer is less consistent than the other browsers for two reasons: (1) administrators can enable new ciphers, disable default ciphers, and arbitrarily reorder cipher suites through Windows Group Policy and registry changes, and (2) IE uses the Microsoft SChannel library, an OS facility which behaves differently depending on *both* Windows updates and browser version. Because of this additional flexibility, it is hard to rule out handshakes that include outdated ciphers, so we introduce a third categorization, *unlikely*, which we use to indicate configurations that are possible but improbable in practice (e.g., including an export-grade cipher suite). We note that minor OS updates alter TLS behavior, but are not indicated by the HTTP User-Agent header. We mark behavior consistent with any OS update as valid. SChannel connections can by uniquely identified because SChannel is the only TLS library we tested that includes the OCSP status request extension before the supported groups and EC point formats extensions.

Apple Safari Apple Safari ships with its own TLS implementation, Apple Secure Transport, which does not allow user customization. The order of ciphers and extensions is enforced in code. While extension order does not vary, the NPN, ALPN, and OCSP stapling extensions are frequently excluded, and the desktop and mobile versions of Safari have different behavior. One unique aspect of Secure Transport is that it includes the TLS_EMPTY_RENEGOTIATION_INFO_SCSV cipher first, whereas the other libraries we investigated include the cipher last. Similar to Microsoft, Apple has changed TLS behavior in minor OS updates, which are not indicated in the HTTP User-Agent header. We allow for any of the updates when validating handshakes, and we check for the presence and ordering of ciphers, extensions, elliptic curves, and compression methods.

B. *Fingerprinting Security Products*

While our heuristics enable us to detect *when* interception is occurring, they do not indicate *what* intercepted the connection. To identify products used in the wild, we installed and fingerprinted the Client Hello messages generated by well-known corporate middleboxes (Figure 3), antivirus software (Figure 4), and other client-side software previously found to intercept connections (e.g., SuperFish [8]). In this section, we describe these products.

We generate a fingerprint for each product by hashing the non-ephemeral parameters advertised in the Client Hello message (i.e., version, ciphers, extensions, compression methods, elliptic curves, and signature methods). When we detect that a handshake is inconsistent with the indicated browser, we check for an exact match with any of the fingerprints we collected.

[2] We analyzed Chrome 34–50 and Firefox 3–46 on Windows XP, 7, 8, 8.1, and 10 and Mac OS X Snow Leopard, Lion, Mountain Lion, Mavericks, Yosemite, and El Capitan. We additionally captured handshakes from Apple Safari 5–9, Internet Explorer 6–11, Microsoft Edge, and the default Android browser on Android 4.0–5.0. In total, we collected 874 fingerprints.

[3] We compare the presence and order of cipher suites, extensions, compression methods, elliptic curves, signature algorithms, and elliptic curve point formats.

This strategy has several potential weaknesses. First, multiple products could share a single fingerprint. This seems particularly likely given that developers are likely to use one of a small number of popular TLS libraries (e.g., OpenSSL). Second, if products allow customization, then fingerprints of the default configuration will not match these customized versions.

Surprisingly, we find that none of the browsers nor the 104 products we manually investigated shared fingerprints, except for eight pieces of unwanted software that all use the Komodia Redirector SDK [32]. In other words, these fingerprints uniquely identify a single product. None of the client-side security products we tested allow users to customize TLS settings. However, many of the corporate middleboxes allow administrators to specify custom cipher suites. In this situation, we would be able to detect that interception is occurring, but not identify the responsible product.

Middleboxes and Corporate Proxies Nearly every major networking hardware manufacturer — including Barracuda, Blue Coat, Cisco, and Juniper — produces middleboxes that support "SSL Inspection". These devices allow organizations to intercept TLS traffic at their network border for analysis, content filtering, and malware detection. In March 2015, Dormann documented products from nearly 60 manufacturers that advertise this functionality [15]. We configured and fingerprinted twelve appliance demos from well-known manufacturers (e.g., Cisco and Juniper) and anecdotally popular companies (e.g., A10 and Forcepoint), per Figure 3. We note one conspicuous absence: ZScaler SSL Inspection. ZScaler provides a cloud-based SSL inspection service, but did not provide us with a trial or demo.

Antivirus Software We installed, tested, and fingerprinted popular antivirus products based on the software documented by de Carné de Carnavalet and Mannan [12], products previously found to be intercepting connections [27], [46], and a report of popular antivirus products [47]. Products from twelve vendors inject a new root certificate and actively intercept TLS connections.[4] We list the products that intercept connections in Figure 4.[5]

Unwanted Software and Malware Motivated by recent reports of unwanted software intercepting TLS connections, we fingerprinted the Komodia SDK [32], which is used by Superfish, Qustodio, and several pieces of malware [8], [56], and the NetFilter SDK [1], which is used by PrivDog.

We tested products in January–March, 2016. We are publishing our browser heuristics and product fingerprints at https://github.com/zakird/tlsfingerprints.

IV. Measuring TLS Interception

We measured global interception rates by deploying our heuristics at three network vantage points: Mozilla Firefox update servers, a set of popular e-commerce websites, and the Cloudflare content distribution network. We observed 7.75 billion TLS handshakes across the three networks. By deploying the heuristics on different networks, we avoid the bias inherent of any single vantage point. However, as we will discuss in the next section, we find varying amounts of interception and abuse on each network. Below, we describe each perspective in detail:

Vantage Point	% HTTPS Connections Intercepted		
	No Interception	Likely	Confirmed
Cloudflare	88.6%	0.5%	10.9%
Firefox	96.0%	0.0%	4.0%
E-commerce	92.9%	0.9%	6.2%

Fig. 2: **Detecting Interception** — We quantify HTTPS interception at three major Internet services. We estimate that 5–10% of connections are intercepted.

Firefox Update Servers Firefox browsers routinely check for software updates by retrieving an XML document from a central Mozilla server over HTTPS. This check uses Firefox's standard TLS library (Mozilla NSS) and occurs every 24 hours while the browser is running and on browser launch if the last update occurred more than 24 hours prior. We used Bro [49] to monitor connections to `aus5.mozilla.org` — the update server used by Firefox versions 43–48 — between February 14–26, 2016. During this period, we observed 4.36 billion connections from 45K ASes and 243 of the 249 ISO-defined countries. Because we collected traffic using an on-path monitor instead of on the web server, we do not have access to the HTTP User-Agent header. However, only specific versions of Firefox are configured to connect to the server. Instead of looking for a mismatch with the HTTP User-Agent, we look for a mismatch between TLS handshake and any of the Firefox versions configured to connect to the server. There is no user-accessible content available on the site and there should be negligible other traffic. This vantage point provides one of the cleanest perspectives on clients affected by TLS interception. However, it only provides data for Firefox, one of the browsers believed to be least affected by client-side interception software [12].

Popular E-commerce Sites During two weeks in August and September 2015, a set of popular e-commerce sites hosted JavaScript that loaded an invisible pixel from an external server that recorded the raw TLS Client Hello, HTTP User-Agent string, and client HTTP headers. This perspective sees traffic from all browsers, but may suffer from falsified User-Agent headers. However, because the measurement required JavaScript execution, the dataset excludes simple page fetches. The sites have an international presence, but the connections we observe are likely skewed towards desktop users because the e-commerce provider has popular mobile applications. The dataset has the added benefit that it contains HTTP headers beyond User-Agent, which allow another avenue for detecting interception: looking for proxy related headers (e.g., `X-Forwarded-For` and `X-BlueCoat-Via`) and the modifications documented by Weaver et al. [60].

Cloudflare Cloudflare is a popular CDN and DDoS protection company that serves approximately 5% of *all* web traffic [25]. Cloudflare provides these services by acting as a reverse proxy. Clients connect to one of Cloudflare's servers when accessing a website, which serve cached content or proxy

[4] We found that the following products did not intercept HTTPS: 360 Total, Ahnlabs V3 Internet Security, Avira AV 2016, Comodo Internet Security, F-Secure Safe, G DATA products, K7 Total Security, Malwarebytes, McAfee Internet Security, Microsoft Windows Defender, Norton Security, Panda Internet Security 2016, Security Symantec Endpoint Protection, Tencent PC Manager, Trend Micro Maximum Security 10, and Webroot SecureAnywhere.

[5] A preprint of this paper incorrectly listed G DATA products in Figure 4.

Product	Grade	Validates Certificates	Modern Ciphers	Advertises RC4	TLS Version	Grading Notes
A10 vThunder SSL Insight	F	✓	✓	Yes	1.2	Advertises export ciphers
Blue Coat ProxySG 6642	A*	✓	✓	No	1.2	Mirrors client ciphers
Barracuda 610Vx Web Filter	C	✓	✗	Yes	1.0	Vulnerable to Logjam attack
Checkpoint Threat Prevention	F	✓	✗	Yes	1.0	Allows expired certificates
Cisco IronPort Web Security	F	✓	✓	Yes	1.2	Advertises export ciphers
Forcepoint TRITON AP-WEB Cloud	C	✓	✓	No	1.2	Accepts RC4 ciphers
Fortinet FortiGate 5.4.0	C	✓	✓	No	1.2	Vulnerable to Logjam attack
Juniper SRX Forward SSL Proxy	C	✓	✗	Yes	1.2	Advertises RC4 ciphers
Microsoft Threat Mgmt. Gateway	F	✗	✗	Yes	SSLv2	No certificate validation
Sophos SSL Inspection	C	✓	✓	Yes	1.2	Advertises RC4 ciphers
Untangle NG Firewall	C	✓	✗	Yes	1.2	Advertises RC4 ciphers
WebTitan Gateway	F	✗	✓	Yes	1.2	Broken certificate validation

Fig. 3: **Security of TLS Interception Middleboxes**—We evaluate popular network middleboxes that act as TLS interception proxies. We find that nearly all reduce connection security and five introduce severe vulnerabilities. *Mirrors browser ciphers.

Product	OS	Browser MITM				Grade	Validates Certificates	Modern Ciphers	TLS Version	Grading Notes
		IE	Chrome	Firefox	Safari					
Avast …										
AV 11	Win	●	○	○		A*	✓	✓	1.2	Mirrors client ciphers
AV 11.7	Mac		●	●	●	F	✓	✓	1.2	Advertises DES
AVG …										
Internet Security 2015–6	Win	●	●	○		C	✓	✓	1.2	Advertises RC4
Bitdefender …										
Internet Security 2016	Win	●	●	●		C	✓	○	1.2	RC4, 768-bit D-H
Total Security Plus 2016	Win	●	●	●		C	✓	○	1.2	RC4, 768-bit D-H
AV Plus 2015–16	Win	●	●	●		C	✓	○	1.2	RC4, 768-bit D-H
Bullguard …										
Internet Security 16	Win	●	●	●		A*	✓	✓	1.2	Mirrors client ciphers
Internet Security 15	Win	●	●	●		F	✓	✗	1.0	Advertises DES
CYBERsitter …										
CYBERsitter 11	Win	●	●	●		F	✗	✗	1.2	No cert. validation, DES
Dr. Web …										
Security Space 11	Win	●	●	●		C	✓	○	1.2	RC4, FREAK
Dr. Web 11 for OS X	Mac		●	●	●	F	✓	✗	1.0	Export ciphers, DES, RC2
ESET …										
NOD32 AV 9	Win	●	●	●		F	○	○	1.2	Broken cert. validation
Kaspersky …										
Internet Security 16	Win	●	●	●		C	✓	✓	1.2	CRIME vulnerability
Total Security 16	Win	●	●	●		C	✓	✓	1.2	CRIME vulnerability
Internet Security 16	Mac		●	●	●	C	✓	✓	1.2	768-bit D-H
KinderGate …										
Parental Control 3	Win	●	●	●		F	○	✗	1.0	Broken cert. validation
Net Nanny …										
Net Nanny 7	Win	●	●	●		F	✓	✓	1.2	Anonymous ciphers
Net Nanny 7	Mac		●	●	●	F	✓	✓	1.2	Anonymous ciphers
PC Pandora …										
PC Pandora 7	Win	●	◐	◐		F	✗	✗	1.0	No certificate validation
Qustodio …										
Parental Control 2015	Mac		●	●	●	F	✓	✓	1.2	Advertises DES

Interception:	**Certificate Validation:**	**Modern Ciphers:**
○ No Interception (conn. allowed)	✗ No Validation	✗ No Support
◐ Connections Blocked	○ Broken Validation	○ Non-preferred Support
● Connections Intercepted	✓ Correct Validation	✓ Preferred Support

Fig. 4: **Security of Client-side Interception Software**—We evaluate and fingerprint popular antivirus and client-side security products, finding that products from twelve vendors intercept connections.[5] In all but two cases, products degrade TLS connection security. *Mirrors browser ciphers.

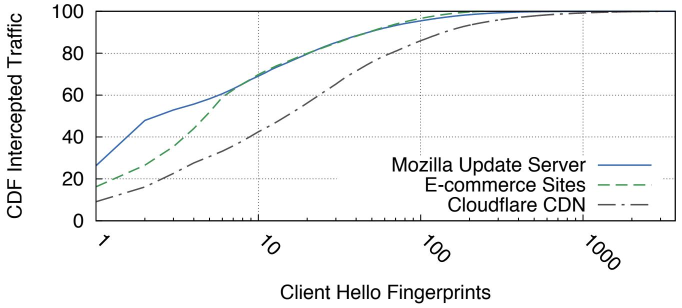


Fig. 5: **CDF of Interception Fingerprints**—We fingerprint non-ephemeral parameters in replacement Client Hello messages to track the products that perform interception. Ten fingerprints account for 69% of the intercepted e-commerce traffic, 69% of Firefox update traffic, and 42% of Cloudflare traffic.

	Fingerprint Description	% Total
E-commerce	*Unknown*	17.1%
	Avast Antivirus	10.8%
	Unknown	9.4%
	Blue Coat	9.1%
	Unknown	8.3%
Cloudflare	Avast Antivirus	9.1%
	AVG Antivirus	7.0%
	Unknown. Likely AV; mainly Windows 10/Chrome 47	6.5%
	Kaspersky Antivirus	5.0%
	BitDefender Antivirus	3.1%
Firefox	Bouncy Castle (Android 5)	26.3%
	Bouncy Castle (Android 4)	21.6%
	Unknown. Predominantly India	5.0%
	ESET Antivirus	2.8%
	Dr. Web Antivirus	2.6%

Fig. 6: **Top Interception Fingerprints**—We show the products responsible for the most interception at each vantage point.

the connection to the origin web server. We logged the raw TLS Client Hello messages and HTTP User-Agent for a random 0.5% sample of all TLS connections to Cloudflare's frontend between May 13–20, 2016. We measure interception by detecting mismatches between the HTTP User-Agent and the TLS handshake. Cloudflare provides a more representative sample of browsers than the Firefox update servers. However, one of Cloudflare's foremost goals is to prevent DDoS attacks and other abuse (e.g., scripted login attempts), so the data is messier than the other two datasets, and a portion of connections likely have falsified HTTP User-Agent headers.

V. Results

Our three vantage points provide varying perspectives on the total amount of interception: 4.0% of Firefox update connections, 6.2% of the e-commerce connections, and 10.9% of Cloudflare sessions in the United States (Figure 2). In all cases, this is more than an order of magnitude higher than previously estimated [27], [46].

Detection Method	Firefox	E-commerce	Cloudflare
Invalid Extensions	16.8%	85.6%	89.0%
Invalid Ciphers	98.1%	54.2%	68.7%
Invalid Version	–	2.0%	–
Invalid Curves	–	5.5%	9.4%
Invalid Extension Order	87.7%	33.9%	40.4%
Invalid Cipher Order	98.8%	21.2%	21.1%
Missing Required Ext.	97.9%	91.1%	50.9%
Injected HTTP Header	–	14.0%	–

Fig. 7: **Handshake Mismatches**—We break down the mismatches used to detect intercepted sessions. For more than 85% of intercepted connections, we detect an invalid handshake based on the use of unsupported extensions, ciphers, or curves. Some features were unavailable for Firefox and Cloudflare.

A. Firefox Update Server

HTTPS connections for 4.0% of Firefox clients were intercepted, which is the lowest rate among the three perspectives. Interception is likely less common for Firefox users because the browser ships with its own certificate store, whereas Internet Explorer, Chrome, and Safari use the host operating system's root store. Prior work [12] and our own testing (Figure 4) both find that some antivirus products (e.g., Avast) will intercept connections from these other browsers but neglect to proxy Firefox sessions. In corporate environments, administrators can separately install additional root authorities in Firefox [41], but the added step may dissuade organizations that proxy connections from deploying the browser.

Sources of Interception The two most common interception fingerprints belong to the default configurations of Bouncy Castle on Android 4.x and 5.x, and account for 47% of intercepted clients (Figure 5). These fingerprints were concentrated in large ASes belonging to mobile wireless providers, including Verizon Wireless, AT&T, T-Mobile, and NTT Docomo (a Japanese mobile carrier). As can be seen in Figure 9, 35% of all Sprint and 25.5% of all Verizon Firefox connections (including non-intercepted) matched one of the two fingerprints. It is possible to intercept TLS connections on Android using the VPN and/or WiFi permissions. However, given the default values, it is unclear exactly which Android application is responsible for the interception. Bouncy Castle on Android 5.x provides reasonable ciphers equivalent to a modern browser; on Android 4.x, Bouncy Castle advertises export-grade cipher suites, making it vulnerable to interception by an on-path attacker. The third most common fingerprint accounts for 5.3% of Firefox traffic. We were not able to identify the product associated with the fingerprint but note that nearly half of its traffic occurred in India and its diurnal and weekend patterns are consistent with home antivirus or malware.

Temporal Pattern The number of raw intercepted connections mirrors the diurnal pattern of all Firefox traffic. As can be seen in Figure 8, there are typically more connections on weekdays, and we observe the peak number of connections on weekday mornings. This intuitively aligns with the first computer access of the day triggering a connection to the Firefox update server. Oddly, though, the percentage of intercepted traffic is inversely proportional to total traffic, peaking near midnight and in the early morning. When we remove the two Android

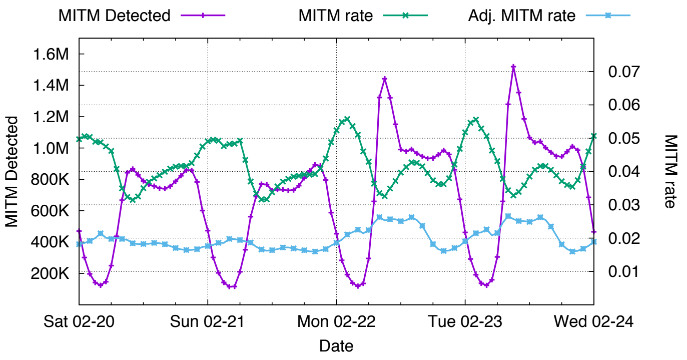


Fig. 8: **Temporal Variation in Firefox Interception**—We observe the highest amount of raw interception during times of peak traffic, but the highest interception rates during periods of low total traffic. This is likely because the two largest fingerprints are associated with mobile carriers and will update at night when desktop computers are powered off. Excluding these two fingerprints, interception remains relatively stable.

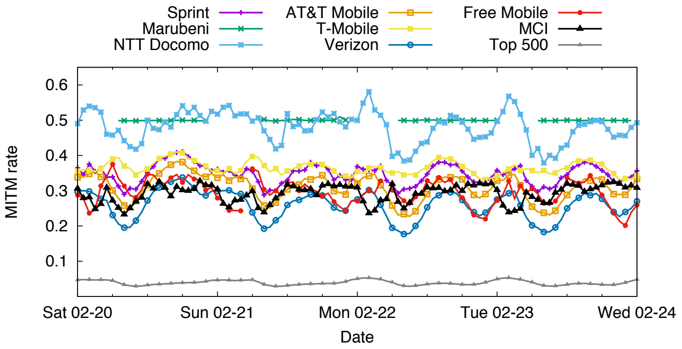


Fig. 9: **ASes with Highest Firefox Interception**—We find that 8 ASes have significantly higher interception rates within the top 500 ASes. All but one are mobile providers.

Bouncy Castle fingerprints, the total percentage of intercepted connections decreases by 47% and no longer peaks during these off hours. We suspect that the interception peaks are the result of mobile devices remaining on at night when other desktop computers are powered off. Weekend interception rates level out around 2% and increase during the weekdays, suggesting the presence of corporate TLS proxies.

Geographic Disparities Interception is more prevalent in several countries (Figure 10). For example, 15% of TLS connections from Guatemala were intercepted, a rate 3–4 times higher than the global average. This is primarily due to COMCEL, a mobile provider responsible for 34.6% of all Guatemalan update server traffic, having a 32.9% interception rate. Greenland has the second highest interception rate (9.9%), which is caused by a single AS: TELE Greenland. Nearly half of the interception is performed by a Fortigate middlebox. The third most commonly intercepted country, South Korea, is one of the most highly mobile-connected countries in the world [55]. In general, large ASes with above average interception belong to mobile providers and fluctuate between 20% to 55% interception depending on the time of day, as seen in Figure 9. The single exception is Marubeni OKI Network Solutions, which maintains a consistent interception rate around 50% that begins in the morning and ends near midnight everyday. It is unclear what behavior results in this temporal pattern.

Country	MITM %	Country	MITM %
Guatemala	15.0%	Kiribati	8.2%
Greenland	9.9%	Iran	8.1%
South Korea	8.8%	Tanzania	7.3%
Kuwait	8.5%	Bahrain	7.3%
Qatar	8.4%	Afghanistan	6.7%

Fig. 10: **Countries with Highest Firefox Interception**—We show the ten countries with the highest interception rates when connecting to the Mozilla update server. Countries with above average interception rates generally have a large amount of traffic intercepted by a single, dominant mobile provider.

B. *Popular E-commerce Sites*

The e-commerce dataset is composed of visits to set of popular e-commerce websites and is not limited to a specific browser version. To account for this, we parsed the HTTP User-Agent header and identified mismatches between the announced browser and TLS handshake. We observed 257K unique User-Agent headers, and successfully parsed the header for 99.5% of connections. The browsers we fingerprinted account for 96.1% of connections; 2.5% belong to browsers we did not fingerprint and 1.4% are from old browser versions.

We find that 6.8% of connections were intercepted and another 0.9% were likely intercepted, but cannot be definitively classified. For the connections where we could detect a specific fingerprint, 58% belong to antivirus software and 35% to corporate proxies. Only 1.0% of intercepted traffic is attributed to malware (e.g., SuperFish), and the remaining 6% belong to miscellaneous categories. The three most prevalent known fingerprints belong to Avast Antivirus, Blue Coat, and AVG Antivirus, which account for 10.8%, 9.1%, and 7.6% of intercepted connections, respectively.

The e-commerce dataset also includes HTTP headers, which allow us to identify a subset of connections that were intercepted by network middleboxes, but do not match any our existing fingerprints. We find proxy-related headers in 14.0% of invalid handshakes, most prominently `X-BlueCoat-Via`, `Via`, `X-Forwarded-For`, and `Client-IP`. We additionally use these headers to detect interception. We detected 96.1% of interception based on a version mismatch or the presence of invalid extensions or ciphers. Another 2.2% of intercepted connections lacked expected extensions, 0.7% used invalid cipher or extension ordering, and 1.6% contained proxy-related HTTP headers (Figure 7).

Chrome accounts for 40.3% of TLS traffic, of which 8.6% was intercepted, the highest interception rate of any browser. On the other extreme, only 0.9% of Mobile Safari connections were intercepted. Interception is far more prominent on Windows, where we see 8.3%–9.6% interception compared to 2.1% on Mac OS. This is likely because most corporate users use Windows in the workplace, and many antivirus products that perform interception are Windows-based. We summarize these results in Figure 13.

Falsified User-Agents It is possible that some connections in the e-commerce dataset have falsified User-Agents headers, which would artificially inflate the interception rate. We intuitively expect that handshakes belonging to interception products will have a larger number of associated User-Agents than a custom crawler with a falsified User-Agent. All but a handful of the TLS products we investigated have at least 50 associated User-Agents, and at the extreme, Avast Antivirus and Blue Coat's corporate proxy had 1.8K and 5.2K associated User-Agents, respectively. When we excluded interception fingerprints associated with less than 50 User-Agents, the global interception rate dropped from 6.8% to 6.2%. Given this modest decrease, we suspect that the mismatches we detect are due to interception instead of spoofed User-Agents. However, we take a conservative route and restrict our analysis to the 6.2%.

During our analysis, we noted two irregularities. First, nearly 3 million connections (approximately 75% of which had a Blue Coat header) used the generic User-Agent string "`Mozilla/4.0 (compatible;)`"). Cisco has documented that Blue Coat devices will frequently mask browser User-Agents with this generic agent string [31]. Despite knowing Blue Coat devices intercept these TLS connections, we take the most conservative approach and exclude these connections from our analysis because we do not know what percentage of devices behind the proxy would have been identifiable. However, we note that if we assume the same proportion of identifiable browsers as the general population (95.6%), Blue Coat would be the second largest fingerprint and the total percentage of connections intercepted would rise from 6.2% to 7.0%. Second, we find that over 90% of Internet Explorer connections on Windows XP appear intercepted because they include modern ciphers and extensions that were not previously documented on XP, nor that we could reproduce. We exclude these connections.

C. Cloudflare

The Cloudflare network provides perhaps the most representative view of global HTTPS traffic, but also the messiest. We initially observe a 31% interception rate, which is 3–7 times higher than the other vantage points. This elevated interception rate is likely due to abuse (e.g., login attempts or content scraping) and falsified User-Agent headers — some of the very types of requests that Cloudflare protects against. The Firefox server relied on only Firefox browsers accessing an obscure update server and the e-commerce websites required JavaScript execution in order to record a TLS connection. In contrast, the Cloudflare data reflects all TLS connections across a broad range of websites, so even simple command line utilities such as *wget* and *curl* can appear in the Cloudflare dataset with falsified User-Agent headers.

We take several steps to account for this abuse. First, we removed fingerprints associated with fewer than 50 unique User-Agent headers. Next, we limited our analysis to the 50 largest ASes that are not cloud or hosting providers. Unfortunately, even after this filter, we still observe an artificially high interception rate ranging from 11% in the Americas to 42% in Asia. We find that while large ASes in the U.S. have clear purposes, the majority of networks in Europe and Asia do not. In Asia, numbers varied widely and most ASes had little description. In Europe, ASes would frequently span multiple countries and contain requests that appeared to from both home users and hosting providers. We limit our analysis to the ASes from the top 50 that were located in the United States and primarily serve end users. While this reduces the scope of the dataset, there are lower interception rates in the U.S. compared to any other region, providing a conservative lower bound.

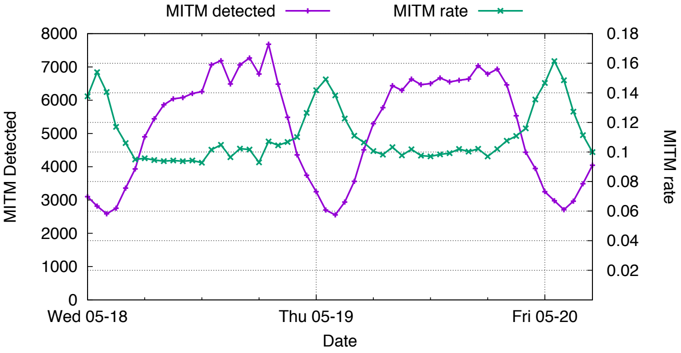


Fig. 11: **Temporal Variation in Cloudflare Interception** — We observe the highest amount of raw interception during times of peak traffic, but the highest interception rates during periods of low total traffic. This is likely due to higher percentages of automated bot traffic where the User-Agent header is spoofed.

Network Type	No Interception	Likely	Confirmed
Residential/Business	86.0%	0.4%	13.6%
Cell Provider	94.1%	0.1%	5.8%

Fig. 12: **U.S. Network Breakdown** — We show the Cloudflare interception rates for types of U.S. networks.

In the U.S., we observe a 10.9% interception rate, with a stark contrast between mobile ASes (5.2–6.5%) and residential/enterprise ASes 10.3–16.9%), per Figure 12. Four of the top five handshake fingerprints belong to antivirus providers: Avast, AVG, Kaspersky, and BitDefender, which are also prominent on the e-commerce sites. The remaining unidentified fingerprint primarily occurs for Chrome 47 on Windows 10. Despite the alignment with a specific browser version and OS — which might indicate an incorrect heuristic — we confirm that this handshake cannot be produced by Chrome and advertises 80 cipher suites including IDEA/CAMELLIA, diverging significantly from the Chrome family. The fingerprint also occurs consistently across non-mobile ASes and peaks usage during evening hours, suggesting malware or antivirus software. These five fingerprints account for 31% of intercepted traffic (Table 5).

Similar to Firefox updates, the total amount of HTTPS interception correlates with total HTTPS traffic, peaking in the middle of the day and declining during evening hours, but with the highest interception rates at night (Figure 11). This might be due to mobile traffic as we saw for Firefox but could also indicate the presence of bot traffic.

D. Results Summary and Validation

We can partially validate our methodology by checking whether we failed to detect any connections that included proxy-related HTTP headers as intercepted. We find that 1.6% of the e-commerce connections included proxy headers, but

E-commerce Sites				Cloudflare			
Browser	All Traffic	Intercepted	Of Intercepted	Browser	All Traffic	Intercepted	Of Intercepted
Chrome	40.3%	8.6%	56.2%	Chrome	36.2%	14.7%	48.8%
Explorer	16.8%	7.4%	19.6%	Mobile Safari	17.5%	1.9%	3.3%
Firefox	13.5%	8.4%	18.2%	Explorer	14.9%	15.6%	21.2%
Safari	10.2%	2.1%	3.4%	Safari	8.9%	6.5%	5.3%
Chromium	7.6%	0.1%	0.1%	Firefox	8.5%	18.2%	14.2%
Mobile Safari	7.6%	0.9%	1.1%	Mobile Chrome	8.4%	4.7%	3.6%
Other	4.0%	4.0%	2.4%	Other	5.6%	7.0%	3.6%
OS	All Traffic	Intercepted	Of Intercepted	OS	All Traffic	Intercepted	Of Intercepted
Windows 7	23.3%	9.6%	56.6%	Windows 7	23.9%	13.4%	29.2%
Windows 10	22.5%	9.3%	14.3%	Windows 10	22.9%	13.1%	27.4%
iOS	17.3%	0.1%	1.1%	iOS	17.5%	2.0%	3.2%
Mac OS	15.8%	2.1%	6.5%	Mac OS	16.0%	6.6%	9.6%
Android	9.4%	1.0%	0.5%	Android	9.5%	4.8%	4.2%
Windows 8.1	6.9%	8.3%	15.8%	Windows 8.1	4.9%	24.4%	11.0%
Other	4.8%	21.4%	15.2%	Other	5.3%	31.7%	15.4%

Fig. 13: **OS and Browser Breakdown**—We show the breakdown of all traffic, the amount of traffic intercepted, and percentage of all interception that each browser and operating system accounts for across both the e-commerce and Cloudflare vantage points.

did not have evidence of interception in their TLS handshakes. This suggests that the methodology catches the vast majority of interception, but it does miss some edge cases.

To verify that our heuristics aren't incorrectly classifying valid handshakes, we investigated why our heuristics marked connections as intercepted. We detected more than 85% of intercepted connections based on the presence of known unsupported ciphers or extensions, rather than a missing extension or invalid ordering. More than 98% of intercepted connections in the Firefox dataset were found based on the inclusion of ciphers that have never been implemented in NSS, and 82% of all intercepted connections indicated support for the heartbeat extension—an immediate giveaway given that no browsers support the extension. This suggests that our heuristics are finding handshakes produced by other libraries rather than misclassifying browser edge cases.

However, while the methodology appears sound, the three perspectives we studied provide differing numbers on the total amount of interception. All three perspectives find more than an order of magnitude more interception than previously estimated, and we estimate that 5–10% of connections are intercepted. However, we offer a word of caution on the exact numbers, particularly for the Cloudflare dataset, where abuse may inflate the interception rate we observe.

VI. Impact on Security

In this section, we investigate the security impact of HTTPS interception. First, we introduce a grading scale for quantifying TLS client security. Then, we investigate common interception products, evaluating the security of their TLS implementations. Based on these ratings and the features advertised in Client Hello messages, we quantify the change in security for intercepted connections.

A. Client Security Grading Scale

There does not exist a standardized rubric for rating TLS client security. We define and use the following scale to consistently rate browsers, interception products, and the connections we observe in the wild:

A: Optimal. The TLS connection is equivalent to a modern web browser in terms of both security and performance. When grading cipher suites, we specifically use Chrome's definition of "secure TLS" [11].

B: Suboptimal. The connection uses non-ideal settings (e.g., non-PFS ciphers), but is not vulnerable to known attacks.

C: Known Attack. The connection is vulnerable to known TLS attacks (e.g., BEAST, FREAK, and Logjam), accepts 768-bit Diffie-Hellman parameters, or advertises RC4 support.

F: Severely Broken. The connection is severely broken such that an active man-in-the-middle attacker could intercept and decrypt the session. For example, the product does not validate certificates, or offers known-broken cipher suites (e.g., DES).

Our grading scale focuses on the security of the TLS handshake and does not account for the additional HTTPS validation checks present in many browsers, such as HSTS, HPKP, OneCRL/CRLSets, certificate transparency validation, and OCSP must-staple. None of the products we tested supported these features. Therefore, products that receive an A grade for their TLS security likely still reduce overall security when compared to recent versions of Chrome or Firefox.

B. Testing Security Products

To measure the security of the connections initiated by interception products, we installed the trial versions of the corporate proxies, popular client security software, and malware listed in Section III.[6] We then ran the latest version of Chrome, Internet Explorer, Firefox, and Safari through each product, visiting a website that executed the following tests:

[6] Product demos could have different security profiles than their production counterparts. However, during our disclosure process, none of the manufacturers we contacted indicated this.

1) **TLS Version.** We check the highest version of TLS that the product supports. We grade any client that supports at best TLS 1.1 as B, SSLv3 as C, and SSLv2 as F.
2) **Cipher Suites.** We investigate the cipher suites present in the Client Hello. We rate any product that does not support Chrome's Strong TLS ciphers [11] as B, handshakes that offer RC4 as C, and any product that advertises broken ciphers (e.g., export-grade or DES) as F.
3) **Certificate Validation.** We present a series of untrusted certificates, including expired, self-signed, and invalidly-signed certificates. We further test with certificates signed by CAs with a publicly-known private key (i.e., Superfish [58], eDell, and Dell provider roots [13]). We rate any product that accepts one of these certificates as F.
4) **Known TLS Attacks.** We check whether clients are vulnerable to the BEAST, FREAK, Heartbleed, and Logjam attacks, or accept weak Diffie-Hellman keys. We rate any vulnerable client as C.[7]

Corporate Middleboxes The default configurations for eleven of the twelve middleboxes we tested weaken connection security, and five of the twelve products introduce severe vulnerabilities that would enable future interception by a man-in-the-middle attacker. Ten support RC4-based ciphers, two advertise export-grade ciphers, and three have broken certificate validation (Figure 3). We note that the installation process for many of these proxies is convoluted, crash-prone, and at times, non-deterministic. Configuration is equally confusing, oftentimes with little to no documentation. For example, Cisco devices allow administrators to customize permitted cipher suites, but do not provide a list of ciphers to choose from. Instead, the device provides an undocumented text box that appears to accept OpenSSL cipher rules (e.g., `ALL:!ADH:@STRENGTH`). We suspect that this poor usability contributes to the abysmal configurations we see in the wild.

We were not able to acquire a demo or trial of the ZScaler Proxy, a prominently advertised cloud-based middlebox. Instead, we investigated the traffic that originated from one of the seven ZScaler ASes in the Cloudflare dataset.[8] We found one predominant handshake fingerprint that accounts for more than four times more traffic than the second most popular, and does not match any popular browsers indicated in the associated User-Agent strings. We would have rated this handshake at best B due to the lack of any perfect-forward-secret ciphers. However, we exclude it from any other analysis, because we were not able to check for further vulnerabilities.

Client-side Security Software In line with de Carné de Carnavalet and Mannan [12], we find that nearly all of the client-side security products we tested reduce connection security and ten introduce severe vulnerabilities (Figure 4). We note that these security grades are a lower bound that assume TLS stacks have no additional vulnerabilities. However, in practice, researchers discover bugs in antivirus software regularly. For Avast alone, ten vulnerabilities have been publicly disclosed within the last eight months, one of which allowed remote code execution through carefully crafted certificates [48].

Malware and Unwanted Software Researchers have previously found that Komodia does not validate certificates [58] and we find that the NetFilter SDK similarly does not properly validate certificate chains. We grade both as F: severely broken.

C. Impact on TLS Traffic

While many security products have insecure defaults, intercepted connections could have a different security profile. Security might be improved if administrators configure middleboxes to perform responsible handshakes, or, even with their poor security, proxies might protect further out-of-date clients. We investigated the security of intercepted handshakes based on the parameters advertised in the handshake (e.g., TLS version and cipher suites), and in the cases where we can identify the interception product, its security rating. To determine the change in security rather than just the security of the new connection, we calculated the security of the browser version specified in the HTTP User-Agent and compare that to the security of the handshake we observe.

Similar to how each of our three networks has a different interception rate, each vantage point presents a different security impact. For Firefox, 65% of intercepted connections have reduced security and an astounding 37% have negligent security vulnerabilities. 27% of the e-commerce and 45% of the Cloudflare connections have reduced security, and 18% and 16% are vulnerable to interception, respectively.

Interception products increased the security for 4% of the e-commerce and 14% of the Cloudflare connections. The discrepancy in increased security is largely due to temporal differences in data collection and the deprecation of RC4 cipher suites during this period. When the e-commerce sites collected data in August 2015, browsers considered RC4 to be safe. However, between August 2015 and April 2016, standards bodies began advising against RC4 [50] and both Chrome and Firebox deprecated the cipher [59]. When grading Cloudflare connections in May 2016, we labeled connections that advertised RC4 as C. This results in connections from older versions of Internet Explorer and Safari being marked insecure, and proxies improving the security for an increased number of connections.

Corporate Middleboxes During our earlier analysis of corporate proxies, we found that many network middleboxes inject HTTP headers, such as `X-Forwarded-For` and `Via`, to assist managing simultaneous proxied connections. We analyzed the connections in the e-commerce dataset with proxy-related headers to better understand the security of corporate middleboxes compared to client-side software. We find that connection security is significantly worse for middleboxes than the general case. As can be seen in Figures 14 and 15, security is degraded for 62.3% of connections that traverse a middlebox and an astounding 58.1% of connections have severe vulnerabilities.

We note a similar phenomenon in the Firefox data where we manually investigated the top 25 ASes with more than 100K connections, the highest interception rates, and a single predominant interception fingerprint. We primarily find financial firms, government agencies, and educational institutions. With the exception of one bank, 24 of the top 25 ASes have worsened security due to interception. For 12 of the 25 ASes,

[7] We tested these products in January–March, 2016, which was approximately eight months after the Logjam disclosure and eleven months after FREAK.

[8] We investigated ASes 62907, 55242, 53813, 53444, 40384, 32921, and 22616.

Network	Increased Security	Decreased Security	Severely Broken
E-commerce (All Traffic)	4.1%	26.5%	17.7%
E-commerce (Middleboxes)	0.9%	62.3%	58.1%
Cloudflare	14.0%	45.3%	16.0%
Firefox Updates	0.0%	65.7%	36.8%

Fig. 14: **Impact of Interception**— We summarize the security impact of HTTPS interception, comparing client–proxy connection security with proxy–server connection security.

Dataset	Original Security	New Security			
		→A	→B	→C	→F
Firefox	A→	34.3%	16.8%	12.2%	36.8%
E-commerce Sites: All Traffic	A→	57.1%	2.9%	5.6%	8.1%
	B→	2.7%	10.2%	1.2%	8.3%
	C→	0.6%	0.4%	1.0%	0.3%
	F→	0.0%	0.2%	0.1%	1.0%
E-commerce Sites: Middleboxes	A→	13.5%	3.0%	0.8%	18.0%
	B→	0.7%	23.3%	0.6%	37.8%
	C→	0.1%	0.1%	0.0%	2.2%
	F→	0.0%	0.0%	0.0%	0.0%
Cloudflare	A→	17.3%	1.1%	29.7%	10.0%
	B→	0.0%	0.0%	0.0%	0.0%
	C→	9.4%	3.3%	22.0%	4.5%
	F→	0.8%	0.1%	0.4%	1.5%

Fig. 15: **Change in Security**— We calculate the change in connection security based on the parameters advertised in the Client Hello message and the security of the browser in the HTTP User-Agent header.

the predominant TLS handshake includes export-grade cipher suites, making them vulnerable to future interception by an active man-in-the-middle attacker.

VII. Discussion

While the security community has long known that security products intercept TLS connections, we have largely ignored the issue. We find that interception is occurring more pervasively than previously estimated and in many cases, introduces significant vulnerabilities. In this section, we discuss the implications of our measurements and make recommendations for both vendors and the security community.

We need community consensus. There is little consensus within the security community on whether HTTPS interception is acceptable. On the one hand, Chrome and Firefox have provided tacit approval by allowing locally installed roots to bypass key pinning restrictions [34]. However, at the same time, discussions over protocol features that facilitate safer interception have been met with great hostility within standards groups [35], [37]. These communities need to reach consensus on whether interception is appropriate in order to develop sustainable, long-term solutions.

We should reconsider where validation occurs. Many HTTPS security features expect connections to be end-to-end by mixing the HTTP and TLS layers, and by implementing HTTPS features in browser code rather than in TLS libraries. For example, to overcome weaknesses in existing revocation protocols, Firefox ships with OneCRL [43] and Chrome, CRLSets [24]. Both of these solutions increase browser security in the typical end-to-end case. However, these solutions provide no protection in the presence of a TLS proxy and because the solution is not part of the TLS protocol itself, TLS libraries do not implement these safe revocation checks. In a second example, HPKP directives are passed over HTTP rather than during the TLS handshake. Browsers cannot perform HPKP validation for proxied connections because they do not have access to the destination certificate and proxies do not perform this validation in practice.

While it is possible for proxies to perform this additional verification, they are not doing so, and in many cases vendors are struggling to correctly deploy existing TLS libraries, let alone implement additional security features. Given this evidence, our community needs to decide what roles should be carried out by the browser versus TLS implementation. If we expect browsers to perform this additional verification, proxies need a mechanism to pass connection details (i.e., server certificate and cryptographic parameters) to the browser. If we expect proxies to perform this validation, we need to standardize these validation steps in TLS and implement them in popular libraries. Unfortunately the current situation, in which we ignore proxy behavior, results in the worst case scenario where neither party is performing strict validation.

Cryptographic libraries need secure defaults. Several proxies deployed TLS libraries with minimal customization. Unfortunately the default settings for these libraries were vulnerable rendering the middlebox vulnerable. Client libraries and web servers need to prioritize making their products safe by default. We applaud OpenSSL's recent decision to remove known-broken cipher suites [2]. However, this change should have occurred more than a decade earlier and libraries continue to accept other weak options. Our community should continue restricting default options to known safe configurations.

Antivirus vendors should reconsider intercepting HTTPS. Antivirus software operates locally and already has access to the local filesystem, browser memory, and any content loaded over HTTPS. Given their history of both TLS misconfigurations [12] and RCE vulnerabilities [48], we strongly encourage antivirus providers to reconsider whether intercepting HTTPS is responsible.

Security companies are acting negligently. Many of the vulnerabilities we find in antivirus products and corporate middleboxes— such as failing to validate certificates and advertising broken ciphers— are negligent and another data point in a worrying trend of security products worsening security rather than improving it [12], [17]. We hope that by disclosing vulnerabilities in existing products, we can encourage manufacturers to patch problems. We urge companies to prioritize the security of their TLS implementations and to consider the pace at which the HTTPS ecosystem evolves and whether they can keep up with the necessary updates.

Do not rely on client configuration. Because cryptographic parameters must be supported by both the client and server, the security community has largely ignored HTTPS servers' lenient cipher support with the implied understanding that browser vendors will only advertise secure parameters. In 2015, Durumeric et al. found that nearly 37% of browser-trusted HTTPS servers on the IPv4 address space supported RSA export ciphers [16] despite their known weaknesses and discontinued use. It was only after the discovery of the FREAK attack—a bug in OpenSSL that allowed an active attacker to downgrade connections to export-grade cryptography—that operators began to actively disable export cipher suites.

While modern browsers are not vulnerable to active downgrade attacks, nearly two thirds of connections that traverse a network middlebox advertise export ciphers and nearly 3% of *all* HTTPS connections to the e-commerce sites included at least one export-grade cipher suite. While these products would optimally not be vulnerable, their risk can be reduced by encouraging websites to disable weak ciphers. Similarly, some interception products support secure ciphers, but do not order them correctly. In this situation, servers that explicitly choose strong ciphers will negotiate a more secure connection than those that honor client preference. We need to practice defense-in-depth and encourage both clients and servers to select secure parameters instead of relying on one side to always act sanely.

Administrators need to test middleboxes. Many of the products we tested support more secure connections with additional configuration. Unfortunately, this does not appear to be happening in practice. While manufacturers clearly need to improve their default settings, we also encourage the administrators who are deploying proxies to test their configurations using sites such as Qualys SSL Lab's client test, https://howsmyssl.com, and https://badssl.com. To incentivize better software, servers could consider checking client configuration and rejecting insecure clients.

VIII. Related Work

There have been several recent studies on fingerprinting TLS connections, HTTPS interception, and safe interception protocols:

Fingerprinting TLS Handshakes While there has been little rigorous study on TLS handshake fingerprinting, several groups have previously suggested the idea. Ristić first described the approach in 2009 [52]. Later, in 2012, Majkowski implemented SSL fingerprinting in p0f [36]. In 2015, Husá et al. used client fingerprinting to broadly describe the types of HTTPS traffic on their institutional network [28], and in 2016, Brotherston showed how desktop applications could be identified by their Client Hello messages [9]. Concurrent to our work, Cisco showed that malware uses different TLS parameters than browsers [3]. To the best of our knowledge, no groups have used the methodology to measure HTTPS interception. We also note that we take a slightly different approach than described in these prior works in which we look for inconsistencies between the HTTP and TLS layers rather than trying to passively identify handshakes seen on the wire.

Measuring HTTPS Interception Several groups have measured the prevalence of TLS interception by deploying Flash-based measurements on popular websites or by purchasing ads. In early 2014, Huang et al. analyzed 3M SSL connections to Facebook and found that 0.2% of connections were intercepted by a variety of antivirus software, corporate content filters, and malware [27]. A year later in 2015, O'Neill et al. deployed a Google AdWords campaign in which they observed 15.2M connections and found 0.4% of TLS connections are intercepted—nearly double the number found by Facebook [46]. We find approximately an order of magnitude more interception at each of our three vantage points.

It is not immediately evident why our numbers differ from these previous studies. It is possible that some of our numbers—particularly for Cloudflare—are overestimates due to abuse. However, we note that the number of intercepted connections found in these prior studies is less than the number of connections with Blue Coat HTTP headers alone and less than half of the number of connections that include proxy-related HTTP headers. There may be bias introduced by using Flash, which browsers are in the process of deprecating and is not available on many mobile devices. Facebook may be blocked on the types of corporate networks that have deployed TLS inspection, and some malware may block connections to ad providers in order to inject their own ads.

The two methodologies have other differences. Studying the certificates presented to clients provides a limited view on the security impacts of MITM because the client is not able to observe the security properties of the outgoing connection to the destination web server. However, because clients can access the certificate presented by the proxy, there is more data on the product that intercepted the connection.

Client Security Several studies have investigated the security of specific TLS clients. Georgiev et al. analyzed the security of non-browser software that validates SSL certificates by installing TLS clients and validating their behavior [21]. They identify numerous vulnerabilities in TLS clients, including Amazon's EC2 client, Amazon and PayPal's SDKs, and a number of other e-commerce products. They conclude that these vulnerabilities are the result of poorly designed cryptographic libraries.

de Carné de Carnavalet and Mannan demonstrated that many popular Windows antivirus applications perform TLS traffic inspection that weakens connection security [12]. We extend this work by investigating additional products, showing that Mac connections have worse security than their Windows-based counterparts, and fingerprinting their connections. There have been repeated discussions about the theoretical danger of network middleboxes that intercept TLS connections. Dormann details these potential risks and lists devices that perform interception [15], but to the best of our knowledge, no one has systematically characterized these devices. We build on this investigation, testing devices for vulnerabilities and fingerprinting their behavior.

Alternatives to Interception Researchers have proposed both alternatives to TLS interception and TLS extensions to help middleboxes safely intercept connections. In the simplest case, the HTTP 2.0 Explicit Trusted Proxy RFC [26] requires middleboxes to explicitly notify the client of interception.

The proposed TLS Proxy Server Extension [38] extends the idea, requiring the proxy to indicate the interception, but to additionally relay proxy–server session information back to the client, such that the client can validate the server's identify and perform additional validation.

Naylor et al. introduced multi-context TLS (mcTLS) [44], an extended version of TLS that requires endpoints to explicitly specify permitted middleboxes in order to securely authenticate each hop and cryptographically control *exactly* what data middleboxes can access. Sherry et al. eschewed any plaintext data access for middleboxes and developed BlindBox, a cryptographic protocol that supports both intrusion and data exfiltration detection through searchable encryption [54]. Lan et al. extended BlindBox with Embark, which further optimized keyword and prefix matching searchable encryption schemes [33].

IX. Responsible Disclosure

In the course of analyzing corporate middleboxes and client-side security software, we uncovered a range of TLS implementation errors, many of which allow connections to be intercepted by a man-in-the-middle attacker. We disclosed these weaknesses to manufacturers in August 2016. Several manufacturers indicated that they deployed updates that protect against attacks after our testing but before disclosure. Others indicated plans to deprecate RC4 and move to modern cipher suites. In many cases, we received no response and in other cases, we were unable to convince manufacturers that TLS vulnerabilities such as Logjam required patching. One company would not accept our vulnerability report without a product serial number, and several indicated that secure product configuration was a customer responsibility and that they would not be updating their default configuration. Figures 3 and 4 show scores prior to disclosure.

Several manufacturers asked that we note patched versions of their products. In May 2016, A10 introduced a configuration wizard that recommends a more sane set of cipher suites. Avast removed broken ciphers from their Mac product line in July, 2016. Fortinet patched the Logjam vulnerability in version 5.4.1, which was released in September, 2016. Qustodio patched in October, 2016. Untangle indicated that issues had already been addressed at the time of disclosure.

X. Conclusion

In this paper, we conducted the first comprehensive study on the security impact of HTTPS interception in the wild. We characterized the TLS handshakes produced by modern browsers, common security products, and malware, finding that products advertise varied TLS parameters. Building on this observation, we constructed a set of heuristics that allow web servers to detect HTTPS interception and identify popular interception products. We deployed these heuristics on three diverse networks: (1) Mozilla Firefox update servers, (2) a set of popular e-commerce sites, and (3) the Cloudflare content distribution network. In each case, we find more than an order of magnitude more interception than previously estimated, ranging from 4–11%. As a class, interception products drastically reduce connection security. Most concerningly, 62% of traffic that traverses a network middlebox has reduced security and 58% of middlebox connections have severe vulnerabilities. We investigated popular antivirus and corporate proxies, finding that nearly all reduce connection security and that many introduce vulnerabilities (e.g., fail to validate certificates). While the security community has long known that security products intercept connections, we have largely ignored the issue, believing that only a small fraction of connections are affected. However, we find that interception has become startlingly widespread and with worrying consequences. We hope that by bringing these issues to light, we can encourage manufacturers to improve their security profiles and prompt the security community to discuss alternatives to HTTPS interception.

Acknowledgments

The authors thank Peter Bowen, Xavier de Carné de Carnavalet, Patrick Donahue, Alessandro Ghedini, Mohammad Mannan, Paul Pearce, Michal Purzynski, Ivan Ristić, Narseo Vallina Rodriguez, and Ryan Sleevi. This work was supported in part by the National Science Foundation under awards CNS-1505790, CNS-1518741, CNS-1345254, CNS-1409505, CNS-1518888, CNS-1530915, CNS-1237265, and CNS-1348077, by the Department of Energy under award DE-OE0000780, by the NSF Graduate Research Fellowship Program under grant DGE-1256260, by the Post-9/11 GI Bill, by a Google Ph.D. Fellowship, and by an Alfred P. Sloan Foundation Research Fellowship.

References

[1] NetFilter SDK. http://netfiltersdk.com/.

[2] OpenSSL changelog. https://www.openssl.org/news/changelog.html.

[3] B. Anderson, S. Paul, and D. McGrew. Deciphering malware's use of TLS (without decryption). *arXiv preprint arXiv:1607.01639*, 2016.

[4] R. Arora and N. Aggarwal. Browserstack. https://browserstack.com.

[5] R. Barnes. Deprecating non-secure HTTP. Mozilla Security Blog. https://blog.mozilla.org/security/2015/04/30/deprecating-non-secure-http/.

[6] R. Barnes. Man-in-the-middle interfering with increased security. Mozilla Security Blog. https://blog.mozilla.org/security/2016/01/06/man-in-the-middle-interfering-with-increased-security/.

[7] H. Böck. Software Privdog worse than Superfish. https://blog.hboeck.de/archives/865-Software-Privdog-worse-than-Superfish.html.

[8] H. Böck. Superfishy. https://github.com/hannob/superfishy.

[9] L. Brotherston. Stealthier attacks and smarter defending with TLS fingerprinting. *DerbyCon 2015*.

[10] E. Bursztein. Speeding up and strengthening HTTPS connections for Chrome on Android. https://security.googleblog.com/2014/04/speeding-up-and-strengthening-https.html.

[11] Chromium. IsSecureTLSCipherSuite function. https://chromium.googlesource.com/chromium/src/net/+/master/ssl/ssl_cipher_suite_names.cc#373.

[12] X. de Carné de Carnavalet and M. Mannan. Killed by proxy: Analyzing client-end TLS interception software. In *Network and Distributed System Security Symposium*, 2016.

[13] Dell. Information on the eDellRoot and DSDTestProvider certificates. http://www.dell.com/support/article/us/en/19/SLN300321.

[14] T. Dierks and E. Rescorla. Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246.

[15] W. Dormann. The risks of SSL inspection. https://insights.sei.cmu.edu/cert/2015/03/the-risks-of-ssl-inspection.html.

[16] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. Tracking the FREAK attack. https://freakattack.com/.

[17] Z. Durumeric, D. Adrian, A. Mirian, J. Kasten, E. Bursztein, N. Lidzborski, K. Thomas, V. Eranti, M. Bailey, and J. A. Halderman. Neither snow nor rain nor MITM...: An empirical analysis of email delivery security. In *ACM Internet Measurement Conference*, 2015.

[18] P. Eckersley. How unique is your web browser? In *Symposium on Privacy Enhancing Technologies*, 2010.

[19] C. Evans, C. Palmer, and R. Sleevi. Public key pinning extension for HTTP. RFC 7469.

[20] S. Farrell and H. Tschofenig. Pervasive monitoring is an attack. RFC-7258, 2014.

[21] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: validating SSL certificates in non-browser software. In *ACM Conference on Computer and Communications Security*, 2012.

[22] Google. Certificate transparency in Chrome. https://www.certificate-transparency.org/certificate-transparency-in-chrome.

[23] Google. HTTPS as a ranking signal. https://webmasters.googleblog.com/2014/08/https-as-ranking-signal.html.

[24] Google Chromium. CRLSets. https://dev.chromium.org/Home/chromium-security/crlsets.

[25] J. Graham-Cumming. The two reasons to be an engineer at Cloudflare. https://blog.cloudflare.com/the-two-reasons-to-work-for-cloudflare/.

[26] H. W. Group. Explicit trusted proxy in HTTP/2.0, 2014. https://tools.ietf.org/html/draft-loreto-httpbis-trusted-proxy20-01.

[27] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson. Analyzing forged SSL certificates in the wild. In *IEEE Symposium on Security and Privacy 2014*.

[28] M. Husák, M. Cermák, T. Jirsík, and P. Celeda. Network-based HTTPS client identification using SSL/TLS fingerprinting, 2015.

[29] IANA. Transport layer security (TLS) extensions. http://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml.

[30] IANA. Transport layer security (TLS) parameters. http://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml.

[31] A. Kirk. Web proxies, user-agent strings, and malware detection. http://blog.talosintel.com/2012/11/web-proxies-user-agent-strings-and.html.

[32] Komodia. Redirector. http://www.komodia.com/products/komodia-redirector.

[33] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu. Embark: securely outsourcing middleboxes to the cloud. In *USENIX Symposium on Networked Systems Design and Implementation*, 2016.

[34] A. Langley. Public key pinning. https://www.imperialviolet.org/2011/05/04/pinning.html.

[35] P. Lepeska. Comments on explicit/trusted proxy. https://www.ietf.org/mail-archive/web/httpbisa/current/msg13124.html.

[36] M. Majkowski. SSL fingerprinting for p0f. https://idea.popcount.org/2012-06-17-ssl-fingerprinting-for-p0f/.

[37] D. McGrew. Comments on explicit/trusted proxy. https://www.ietf.org/mail-archive/web/tls/current/msg07815.html.

[38] D. McGrew, D. Wing, Y. Nir, and P. Gladstone. TLS proxy server extension, 2012. https://tools.ietf.org/html/draft-mcgrew-tls-proxy-server-01.

[39] Microsoft. Platform status. https://developer.microsoft.com/en-us/microsoft-edge/platform/status/publickeypinningextensionforhttp.

[40] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham. Fingerprinting information in JavaScript implementations. *Web 2.0 Security & Privacy*, 2011.

[41] Mozilla. Installing certificates into Firefox. https://wiki.mozilla.org/CA:AddRootToFirefox.

[42] Mozilla. Network security services (NSS). https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS.

[43] Mozilla. Revoking intermediate certificates: Introducing OneCRL. https://blog.mozilla.org/security/2015/03/03/revoking-intermediate-certificates-introducing-onecrl/.

[44] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste. Multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS. In *ACM SIGCOMM Computer Communication Review*, 2015.

[45] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *IEEE Symposium on Security and Privacy*, 2013.

[46] M. O'Neill, S. Ruoti, K. Seamons, and D. Zappala. TLS proxies: Friend or foe? In *ACM Internet Measurement Conference*, 2016.

[47] OPSWAT. Antivirus and compromised device report: Janurary 2015. https://www.opswat.com/resources/reports/antivirus-and-compromised-device-january-2015.

[48] T. Ormandy. Avast antivirus: X.509 error rendering command execution. https://bugs.chromium.org/p/project-zero/issues/detail?id=546&can=1&q=avast.

[49] V. Paxson. Bro: a system for detecting network intruders in real-time. In *7th USENIX Security Symposium*, 1998.

[50] A. Popov. Prohibiting RC4 cipher suites. RFC 7465.

[51] B. Riordan-Butterworth. Adopting encryption: The need for HTTPS. http://www.iab.com/adopting-encryption-the-need-for-https/.

[52] I. Ristic. HTTP client fingerprinting using SSL handshake analysis. https://blog.ivanristic.com/2009/06/http-client-fingerprinting-using-ssl-handshake-analysis.html.

[53] R. Seggelmann, M. Tuexen, and M. Williams. Transport layer security (TLS) and datagram transport layer security (DTLS) heartbeat extension. RFC 6520.

[54] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *SIGCOMM Computer Communication Review*, 2015.

[55] Statista. Mobile phone internet user penetration in South Korea from 2014 to 2019, 2016. http://www.statista.com/statistics/284204/south-korea-mobile-phone-internet-user-penetration/.

[56] Symantec. Trojan.Nurjax. https://www.symantec.com/security_response/writeup.jsp?docid=2014-121000-1027-99.

[57] U.S. Digital Analytics Program. The U.S. federal government's web traffic. https://analytics.usa.gov/.

[58] F. Valsorda. Komodia/superfish SSL validation is broken, February 2015. https://blog.filippo.io/komodia-superfish-ssl-validation-is-broken/.

[59] M. Vanhoef and F. Piessens. All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS. In *USENIX Security Symposium*, 2015.

[60] N. Weaver, C. Kreibich, M. Dam, and V. Paxson. Here be web proxies. In *International Conference on Passive and Active Network Measurement*, 2014.

[61] T.-F. Yen, Y. Xie, F. Yu, R. P. Yu, and M. Abadi. Host fingerprinting and tracking on the web: Privacy and security implications. In *Network and Distributed System Security Symposium*, 2012.